Logo, company name

Description automatically generated

**Module: 5DATA001C**

**Machine Learning & Data Mining**

**COURSE WORK REPORT**

**Tutorial Group** : Group B

**Student IIT ID** : 20212016

**Student UoW ID**  : W1871471

**Student Name**  : Pasindu Geevinda

Contents

[**Partitioning Clustering Part** 3](#_Toc134096702)

[**Subtask 1** 3](#_Toc134096703)

[Data Pre-processing – Scaling and Outlier Removal 3](#_Toc134096704)

[Determining the Number of Cluster Centres 4](#_Toc134096705)

[K-means Clustering Analysis 6](#_Toc134096706)

[Silhouette Plot 7](#_Toc134096707)

[**Subtask 2** 8](#_Toc134096708)

[Principal Component Analysis 8](#_Toc134096709)

[Determining the Number of Cluster Centres for New PCA-based dataset 11](#_Toc134096710)

[K-means Analysis with New PCA-dataset 13](#_Toc134096711)

[Silhouette Plot for new K-means Attempt 15](#_Toc134096712)

[Calinski-Harabasz index 16](#_Toc134096713)

[**Energy Forecasting Part** 17](#_Toc134096714)

[Defining Input Vectors for MLP Models in Electricity Load Forecasting 17](#_Toc134096715)

[A Study of Various Input Vector Combinations 18](#_Toc134096716)

[Why Normalising Data and How? 21](#_Toc134096717)

[MLP models 23](#_Toc134096718)

[Evidence 26](#_Toc134096719)

[Standard statistical indices 28](#_Toc134096720)

[Efficiency of models 30](#_Toc134096721)

[Efficiency 31](#_Toc134096722)

[**Appendix** 32](#_Toc134096723)

[**1st Objective (Partitioning Clustering)** 32](#_Toc134096724)

[**2nd Objective (MLP)** 37](#_Toc134096725)

[**References** 50](#_Toc134096726)

# **Partitioning Clustering Part**

## **Subtask 1**

### Data Pre-processing – Scaling and Outlier Removal

Outlier removal and scaling are 2 main pre-processing tasks that is necessary to be followed before working on machine learning algorithms.

Outliers are the data points with significant difference to other data points in the given data set. Outliers can caused by entry errors, measurement errors, etc.. Because of these outliers the machine learning model can become biased with overfitting and underfitting.

Scaling is necessary because it brings all features to a same scale, prevent any specific feature governing on the dataset. Scaling is important because the performance of the model may be impacted by the features' scale differences.

* **Outlier Removal**

By plotting a boxplot of the data frame we can clearly identify the outliers of each column in the data set.

Chart, box and whisker chart

Description automatically generated

Table 1 – Outliers

Table 2- After outliers removed

Chart, box and whisker chart

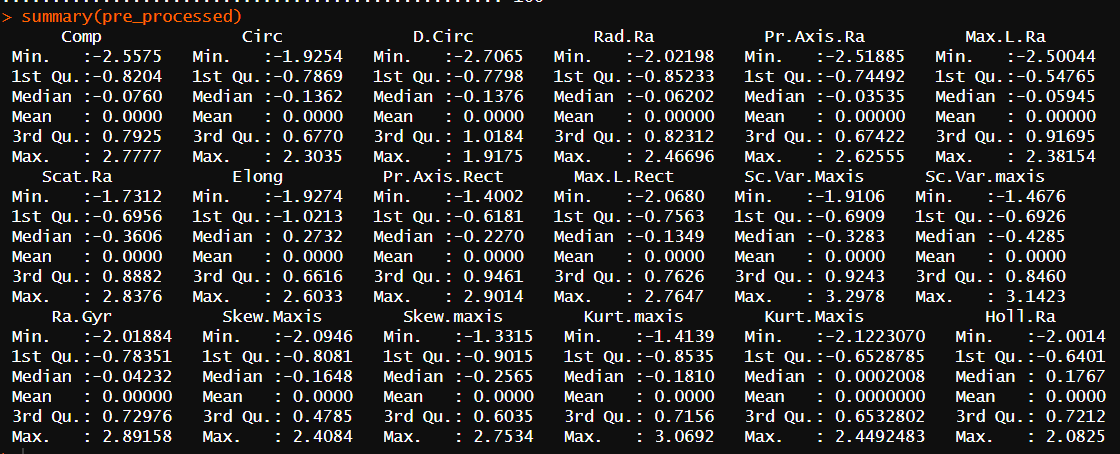
Description automatically generated

In the above figure, it shows that number of columns contains outliers. These outliers create a significant impact on the model. Therefore, in order remove the outlies I have used boxplots and removed all the rows which contain outliers.

By creating a boxplot after the outlier removal process, it gives a dataset with a minimum impact of outlers.

* **Normalization**

Min-max normalization is selected for the scaling process and by using it the data set is taken to a same scale from 1 to 0. Therefore, all the features are treated equally in the dataset.



### Determining the Number of Cluster Centres

NBclust, Elbow, Gap statistics and silhouette methods are used to determine the best number of clusters.

1. Elbow method

Chart, line chart

Description automatically generated

Elbow method suggest 3 as the optimal number of clusters.

1. Silhoutte plot

Chart, line chart

Description automatically generated

Silhoutte plot method suggest 2 as the optimal number of clusters.

1. Gap Statistic Algorithm

Chart, line chart

Description automatically generated

1. NBclust method

Graphical user interface

Description automatically generated with medium confidence

NBclust method suggest 3 as the best number of clusters.

Except silhouette plot method another 3 methods have suggested 3 as the best number of clusters. Therefore, I have decided to run k-means with k = 3.

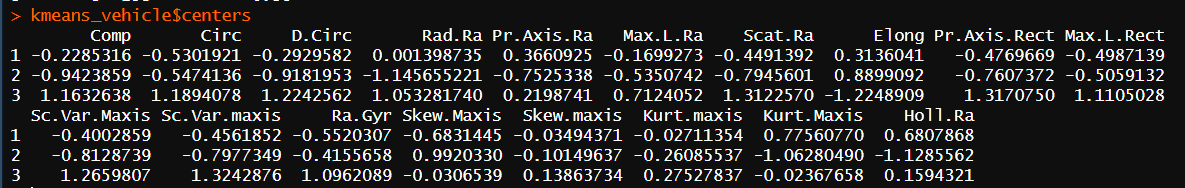
### K-means Clustering Analysis

* Cluster sizes =>
* Within Cluster Sum of Squares (WSS) =>
* Between Cluster Sum of Squares (BSS) =>
* Total Sum of Squares (TSS) =>
* (BSS / TSS) =>

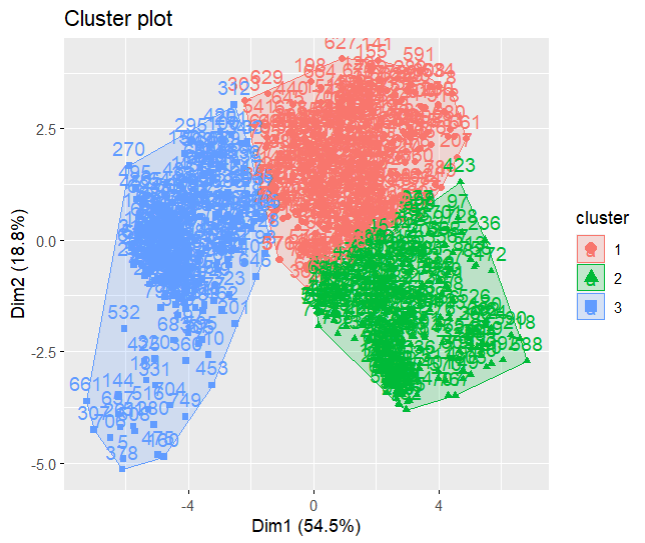
Graphical user interface

Description automatically generated with medium confidence

* Centers



Higher (BSS) indicates that the clusters are well separated and lower (WSS) indicates that the datapoints in the cluster are similar to each other.



### Silhouette Plot

Using this method, we can visualize how the data points are assigned to each cluster. I f the silhouette width is positive and large then the data points are well clustered and if they are closer to zero means they positioned between two clusters.



Text

Description automatically generated

As you can see in the above code and figures, All the 3 clusters are well distributed and have a positive large width. It is a good

## **Subtask 2**

### Principal Component Analysis

Principal Component Analysis is a method used to reduce high dimensionalities of large datasets by selecting the most important components available in the data set.

To perform principal component analysis we have to select the best few components which have a higher impact on the dataset.

Even in performing PCA reduces no. of components from the dataset. It’s recommended to remove outliers first to ensure that resulting principal components are not affected by any extreme values

Table 3- outliers

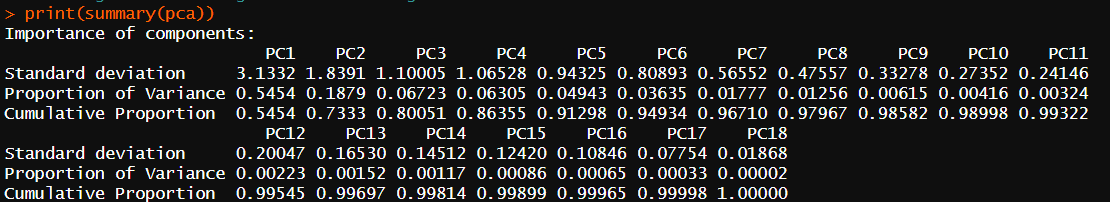
Chart, box and whisker chart

Description automatically generatedChart, box and whisker chart

Description automatically generated

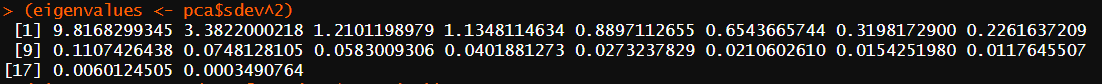
Table 4- outliers removed

By summary function we can get the following data related to PCA

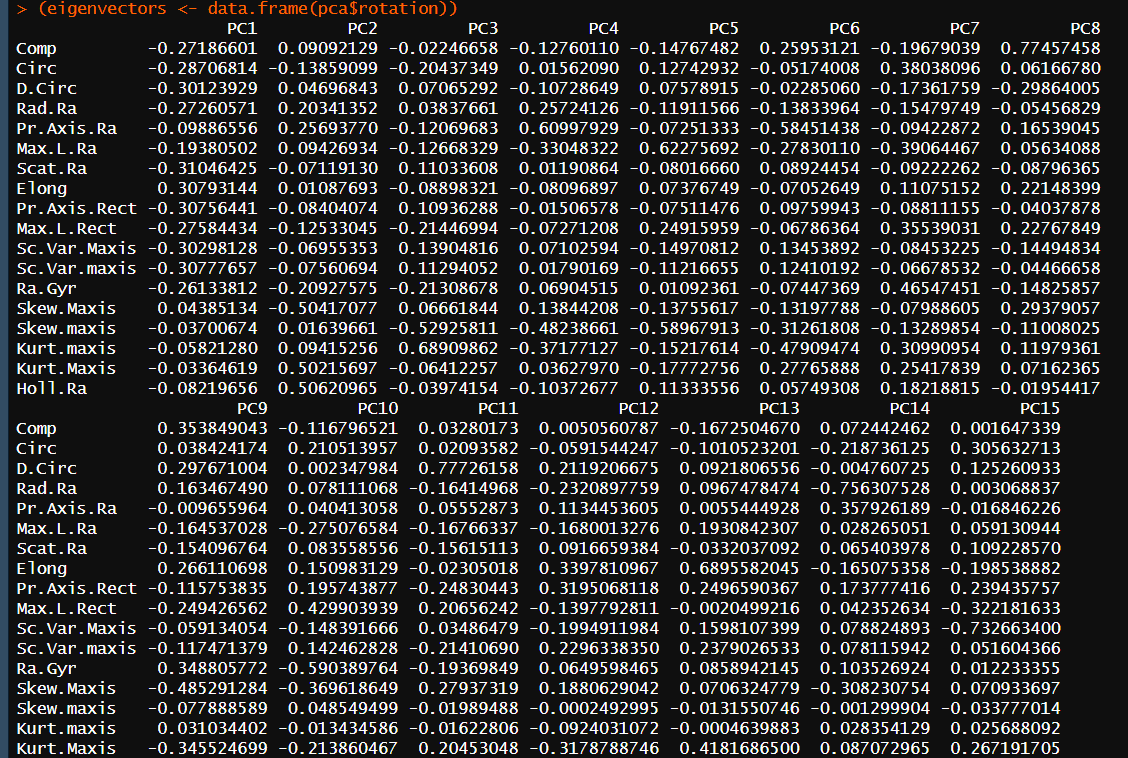


The above code output the **standard deviation, Proportion of variance, Cumulative Proportion are shown**. Here we have to select the minimum number of components which sum up least cumulative score to be greater than 92%.

***Eigen values***



***Eigen vectors***

******

I have selected the first 6 PCs of the dataset to perform principal component analysis.

The following scree plot depicts the variation of principal components against the Proportion of variance.

A picture containing chart

Description automatically generated

Using those 6 principal components transformed matrix is created. Text

Description automatically generated Using this transformed dataset, we are going to proceed to the following steps.

### Determining the Number of Cluster Centres for New PCA-based dataset

NBclust, Elbow, Gap statistics and silhouette methods are used to determine the best number of clusters.

#### **Elbow method**

Chart, line chart

Description automatically generated

Elbow method suggest 3 as the optimal number of clusters.

#### Silhoutte Plot

Chart, line chart

Description automatically generated

**Elbow method suggests 3 as the optimal number of clusters.**

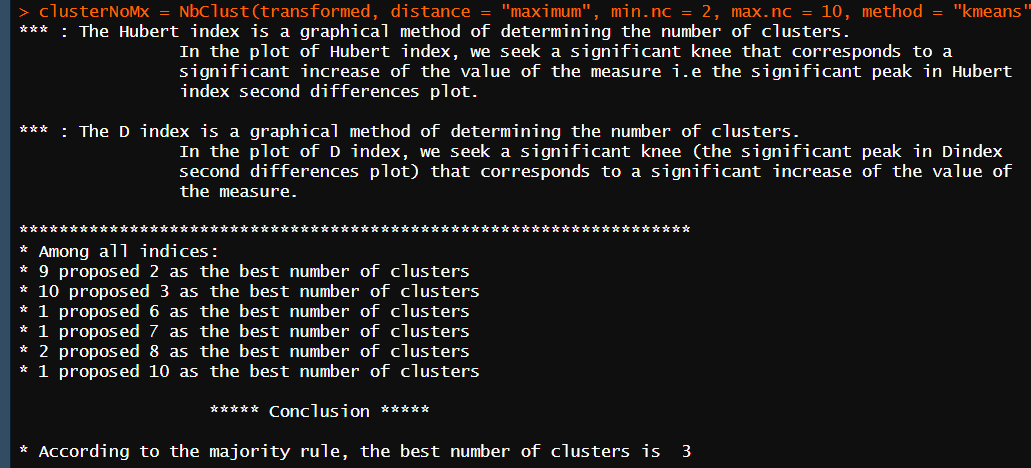
#### Gap statistic algorithm

Chart, line chart

Description automatically generated

Gap Statistic Algorithm suggest 3 as the optimal number of clusters.

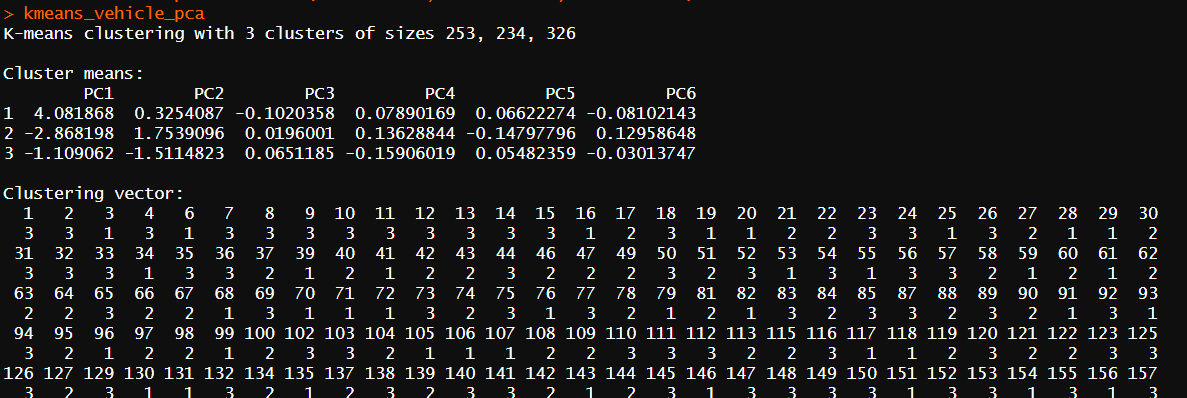
#### NBClust method



Except silhouette plot method another 3 methods have suggested 3 as the best number of clusters. Therefore , I have decided to run k-means with k = 3.

### K-means Analysis with New PCA-dataset

* Cluster sizes =>
* Within Cluster Sum of Squares (WSS) =>
* Between Cluster Sum of Squares (BSS) =>
* Total Sum of Squares (TSS) =>
* (BSS / TSS) =>



A screenshot of a computer

Description automatically generated with medium confidence

● Centers

Text

Description automatically generated

As you can see in the above outputs , this PCA based clusters have higher BSS and lower WSS which is a good indication of the quality of clusters.

Chart, scatter chart

Description automatically generated

### Silhouette Plot for new K-means Attempt

Same as the previous attempt, Silhouette plot is created to display the quality of the clusters in th PCA based dataset.

**Text

Description automatically generated**

Chart

Description automatically generated

Also, at this time the clusters are well formed and it confirms that we have reached the goal of creating high quality clusters.

### Calinski-Harabasz index

Also, at this time the clusters are well formed and it confirms that we have reached the goal of creating high quality clusters.

Also, at this time the clusters are well formed and it confirms that we have reached the goal of creating high quality clusters.

Chart, line chart

Description automatically generated

According to the above code snippet, we have a high CH index and it confirms that the clusters are well separated and consistent.

# **Energy Forecasting Part**

### Defining Input Vectors for MLP Models in Electricity Load Forecasting

In order to anticipate electricity load using MLP-NNs, an input vector must normally be defined that captures the pertinent details of the load's characteristics and data patterns. Depending on the individual technique and application domain, this input vector may contain a variety of input variables.

Utilizing autoregressive (AR) input variables, which use the load variable(s) prior values as input features, is a popular strategy. This method is predicated on the idea that future load values are highly linked with past values and can be forecasted using a linear or nonlinear combination of the prior data. The short-term load forecasting applications, where the emphasis is on forecasting the load values for the near future, frequently use the AR technique.

Other plans/methods can be used to define the input vector for MLP models in electricity load forecasting in addition to the AR approach. These consist of:

1. Time-of-day (TOD) variables: These input variables reflect how the load changes throughout the day, including during peak and off-peak hours, as well as transitional intervals. Long-term load forecasting can benefit from the usage of TOD variables, which can be utilized to capture the daily and weekly patterns in the load data.
2. Calendar-related events, such as holidays, weekends, and seasonal fluctuations, have an impact on the load, and these input variables reflect these effects. Calendar variables can be used in conjunction with TOD variables to capture the long-term trends in the load data.
3. Exogenous variables: These input variables reflect how external variables, such as demography, energy policies, and economic indicators, affect the load. Exogenous variables can be utilized in conjunction with the other input variables to capture the long-term trends and structural changes in the load data.

The application-specific parameters and the accessibility of pertinent data determine the choice of input variables. In general, the load characteristics and patterns can be more fully and accurately represented by combining a variety of input factors, which can improve forecasting performance.

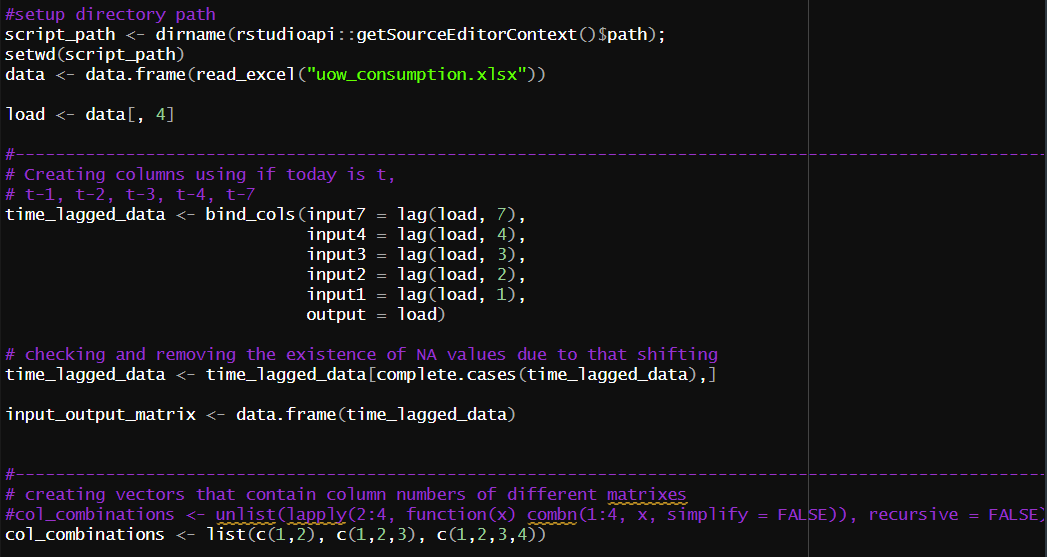
In our case, We used TOD variables for **Autoregressive** approach and for **NARX(nonlinear autoregressive exogenous),** we also used Exogenous variables.

### A Study of Various Input Vector Combinations

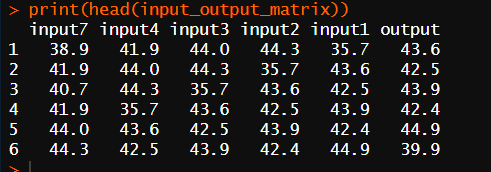
* **AR approach**

1. In the description have said "Experiment with various input vectors up to (t-4) level. According to literature, the electricity consumption forecast depends also on the (t-7) (i.e., one week before) value of the load.”. So, I can create many various combinations for matrixes using t-1 to t-4 and t-7. Firstly, I lagged the data as in the below.

**Code:**



**Output:**



1. Then that created a list of vectors that contain numbers of columns that I am going to create. And I combined that column to a new data frame. Like that, I created a number of 4 matrixes.

I used a list like below. There are lists of column numbers. As below code in screenshot while I go through that list I get a vector of column numbers and combined them to a new frame work. Like that I created 4 matrixes and store them in list named IO\_list

Col combination = {( 1, 2), (1, 2, 3), (1, 2, 3, 4) , (1, 2, 3, 4, 7)}

* **DF\_AR(1) :** Dataset which is responsible for predicting the next day’s (t) electricity consumption based on the (t-1) and (t-2) day electricity consumption data.
* **DF\_AR(2) :** Dataset which is responsible for predicting the next day’s (t) electricity consumption based on the (t-1), (t-2) and (t-3) day electricity consumption data.
* **DF\_AR(3) :** Dataset which is responsible for predicting the next day’s (t) electricity consumption based on the (t-1), (t-2), (t-3) and (t-4) day electricity consumption data.
* **DF\_AR(4) :** Dataset which is responsible for predicting the next day’s (t) electricity consumption based on the (t-1), (t-2), (t-3), (t-4) and (t-7) day electricity consumption data.

**Code:**

Text

Description automatically generated

1. **IO\_list** is containing the matrixes that have been created

**Output:**

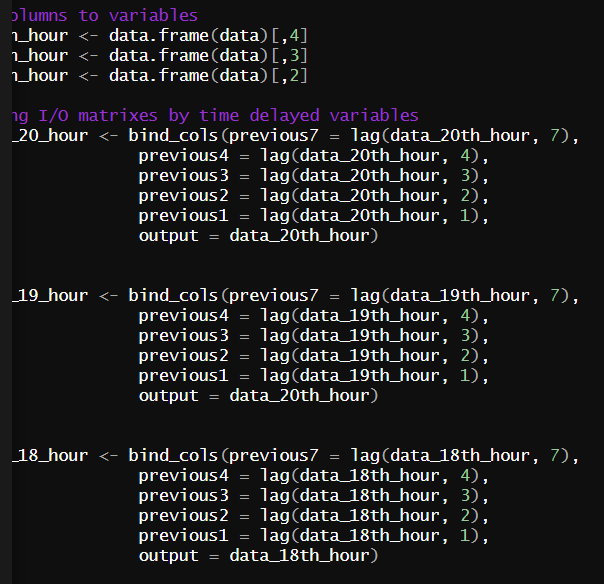
A picture containing text

Description automatically generated

* **NARX approach**

1. Firstly, loaded data and lagged them for separately as likely to AR approach.

**Code:**



1. And created 3 matrixes as in below description. But I created the matrixes also separately based on hours. When creating the models, we have input hours separately but with output of the previous hour.

Dataset’s name and description

* **DF\_** **NARX (1) :** Dataset which predicts the (t)th day’s electricity consumption based on (t-1), (t-2), (t-3), (t-4), (t-7) days’ and 18th hour electricity consumption.
* **DF\_** **NARX (2) :** Dataset which predicts the (t)th day’s electricity consumption based on (t-1), (t-2), (t-3), (t-4) days’ and 19th hour electricity consumption.
* **DF\_** **NARX (3) :** Dataset which predicts the (t)th day’s electricity consumption based on (t-1), (t-2), (t-3), (t-4), (t-7) days’ and 20th electricity consumption.
* 3 separated matrices are created for each of the above matrices

**Code:**

Text, calendar

Description automatically generated

### Why Normalising Data and How?

Data normalization is the act of converting a dataset's numerical values to a common scale without losing information or distorting the variations in the ranges of values. The performance and stability of machine learning models, particularly neural networks, are improved by normalizing data. The model can learn from the data and produce more accurate predictions by normalizing the data. Z-score normalization and min-max scaling are two popular normalizing methods that change the original data into a new representation with a particular distribution and range of values. The data and the needs of the machine learning model being employed determine the normalization strategy to be utilized. Normalization ensures that all input variables have the same scale and distribution, which prevents certain variables from dominating the training process and biasing the model toward them. MLPs require input variables that range between 0 and 1 or -1 and 1, and normalization transforms the data into a zero mean and unit variance. This transformation is applied to each input variable independently, ensuring that all variables have the same scale and distribution. Overall, data normalization is crucial in energy prediction for MLP models, as it can improve accuracy and stability, and ensure efficient convergence during training.

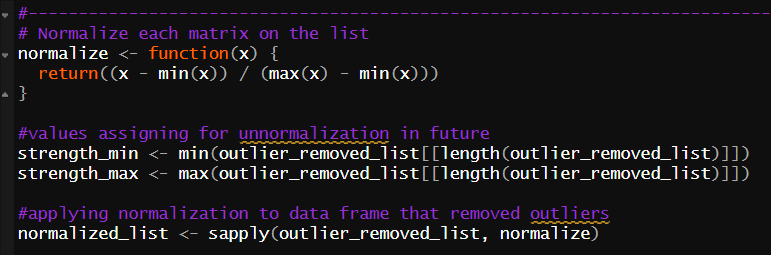
Normalization ensures that all input variables have the same scale and distribution, which prevents certain variables from dominating the training process and biasing the model toward them. MLPs require input variables that range between 0 and 1 or -1 and 1, and normalization transforms the data into a zero mean and unit variance. This transformation is applied to each input variable independently, ensuring that all variables have the same scale and distribution. Overall, data normalization is crucial in energy prediction for MLP models, as it can improve accuracy and stability, and ensure efficient convergence during training.

* **AR approach**

This code defines a function called "normalize" that takes a matrix as an input and returns the normalized version of that matrix. It then assigns the minimum and maximum values of the last matrix in a list called "outlier\_removed\_list" to variables called "strength\_min" and "strength\_max", respectively. Finally, the code applies the "normalize" function to each matrix in the "outlier\_removed\_list" and saves the resulting normalized matrices in a new list called "normalized\_list".

1. Strength\_max and strength\_min, will be used for unnormalized the values of matrixes in the future

**Code:**



1. Unnormalize function is here. We have to compare output and original data because to check the model trained well or not. When we training model we have used normalized data. So we can’t compare the values of output and original data caused of different range. So after got predicted data we are unnormalized it and compare

**Code:**

A screenshot of a computer

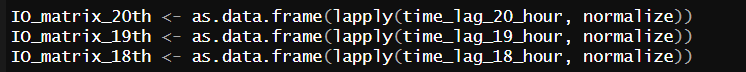
Description automatically generated with medium confidence

Table 5unnormalization

* **NARX approach**

Before creating matrixes hours by hours, we normalized it. After normalization we divided into 3 matrixes, all 3 matrices that have created by lagging based on hours.

**Code:**



And denormalized the output of 6 models, after predicted by 3 NARX models, because we have to test is compare and evaluate in upcoming process

**Code:**

Graphical user interface, text, application, chat or text message

Description automatically generated

### MLP models

* **AR approach**

It is essential to experiment with different input vectors and internal network structures during the MLP models' training phase. To find the best model for the given problem, this entails examining various combinations of input variables, hidden layers, nodes, output types (linear/nonlinear), and activation functions. Finding a model that can correctly forecast future energy consumption based on historical data is the goal. It is possible to learn more about how these various elements affect the overall performance of the MLP by testing multiple models with various configurations.

1. Firstly, while going through the list of matrixes I divided all matrixes to training (row num 1:380) and testing matrixes(row num 380:all).

**Code:**

**Text

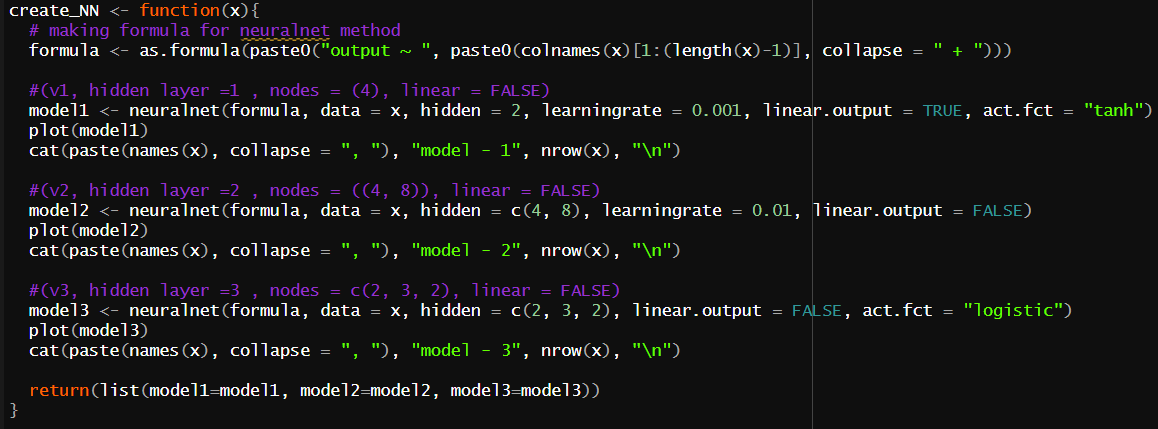
Description automatically generated**

1. Then I created a function for creating 3 model for given matrix. And created a list using that 3 models. And returned it

Specifications of models:

* model1: 1 hidden layer, 2 hidden layer nodes, leanear output, activation function = "tanh"
* model2: 2 hidden layer, (4, 8) hidden layer nodes, nonleanear output
* model3: 3 hidden layer, (2,3,2) hidden layer nodes, leanear output, activation function = "logistic"

**Code:**



1. Using this code while going through matrix we are creating models and stored the list of NN models named nn variable. After it while going through that created model, we are predicting output of testing data that we have created before. And add that predicted data into a list named nn\_result. We have created a method name evaluation and by that we evaluate models and stored the results in a list name evaluation. In the end of the loop we created a list name comparison\_table and stored tables that combined columns of original and predicted data.

**Code:**

Text

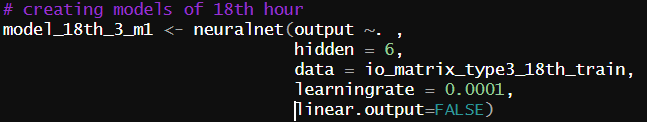
Description automatically generated

* **NARX approach**

1. To experiment with different output vectors, I have created 3 number of matrixes. But in NARX approach it is essential to learn while predicting. Exogenous means we have to consider external inputs like 19th and 18th hours also. As I said in previous topic I have created 3 matrixes for each hour separately by lagging.

Firstly, I have selected the input matrix of 18th hour and input it to the model. (Let’s consider NARX type 1 model with matrix number 3)

**Code:**



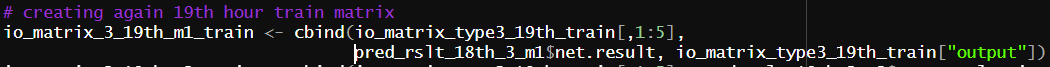
1. Then predicted the output and the predicted output column have combined with 19th input matrix 3. (Remember all matrixes are considering same days but different hours). Add it into next same type model to as a input and predict 19th hour of day **t**

**Code:**

Predicting the output



Combining output with 3type matrix of 19th hour.



Give input and create the model of 19th hour

Text

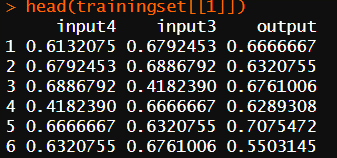
Description automatically generated

1. Likewise previous step we are predicting the 20th hour’s output and train the model. Now we have created NARX model combined with 3 neural network models.
2. So now we have created 6 number of NARX models. First we have to evaluate all created models using standard statistical indices and choose best model.

### Evidence

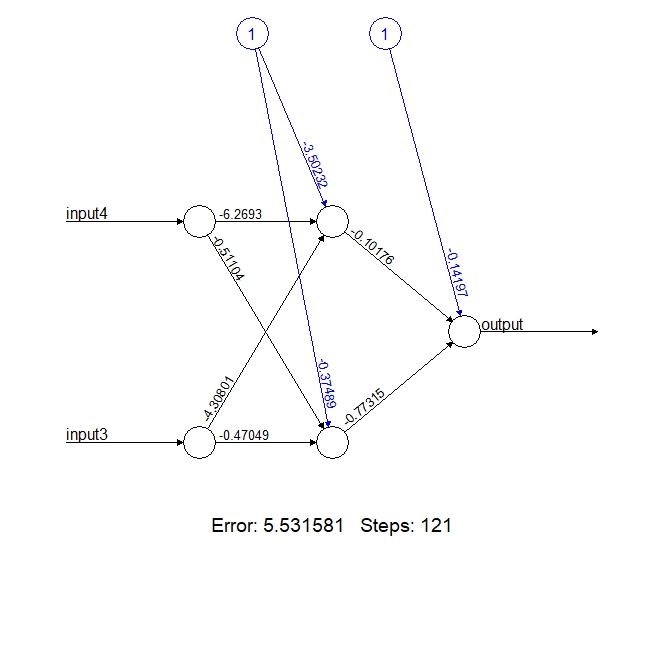
* **AR approach**

1. This is the Input output matrix of (t-3) and (t-4). (There is remaining 4 matrixes)



1. Today’s data predicting model base on matrix (t-3) and (t-4) day’s energy consumption data.

(this is the model 1 and there is remaining 11 models)



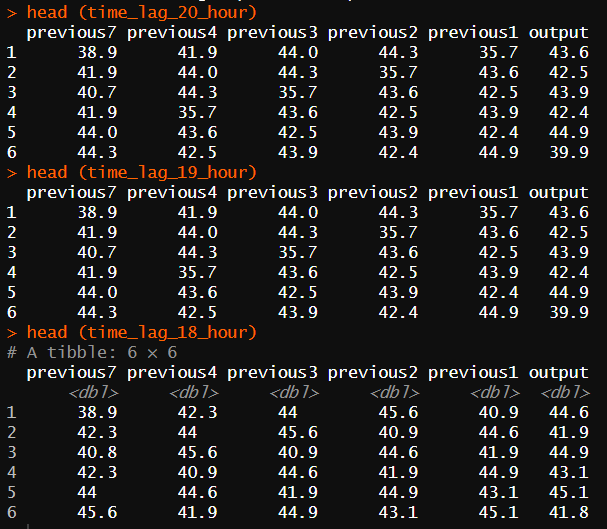
1. This is the table of containing the predicted and original outputs comparison data

Text

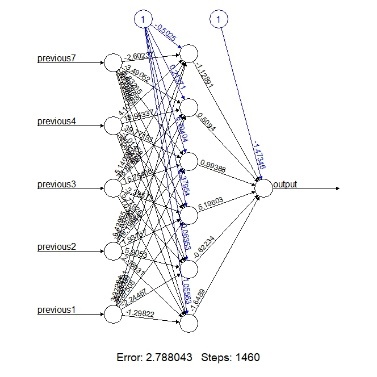
Description automatically generated

* **NARX approach**

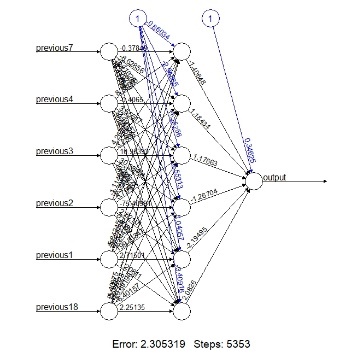
1. This is the matrixes of matrix type 3 base on hours



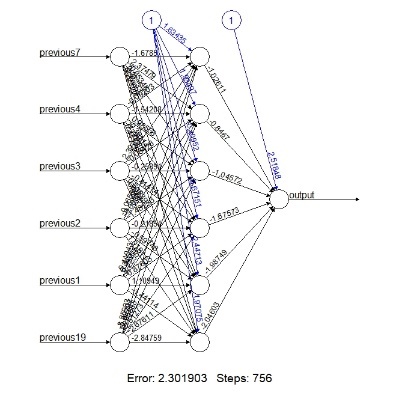
1. This is the NARX model type 1. It has 3 same types of models for hours separately.
   1. Model created for 18th hour.



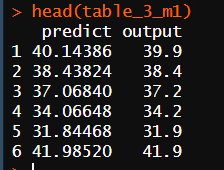
* 1. Model created for 19th hour.



* 1. Model created for 20th hour.



3. This is the table of containing the predicted and original outputs comparison data



### Standard statistical indices

Metrics used to assess the effectiveness of a predictive model, such as a machine learning model, include standard statistical indices. These indices offer details on the precision and accuracy of the model's predictions in relation to the observed values.

Standard statistical indices that are frequently used in forecasting models include:

* Root Mean Squared Error (RMSE): A measure of the difference between predicted values and actual values. It is the square root of the average of the squared difference between the predicted and actual values.

**Code: **

* Mean Absolute Error (MAE): A measure of absolute differences between predicted values and actual values.

**Code:** 

* Mean Absolute Percentage Error (MAPE): A measure of the percentage difference between predicted and actual values. It is calculated by taking the absolute difference between the predicted and actual values, dividing it by the actual value, and then multiplying by 100.

**Code:**

****

* Symmetric Mean Absolute Percentage Error (sMAPE): A measure of the percentage difference between predicted and actual values, calculated as the average of the absolute differences between predicted and actual values, divided by the average of the actual and predicted values, then multiplied by 100

**Code:**

****

MLP Neural Network Analysis - Comparison of NN

* Typically, a good model will have lower values of these indices, indicating that the model's predictions are closer to the actual values. Therefore, you can select the model with the lowest overall values of these indices as the best model.
* Best model => **Red**

Best one hidden layer => light blue

Best two hidden layer => light green

1. MLP Neural Network Analysis for AR approach

Dataset’s name and description

* **DF\_AR(1) :** Dataset which is responsible for predicting the next day’s (t) electricity consumption based on the (t-1) and (t-2) day electricity consumption data.
* **DF\_AR(2) :** Dataset which is responsible for predicting the next day’s (t) electricity consumption based on the (t-1), (t-2) and (t-3) day electricity consumption data.
* **DF\_AR(3) :** Dataset which is responsible for predicting the next day’s (t) electricity consumption based on the (t-1), (t-2), (t-3) and (t-4) day electricity consumption data.
* **DF\_AR(4) :** Dataset which is responsible for predicting the next day’s (t) electricity consumption based on the (t-1), (t-2), (t-3), (t-4) and (t-7) day electricity consumption data.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Dataset Name (contains input vectors) | No of hidden layers and nodes in each layer | Learning rate | Activation Function | Testing Performance Analysis  (standard statistical indices) | | | |
| RMSE | MAE | MAPE | sMAPE |
| DF\_AR(1) | 1 (2) | 0.001 | tanh | 4.083231 | 3.372549 | 8.465277 | 8.536582 |
| DF\_AR(1) | 2 (4, 8) | 0.01 | logistic | 4.07867 | 3.370176 | 8.46253 | 8.528329 |
| DF\_AR(1) | 3 (2, 3, 2) | default | logistic | 4.0736 | 3.366842 | 8.46381 | 8.519723 |
| DF\_AR(2) | 1 (2) | 0.001 | tanh | 3.751716 | 3.065093 | 7.674552 | 7.761144 |
| DF\_AR(2) | 2 (4, 8) | 0.01 | logistic | 3.768574 | 3.077081 | 7.705419 | 7.796324 |
| DF\_AR(2) | 3 (2, 3, 2) | default | logistic | 3.725462 | 3.039135 | 7.693323 | 7.721715 |
| DF\_AR(3) | 1 (2) | 0.001 | tanh | 3.688702 | 2.91433 | 7.394048 | 7.474169 |
| DF\_AR(3) | 2 (4, 8) | 0.01 | logistic | 3.697794 | 3.000662 | 7.596264 | 7.675258 |
| DF\_AR(3) | 3 (2, 3, 2) | default | logistic | 3.604125 | 2.928061 | 7.407 | 7.495187 |
| DF\_AR(4) | 1 (2) | 0.001 | tanh | 2.944437 | 2.32898 | 5.851045 | 5.907481 |
| DF\_AR(4) | 2 (4, 8) | 0.01 | logistic | 3.501257 | 2.705737 | 6.885238 | 6.939866 |
| DF\_AR(4) | 3 (2, 3, 2) | default | logistic | 2.965938 | 2.302971 | 5.792422 | 5.852396 |

Table 6MLP Neural Network Analysis for AR approach

1. MLP Neural Network Analysis for NARX approach

Dataset’s name and description

* **DF\_** **NARX (1) :** Dataset which predicts the (t)th day’s electricity consumption based on (t-1), (t-2), (t-3), (t-4), (t-7) days’ and 18th hour electricity consumption.
* **DF\_** **NARX (2) :** Dataset which predicts the (t)th day’s electricity consumption based on (t-1), (t-2), (t-3), (t-4) days’ and 19th hour electricity consumption.
* **DF\_** **NARX (3) :** Dataset which predicts the (t)th day’s electricity consumption based on (t-1), (t-2), (t-3), (t-4), (t-7) days’ and 20th electricity consumption.
* 3 separated matrices are created for each of the above matrices

Type of model in NARX:

NARX is made up of 2 models. NARX is made up of one model and three other models that are similar to each other. Only the input given to it changes.

Below is an explanation of one of the three identical models within a NARX model.

1. One hidden layer, 6 nodes, learningrate = 0.0001, non-linear output, activation function = “tanh”
2. Two hidden layer, (4, 8) nodes, learningrate = 0.01, non-linear output, activation function = “logistic”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset Name | type | Testing Performance Analysis  (standard statistical indices) | | | |
| RMSE | MAE | MAPE | sMAPE |
| DF\_NARX (1) | 1 | 0.624601515517192 | 0.499205435941965 | 0.0126905812361652 | 0.0127520752474337 |
| DF\_NARX (1) | 2 | 0.320453274695244 | 0.24711143986778 | 0.00606517673313126 | 0.00607311549472805 |
| DF\_NARX (2) | 1 | 0.825561444124725 | 0.588367706125024 | 0.014781104935917 | 0.0146327476479016 |
| DF\_NARX (2) | 2 | 0.269452207849168 | 0.207693951672132 | 0.00518673038926825 | 0.00519526748523835 |
| DF\_NARX (3) | 1 | 0.259332990972099 | 0.196320437290773 | 0.196320437290773 | 0.00482574165792954 |
| DF\_NARX (3) | 2 | 2.93053228191011 | 2.11567571441366 | 0.0559182037539554 | 0.059278778060077 |

### Efficiency of models

Assume that we have developed two neural network models. One with one hidden layer and the other with two hidden layers. We have compared their test performance using different statistical indices. Now think, two models have similar performance in terms of statistical indices, but the one with two hidden layers has slightly better overall performance.

To check the efficiency of these models, we need to calculate the total number of weight parameters for each model. The total number of weight parameters can be calculated by multiplying the number of neurons in each layer (excluding the input layer) and adding the products for all layers, including the output layer.

As you can see above table, by comparing the standard statistical indices best best one-hidden layer network is DF\_AR(4)’s model 1 and two-hidden layer is DF\_AR(4)’s model 2. Let’s calculate total number of weight parameter(G) for each other.

DF\_AR(4) => {t-1, t-2, t-3, t-4, t-7}

G1(DF\_AR(4) model 1)  => (5\*1) + (1\*1) = 6

G2(DF\_AR(4) model 2)  => (5\*4) + (4\*8) + (8\*1) = 60

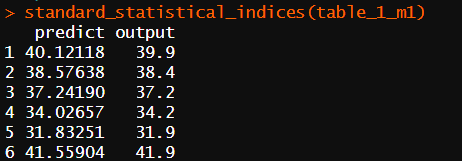
G1 < G2 Therefore, we may conclude that the two-hidden layer model is more preferable in this case.

### Efficiency

The best hidden layer 1 MLP networks are highlighted in green, and the best hidden layer 2 MLP networks are highlighted in red, when comparing the "AR" and "NARX" MLP Neural Network tables. It is noteworthy that the RMSE, MAE, and MAPE values produced by the NARX approach are comparatively low. As a result, for this particular time series problem, the NARX approach performs better than the AR approach.

Additionally, the NARX Network comparison table's best MLP network with two hidden layers displays relatively low values for RMSE, MAE, and MAPE. This demonstrates that this MLP Neural Network was the top network in both tables among all networks.

And the following picture shows the details about the (desired output vs predicted output) from the testing dataset.



# **Appendix**

## **1st Objective (Partitioning Clustering)**

1st subtask

library(NbClust)

library(dplyr)

library(tidyr)

library(ggplot2)

library(factoextra)

library(cluster)

script\_path <- dirname(rstudioapi::getSourceEditorContext()$path);

setwd(script\_path)

df <- read.csv("vehicles.csv", row.names = 1)

################# PART A. Scale data using z-score normalization ###############

df\_data <- df[1:18][complete.cases(df[1:18]),]

################## Identify ourliers using boxplots and remove them ############

# Create empty list to store outlier row indices

outlier\_rows <- list()

# Loop through columns and extract outlier row indices

boxplot(df\_data, plot = TRUE)

for (col in names(df\_data)) {

outliers <- boxplot(df\_data[[col]], plot = FALSE)$out

row\_indices <- which(df\_data[[col]] %in% outliers)

if (length(row\_indices) > 0) {

outlier\_rows[[col]] <- as.integer(row\_indices)

}

}

outlier\_rows <- unique(unlist(outlier\_rows))

df\_outliers\_removed <- df\_data[-outlier\_rows, ]

boxplot(df\_outliers\_removed, plot = TRUE)

################################ normalizing data ##############################

#function to normalize

normalize <- function(x){

return((x - min(x))/(max(x) - min(x)))

}

# apply normalization

pre\_processed <- data.frame(lapply(df\_outliers\_removed, scale))

pre\_processed

########################### PART B. cchecking best K value #####################

### NbClust Method

set.seed(30)

clusterNoE = NbClust(pre\_processed, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans", index = "all")

clusterNoM = NbClust(pre\_processed, distance = "manhattan", min.nc = 2, max.nc = 10, method = "kmeans", index = "all")

clusterNoMx = NbClust(pre\_processed, distance = "maximum", min.nc = 2, max.nc = 10, method = "kmeans", index = "all")

### Elbow Method

set.seed(42)

fviz\_nbclust(pre\_processed, kmeans, method = 'wss')

#WSS = sapply(k, function(k) {kmeans(pre\_processed, centers=k)$tot.withinss})

#plot(k, WSS, type="l", xlab= "Number of k", ylab="Within sum of squares")

### silhouette method

fviz\_nbclust(pre\_processed, kmeans, method = 'silhouette')

### Gap Statistic Algorithm

fviz\_nbclust(pre\_processed, kmeans, method = 'gap\_stat')

######################## PART C. k-means clustering ############################

k=3

kmeans\_vehicle <- kmeans(pre\_processed, centers = k, nstart = 10)

kmeans\_vehicle

#visualizing the clusters

fviz\_cluster(kmeans\_vehicle, data = pre\_processed)

wss <- kmeans\_vehicle$tot.withinss

bss <- kmeans\_vehicle$betweenss

tss <- kmeans\_vehicle$totss

sprintf("Within-cluster sum of square %.2f ", wss)

sprintf("Between sum of square %.2f", bss)

sprintf("Total sum of squares %.2f", tss)

kmeans\_vehicle$centers

################ PART D. Internal metrics for cluster evaluation ###############

sil <- silhouette(kmeans\_vehicle$cluster, dist(pre\_processed))

fviz\_silhouette(sil)

2nd subtask – PCA

library(NbClust)

library(dplyr)

library(tidyr)

library(ggplot2)

library(factoextra)

library(cluster)

script\_path <- dirname(rstudioapi::getSourceEditorContext()$path);

setwd(script\_path)

################################ Substask 2 ####################################

############################### PART E. PCA ####################################

vehicleDF <- read.csv("vehicles.csv", row.names = 1)[1:18]

# Create empty list to store outlier row indices

outlier\_rows <- list()

# Loop through columns and extract outlier row indices

boxplot(vehicleDF, plot = TRUE)

for (col in names(vehicleDF)) {

outliers <- boxplot(vehicleDF[[col]], plot = FALSE)$out

row\_indices <- which(vehicleDF[[col]] %in% outliers)

if (length(row\_indices) > 0) {

outlier\_rows[[col]] <- as.integer(row\_indices)

}

}

outlier\_rows <- unique(unlist(outlier\_rows))

vehicleDF <- vehicleDF[-outlier\_rows, ]

boxplot(vehicleDF, plot = TRUE)

# compute variance of each variable

apply(vehicleDF, 2, var)

# Apply PCA using prcomp()

pca <- prcomp(vehicleDF, center = TRUE, scale = TRUE)

pca\_summary <- summary(pca, loadings=TRUE)

print(summary(pca))

# Extract the eigenvalues and eigenvector

(eigenvalues <- pca$sdev^2)

(eigenvectors <- data.frame(pca$rotation))

# Extract the cumulative proportion values

proportion\_of\_Variance <- data.frame(pca\_summary$importance)[2,]

plot(1:length(proportion\_of\_Variance), proportion\_of\_Variance, type = "b",

xlab = "Principal Component", ylab = "Proportion of Variance Explained",

main = "Scree Plot")

abline(h = 0.92, lty = 2)

# if we used the first 6 components we would be able to account for >92% of total variance in the data.

# we are avoiding including too many components, which could result in overfitting or loss of interpretability.

# find the index of PC's that crosses the 0.92(92%) threshold

pc\_indexes = list()

sorted\_values <- sort(pca\_summary$importance[2,], decreasing = TRUE)

sum <- 0

row <- proportion\_of\_Variance[1,]

index\_of\_list <- 1

for (i in sorted\_values) {

sum <- sum + i

index <- which(row == i)

pc\_indexes[[index\_of\_list]] <- index

index\_of\_list <- index\_of\_list +1

if(sum>=0.92){

break

}

}

pc\_indexes <- as.integer(unlist(pc\_indexes))

transformed <- data.frame(-pca$x)[pc\_indexes]

########################### PART F. cchecking best K value #####################

### NbClust Method

set.seed(30)

clusterNoE = NbClust(transformed, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans", index = "all")

clusterNoM = NbClust(transformed, distance = "manhattan", min.nc = 2, max.nc = 10, method = "kmeans", index = "all")

clusterNoMx = NbClust(transformed, distance = "maximum", min.nc = 2, max.nc = 10, method = "kmeans", index = "all")

### Elbow Method

set.seed(42)

fviz\_nbclust(transformed, kmeans, method = 'wss')

#WSS = sapply(k, function(k) {kmeans(transformed, centers=k)$tot.withinss})

#plot(k, WSS, type="l", xlab= "Number of k", ylab="Within sum of squares")

### silhouette method

fviz\_nbclust(transformed, kmeans, method = 'silhouette')

### Gap Statistic Algorithm

fviz\_nbclust(transformed, kmeans, method = 'gap\_stat')

########################### PART G. Perform k-means ############################

k=3

kmeans\_vehicle\_pca <- kmeans(transformed, centers = k, nstart = 10)

kmeans\_vehicle\_pca

wss <- kmeans\_vehicle\_pca$tot.withinss

bss <- kmeans\_vehicle\_pca$betweenss

tss <- kmeans\_vehicle\_pca$totss

sprintf("Within-cluster sum of square %.2f ", wss)

sprintf("Between sum of square %.2f", bss)

sprintf("Total sum of squares %.2f", tss)

kmeans\_vehicle\_pca$centers

################ PART D. Internal metrics for cluster evaluation ###############

sil\_pca <- silhouette(kmeans\_vehicle\_pca$cluster, dist(transformed))

fviz\_silhouette(sil\_pca)

####################### PART D. Calinski-Harabasz Index ########################

library(fpc)

ch\_index <- calinhara(transformed, kmeans\_vehicle\_pca$cluster)

sprintf("Calinski-Harabasz Index = %.2f", ch\_index)

#visualizing the clusters

fviz\_cluster(kmeans\_vehicle\_pca, data = transformed)

library('fpc')

set.seed(789)

round(calinhara(vehicleDF,kmeans\_vehicle\_pca$cluster), digits=2)

# Number of clusters to evaluate

k <- 2:10

# Empty vector for storing Calinski-Harabasz index values

ch\_values <- vector("numeric", length(k))

# Compute the Calinski-Harabasz index for each number of clusters

for (i in 1:length(k)) {

km <- kmeans(vehicleDF, centers = k[i], nstart = 10)

ch\_values[i] <- calinhara(vehicleDF, km$cluster)

}

# Plot the Calinski-Harabasz index values

plot(k, ch\_values, type = "b", xlab = "Number of Clusters", ylab = "Calinski-Harabasz Index", main = "Calinski-Harabasz Index vs. Number of Clusters")

## **2nd Objective (MLP)**

1st subtask

library(readxl)

library(dplyr)

#install first the package before calling it via the library command

#setup directory path

script\_path <- dirname(rstudioapi::getSourceEditorContext()$path);

setwd(script\_path)

data <- data.frame(read\_excel("uow\_consumption.xlsx"))

load <- data[, 4]

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

# Creating columns using if today is t,

# t-1, t-2, t-3, t-4, t-7

time\_lagged\_data <- bind\_cols(input7 = lag(load, 7),

input4 = lag(load, 4),

input3 = lag(load, 3),

input2 = lag(load, 2),

input1 = lag(load, 1),

output = load)

# checking and removing the existence of NA values due to that shifting

time\_lagged\_data <- time\_lagged\_data[complete.cases(time\_lagged\_data),]

input\_output\_matrix <- data.frame(time\_lagged\_data)

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

# creating vectors that contain column numbers of different matrixes

#col\_combinations <- unlist(lapply(2:4, function(x) combn(1:4, x, simplify = FALSE)), recursive = FALSE)

col\_combinations <- list(c(1,2), c(1,2,3), c(1,2,3,4))

# Create list of matrices for each combination

IO\_list <- list()

for (cols in col\_combinations) {

print(cols)

#selecting 2:5 columns because we are creating matrixes combinations only using from t-1 to t-4

#after selecting data.frame that contains t-1:t-4, selecting columns that have column numbers in col\_combinations

mat <- as.matrix(input\_output\_matrix[2:5][, cols])

#append output column to selected column from before step

mat <- cbind(mat, input\_output\_matrix[6])

print(head(mat))

# This line concatenates the column names of the 'mat' matrix and stores the result in a new list element

IO\_list[[paste0(cols, collapse = "")]] <- mat

}

# adding full lagged IO\_matrix, because it is the only one that can combine with t-7

IO\_list[[length(IO\_list)+1]] <- input\_output\_matrix

print(head(input\_output\_matrix))

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

# Remove outliers from each matrix in the list

outlier\_removed\_list <- list()

for (i in IO\_list) {

# Create empty list to store outlier row indices

outlier\_rows <- list()

# Loop through columns and extract outlier row indices

boxplot(i, plot = TRUE)

for (col in names(i)) {

# while going through columns in the matrix collecting outlier values

outliers <- boxplot(i[[col]], plot = FALSE)$out

# get rows indicate to the outlier values

row\_indices <- which(i[[col]] %in% outliers)

#if there isnt any outlier it wont go through this process

if (length(row\_indices) > 0) {

outlier\_rows[[col]] <- as.integer(row\_indices)

}

}

#checking rows that have the values on df\_outliers. if it's in there the row will remove

outlier\_rows <- unlist(unique(outlier\_rows))

removed <- i

if(!is.null(outlier\_rows)) {removed <- i[-outlier\_rows, ]}

#adding to a new list that removed outliers

outlier\_removed\_list[[(length(outlier\_removed\_list)+1)]] <- as.data.frame(removed)

}

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

# Normalize each matrix on the list

normalize <- function(x) {

return((x - min(x)) / (max(x) - min(x)))

}

#values assigning for unnormalization in future

strength\_min <- min(outlier\_removed\_list[[length(outlier\_removed\_list)]])

strength\_max <- max(outlier\_removed\_list[[length(outlier\_removed\_list)]])

#applying normalization to data frame that removed outliers

normalized\_list <- sapply(outlier\_removed\_list, normalize)

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

## creating matrix for train NN and test it

trainingset <- list()

testingset <- list()

for (i in normalized\_list) {

#create dataframe that contain first 380 rows to train the NN

trainingset[[(length(trainingset)+1)]] <- as.data.frame(i[1:380,])

#create dataframe that contain after 380 rows to test the NN

test.frame <- as.data.frame(i[380:nrow(i),])

#setting up row value to start from 0

row.names(test.frame) <- 1:nrow(test.frame)

testingset[[(length(testingset)+1)]] <- test.frame

}

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

library(neuralnet)

library(grid)

library(MASS)

set.seed(5056)

create\_NN <- function(x){

# making formula for neuralnet method

formula <- as.formula(paste0("output ~ ", paste0(colnames(x)[1:(length(x)-1)], collapse = " + ")))

#(v1, hidden layer =1 , nodes = (4), linear = TRUE)

model1 <- neuralnet(formula, data = x, hidden = 2, learningrate = 0.001, linear.output = TRUE, act.fct = "tanh")

plot(model1)

cat(paste(names(x), collapse = ", "), "model - 1", nrow(x), "\n")

#(v2, hidden layer =2 , nodes = ((4, 8)), linear = FALSE)

model2 <- neuralnet(formula, data = x, hidden = c(4, 8), learningrate = 0.01, linear.output = FALSE)

plot(model2)

cat(paste(names(x), collapse = ", "), "model - 2", nrow(x), "\n")

#(v3, hidden layer =3 , nodes = c(2, 3, 2), linear = FALSE)

model3 <- neuralnet(formula, data = x, hidden = c(2, 3, 2), linear.output = FALSE, act.fct = "logistic")

plot(model3)

cat(paste(names(x), collapse = ", "), "model - 3", nrow(x), "\n")

return(list(model1=model1, model2=model2, model3=model3))

}

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

standard\_statistical\_indices <- function(testing\_data, compute\_results){

#testing data is list and compute results is double.

testing\_data <- unlist(testing\_data)

rmse <- sqrt(mean((testing\_data - compute\_results)^2))

mae <- mean(abs(testing\_data - compute\_results))

mape <- mean(abs((testing\_data - compute\_results) / testing\_data)) \* 100

smape <- mean(200 \* abs(testing\_data - compute\_results) / (abs(testing\_data) + abs(compute\_results)))

return(list(rmse=rmse, mae=mae, mape=mape, smape=smape))

}

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

# to unnormalize the columns the data

unnormalize <- function(x, min, max) {

return( (max - min)\*x + min )

}

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

# store the outputs = {NN models, predicted resualts, standard\_statistical\_indices= evaluation,comparison\_table}

nn\_models\_result <- list()

# testing the dataset and stored the results on nn\_models\_result list

for (i in 1:length(trainingset)) {

#going through training data matrixes on list trainingset

#creating models base on each matrixes to 3 NN models using create\_NN() method

training.data <- trainingset[[i]]

nn <- create\_NN(training.data)

#going through testing data matrixes on list testing

#get the column of original output column before normalizing

testing.data <- testingset[[i]]

output.data <- outlier\_removed\_list[[i]][380:nrow(outlier\_removed\_list[[i]]),]["output"]

#set first rows number as 1 cause it was 381

row.names(output.data) <- NULL

#After predicting using compute method returned object will store here

#evaluated values{ RMSE, MAE, MAPE and sMAPE }

# original output and predicted output will store here

nn\_result <- list()

evaluation <- list()

comparison\_table <- list()

#Going through each model of matrixes

for (model in nn) {

# give testing matrixes as inputs and getting predited data object as output

# store each outputs of matrixes on nn\_result() list

net.results <- compute(model, testing.data[, 1:(length(testing.data)-1)])

nn\_result[[length(nn\_result)+1]] <- net.results

#By giving predicted data by NN model, normalizing the data to compare with original data

predicted.data <- unnormalize(net.results$net.result, strength\_min, strength\_max)

# Evaluating the each model base on { RMSE, MAE, MAPE and sMAPE }

# store evaluated data of each matrixes on evaluation matrix

net.evaluation <- standard\_statistical\_indices(output.data, predicted.data)

evaluation[[(length(evaluation)+1)]] <- net.evaluation

# creating comparison table of predicted data and original data ans store the tables on comparison\_table list

output.data.table <- as.data.frame(cbind(output.data, predicted.data))

colnames(output.data.table) <- c("Output.data", "Predicted.data")

print(head(output.data.table))

comparison\_table[[(length(comparison\_table)+1)]] <- output.data.table

}

#store all the data of matrixes that have created using NN on nn\_models\_result model

nn\_models\_result[[(length(nn\_models\_result)+1)]] <- list(NN=nn,

NN\_test\_output= nn\_result,

standard\_statistical\_indices= evaluation,

comparison\_table=comparison\_table)

}

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

2nd subtask

# load installed package libraries

library(dplyr)

library(readxl)

library(neuralnet)

library(Matrix)

#install first the package before calling it via the library command

#setup directory path

script\_path <- dirname(rstudioapi::getSourceEditorContext()$path);

setwd(script\_path)

data <- data.frame(read\_excel("uow\_consumption.xlsx"))

# load columns to variables

data\_20th\_hour <- data.frame(data)[,4]

data\_19th\_hour <- data.frame(data)[,3]

data\_18th\_hour <- data.frame(data)[,2]

# Creating I/O matrixes by time delayed variables

time\_lag\_20\_hour <- bind\_cols(previous7 = lag(data\_20th\_hour, 7),

previous4 = lag(data\_20th\_hour, 4),

previous3 = lag(data\_20th\_hour, 3),

previous2 = lag(data\_20th\_hour, 2),

previous1 = lag(data\_20th\_hour, 1),

output = data\_20th\_hour)

time\_lag\_19\_hour <- bind\_cols(previous7 = lag(data\_19th\_hour, 7),

previous4 = lag(data\_19th\_hour, 4),

previous3 = lag(data\_19th\_hour, 3),

previous2 = lag(data\_19th\_hour, 2),

previous1 = lag(data\_19th\_hour, 1),

output = data\_20th\_hour)

time\_lag\_18\_hour <- bind\_cols(previous7 = lag(data\_18th\_hour, 7),

previous4 = lag(data\_18th\_hour, 4),

previous3 = lag(data\_18th\_hour, 3),

previous2 = lag(data\_18th\_hour, 2),

previous1 = lag(data\_18th\_hour, 1),

output = data\_18th\_hour)

# Remove rows with NA

time\_lag\_20\_hour <- time\_lag\_20\_hour[complete.cases(time\_lag\_20\_hour),]

time\_lag\_19\_hour <- time\_lag\_19\_hour[complete.cases(time\_lag\_19\_hour),]

time\_lag\_18\_hour <- time\_lag\_18\_hour[complete.cases(time\_lag\_18\_hour),]

remove\_outliers <- function(IO\_matrix){

# Create empty list to store outlier row indices

outlier\_rows <- list()

# Loop through columns and extract outlier row indices

boxplot(IO\_matrix, plot = TRUE)

for (col in names(IO\_matrix)) {

# while going through columns in the matrix collecting outlier values

outliers <- boxplot(IO\_matrix[[col]], plot = FALSE)$out

# get rows indicate to the outlier values

row\_indices <- which(IO\_matrix[[col]] %in% outliers)

#if there isnt any outlier it wont go through this process

if (length(row\_indices) > 0) {

outlier\_rows[[col]] <- as.integer(row\_indices)

}

}

#checking rows that have the values on df\_outliers. if it's in there the row will remove

outlier\_rows <- unlist(unique(outlier\_rows))

removed <- IO\_matrix

if(!is.null(outlier\_rows)) {removed <- IO\_matrix[-outlier\_rows, ]}

#adding to a new list that removed outliers

return(as.data.frame(removed))

}

time\_lag\_20\_hour <- remove\_outliers(time\_lag\_20\_hour)

time\_lag\_19\_hour <- remove\_outliers(time\_lag\_19\_hour)

time\_lag\_19\_hour <- remove\_outliers(time\_lag\_20\_hour)

# checking time-delayed I/O configuration from these rows

head (time\_lag\_20\_hour)

head (time\_lag\_19\_hour)

head (time\_lag\_18\_hour)

# define function to normalize the dataset

normalize <- function(value) {

(value - min(value)) / (max(value) - min(value))

}

IO\_matrix\_20th <- as.data.frame(lapply(time\_lag\_20\_hour, normalize))

IO\_matrix\_19th <- as.data.frame(lapply(time\_lag\_19\_hour, normalize))

IO\_matrix\_18th <- as.data.frame(lapply(time\_lag\_18\_hour, normalize))

#Declaring the input and output matrices

io\_matrix\_type3\_20th <- IO\_matrix\_20th[,1:6]

io\_matrix\_type2\_20th <- IO\_matrix\_20th[,2:6]

io\_matrix\_type1\_20th <- IO\_matrix\_20th[,3:6]

io\_matrix\_type3\_19th <- IO\_matrix\_19th[,1:6]

io\_matrix\_type2\_19th <- IO\_matrix\_19th[,2:6]

io\_matrix\_type1\_19th <- IO\_matrix\_19th[,3:6]

io\_matrix\_type3\_18th <- IO\_matrix\_18th[,1:6]

io\_matrix\_type2\_18th <- IO\_matrix\_18th[,2:6]

io\_matrix\_type1\_18th <- IO\_matrix\_18th[,3:6]

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

# creating train Data set

io\_matrix\_type3\_20th\_train <- io\_matrix\_type3\_20th[1:380, ]

io\_matrix\_type2\_20th\_train <- io\_matrix\_type2\_20th[1:380, ]

io\_matrix\_type1\_20th\_train <- io\_matrix\_type1\_20th[1:380, ]

io\_matrix\_type3\_19th\_train <- io\_matrix\_type3\_19th[1:380, ]

io\_matrix\_type2\_19th\_train <- io\_matrix\_type2\_19th[1:380, ]

io\_matrix\_type1\_19th\_train <- io\_matrix\_type1\_19th[1:380, ]

io\_matrix\_type3\_18th\_train <- io\_matrix\_type3\_18th[1:380, ]

io\_matrix\_type2\_18th\_train <- io\_matrix\_type2\_18th[1:380, ]

io\_matrix\_type1\_18th\_train <- io\_matrix\_type1\_18th[1:380, ]

# creating test Data set

io\_matrix\_type3\_20th\_test <- io\_matrix\_type3\_20th[381:nrow(io\_matrix\_type3\_20th), ]

io\_matrix\_type2\_20th\_test <- io\_matrix\_type2\_20th[381:nrow(io\_matrix\_type2\_20th), ]

io\_matrix\_type1\_20th\_test <- io\_matrix\_type1\_20th[381:nrow(io\_matrix\_type1\_20th), ]

io\_matrix\_type3\_19th\_test <- io\_matrix\_type3\_19th[381:nrow(io\_matrix\_type3\_19th), ]

io\_matrix\_type2\_19th\_test <- io\_matrix\_type2\_19th[381:nrow(io\_matrix\_type2\_19th), ]

io\_matrix\_type1\_19th\_test <- io\_matrix\_type1\_19th[381:nrow(io\_matrix\_type1\_19th), ]

io\_matrix\_3\_18th\_test <- io\_matrix\_type3\_18th[381:nrow(io\_matrix\_type3\_18th), ]

io\_matrix\_type2\_18th\_test <- io\_matrix\_type2\_18th[381:nrow(io\_matrix\_type2\_18th), ]

io\_matrix\_type1\_18th\_test <- io\_matrix\_type1\_18th[381:nrow(io\_matrix\_type1\_18th), ]

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

# creating models of 18th hour

model\_18th\_3\_m1 <- neuralnet(output ~. , hidden = 6, data = io\_matrix\_type3\_18th\_train, learningrate = 0.0001, linear.output=FALSE)

model\_18th\_3\_m2 <- neuralnet(output ~. , hidden = c(4,8), data = io\_matrix\_type3\_18th\_train ,act.fct = "logistic", learningrate = 0.01, linear.output=FALSE)

model\_18th\_2\_m1 <- neuralnet(output ~. , hidden = 6, data = io\_matrix\_type2\_18th\_train, learningrate = 0.0001, linear.output=FALSE)

model\_18th\_2\_m2 <- neuralnet(output ~. , hidden = c(4,8), data = io\_matrix\_type2\_18th\_train ,act.fct = "logistic", learningrate = 0.01, linear.output=FALSE)

model\_18th\_1\_m1 <- neuralnet(output ~. , hidden = 6, data = io\_matrix\_type1\_18th\_train, learningrate = 0.0001, linear.output=FALSE)

model\_18th\_1\_m2 <- neuralnet(output ~. , hidden = c(4,8), data = io\_matrix\_type1\_18th\_train ,act.fct = "logistic", learningrate = 0.01, linear.output=FALSE)

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

# predicting the output of 18th hour, base on given training data

pred\_rslt\_18th\_3\_m1 <- compute(model\_18th\_3\_m1, io\_matrix\_type3\_18th\_train[,1:5])

pred\_rslt\_18th\_3\_m2 <- compute(model\_18th\_3\_m2, io\_matrix\_type3\_18th\_train[,1:5])

pred\_rslt\_18th\_2\_m1 <- compute(model\_18th\_2\_m1, io\_matrix\_type2\_18th\_train[,1:4])

pred\_rslt\_18th\_2\_m2 <- compute(model\_18th\_2\_m2, io\_matrix\_type2\_18th\_train[,1:4])

pred\_rslt\_18th\_1\_m1 <- compute(model\_18th\_1\_m1, io\_matrix\_type1\_18th\_train[,1:3])

pred\_rslt\_18th\_1\_m2 <- compute(model\_18th\_1\_m2, io\_matrix\_type1\_18th\_train[,1:3])

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

# creating again 19th hour train matrix

io\_matrix\_3\_19th\_m1\_train <- cbind(io\_matrix\_type3\_19th\_train[,1:5], pred\_rslt\_18th\_3\_m1$net.result, io\_matrix\_type3\_19th\_train["output"])

io\_matrix\_3\_19th\_m2\_train <- cbind(io\_matrix\_type3\_19th\_train[,1:5], pred\_rslt\_18th\_3\_m2$net.result, io\_matrix\_type3\_19th\_train["output"])

io\_matrix\_2\_19th\_m1\_train <- cbind(io\_matrix\_type2\_19th\_train[,1:4], pred\_rslt\_18th\_2\_m1$net.result, io\_matrix\_type2\_19th\_train["output"])

io\_matrix\_2\_19th\_m2\_train <- cbind(io\_matrix\_type2\_19th\_train[,1:4], pred\_rslt\_18th\_2\_m2$net.result, io\_matrix\_type2\_19th\_train["output"])

io\_matrix\_1\_19th\_m1\_train <- cbind(io\_matrix\_type1\_19th\_train[,1:3], pred\_rslt\_18th\_1\_m1$net.result, io\_matrix\_type1\_19th\_train["output"])

io\_matrix\_1\_19th\_m2\_train <- cbind(io\_matrix\_type1\_19th\_train[,1:3], pred\_rslt\_18th\_1\_m2$net.result, io\_matrix\_type1\_19th\_train["output"])

colnames(io\_matrix\_3\_19th\_m1\_train)[6] <- "previous18"

colnames(io\_matrix\_3\_19th\_m2\_train)[6] <- "previous18"

colnames(io\_matrix\_2\_19th\_m1\_train)[5] <- "previous18"

colnames(io\_matrix\_2\_19th\_m2\_train)[5] <- "previous18"

colnames(io\_matrix\_1\_19th\_m1\_train)[4] <- "previous18"

colnames(io\_matrix\_1\_19th\_m2\_train)[4] <- "previous18"

# creating models of 19th hour

model\_19th\_3\_m1 <- neuralnet(output ~. , hidden = 6, data = io\_matrix\_3\_19th\_m1\_train, learningrate = 0.0001, linear.output=FALSE)

model\_19th\_3\_m2 <- neuralnet(output ~. , hidden = c(4,8), data = io\_matrix\_3\_19th\_m2\_train ,act.fct = "logistic", learningrate = 0.01, linear.output=FALSE)

model\_19th\_2\_m1 <- neuralnet(output ~. , hidden = 6, data = io\_matrix\_2\_19th\_m1\_train, learningrate = 0.0001, linear.output=FALSE)

model\_19th\_2\_m2 <- neuralnet(output ~. , hidden = c(4,8), data = io\_matrix\_2\_19th\_m2\_train ,act.fct = "logistic", learningrate = 0.01, linear.output=FALSE)

model\_19th\_1\_m1 <- neuralnet(output ~. , hidden = 6, data = io\_matrix\_1\_19th\_m1\_train, learningrate = 0.0001, linear.output=FALSE)

model\_19th\_1\_m2 <- neuralnet(output ~. , hidden = c(4,8), data = io\_matrix\_1\_19th\_m2\_train ,act.fct = "logistic", learningrate = 0.01, linear.output=FALSE)

# predicting the output of 19th hour, base on given training data

pred\_rslt\_19th\_3\_m1 <- compute(model\_19th\_3\_m1, io\_matrix\_3\_19th\_m1\_train[,1:6])

pred\_rslt\_19th\_3\_m2 <- compute(model\_19th\_3\_m2, io\_matrix\_3\_19th\_m2\_train[,1:6])

pred\_rslt\_19th\_2\_m1 <- compute(model\_19th\_2\_m1, io\_matrix\_2\_19th\_m1\_train[,1:5])

pred\_rslt\_19th\_2\_m2 <- compute(model\_19th\_2\_m2, io\_matrix\_2\_19th\_m2\_train[,1:5])

pred\_rslt\_19th\_1\_m1 <- compute(model\_19th\_1\_m1, io\_matrix\_1\_19th\_m1\_train[,1:4])

pred\_rslt\_19th\_1\_m2 <- compute(model\_19th\_1\_m2, io\_matrix\_1\_19th\_m2\_train[,1:4])

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

# creating again 20th hour train matrix

io\_matrix\_type3\_20th\_m1\_train <- cbind(io\_matrix\_type3\_20th\_train[,1:5], pred\_rslt\_19th\_3\_m1$net.result, io\_matrix\_type3\_20th\_train["output"])

io\_matrix\_type3\_20th\_m2\_train <- cbind(io\_matrix\_type3\_20th\_train[,1:5], pred\_rslt\_19th\_3\_m2$net.result, io\_matrix\_type3\_20th\_train["output"])

io\_matrix\_type2\_20th\_m1\_train <- cbind(io\_matrix\_type2\_20th\_train[,1:4], pred\_rslt\_19th\_2\_m1$net.result, io\_matrix\_type2\_20th\_train["output"])

io\_matrix\_type2\_20th\_m2\_train <- cbind(io\_matrix\_type2\_20th\_train[,1:4], pred\_rslt\_19th\_2\_m2$net.result, io\_matrix\_type2\_20th\_train["output"])

io\_matrix\_type1\_20th\_m1\_train <- cbind(io\_matrix\_type1\_20th\_train[,1:3], pred\_rslt\_19th\_1\_m1$net.result, io\_matrix\_type1\_20th\_train["output"])

io\_matrix\_type1\_20th\_m2\_train <- cbind(io\_matrix\_type1\_20th\_train[,1:3], pred\_rslt\_19th\_1\_m2$net.result, io\_matrix\_type1\_20th\_train["output"])

colnames(io\_matrix\_type3\_20th\_m1\_train)[6] <- "previous19"

colnames(io\_matrix\_type3\_20th\_m2\_train)[6] <- "previous19"

colnames(io\_matrix\_type2\_20th\_m1\_train)[5] <- "previous19"

colnames(io\_matrix\_type2\_20th\_m2\_train)[5] <- "previous19"

colnames(io\_matrix\_type1\_20th\_m1\_train)[4] <- "previous19"

colnames(io\_matrix\_type1\_20th\_m2\_train)[4] <- "previous19"

# creating models of 20th hour

model\_20th\_3\_m1 <- neuralnet(output ~. , hidden = 6, data = io\_matrix\_type3\_20th\_m1\_train, learningrate = 0.0001, linear.output=FALSE)

model\_20th\_3\_m2 <- neuralnet(output ~. , hidden = c(4,8), data = io\_matrix\_type3\_20th\_m2\_train ,act.fct = "logistic", learningrate = 0.01, linear.output=FALSE)

model\_20th\_2\_m1 <- neuralnet(output ~. , hidden = 6, data = io\_matrix\_type2\_20th\_m1\_train, learningrate = 0.0001, linear.output=FALSE)

model\_20th\_2\_m2 <- neuralnet(output ~. , hidden = c(4,8), data = io\_matrix\_type2\_20th\_m2\_train ,act.fct = "logistic", learningrate = 0.01, linear.output=FALSE)

model\_20th\_1\_m1 <- neuralnet(output ~. , hidden = 6, data = io\_matrix\_type1\_20th\_m1\_train, learningrate = 0.0001, linear.output=FALSE)

model\_20th\_1\_m2 <- neuralnet(output ~. , hidden = c(4,8), data = io\_matrix\_type1\_20th\_m2\_train ,act.fct = "logistic", learningrate = 0.01, linear.output=FALSE)

#TESTING-----------------------------------------------------------------------------------------------

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

# predicting the output of 18th hour, base on given testing data

pred\_rslt\_18th\_3\_m1\_test <- compute(model\_18th\_3\_m1, io\_matrix\_3\_18th\_test[,1:5])

pred\_rslt\_18th\_3\_m2\_test <- compute(model\_18th\_3\_m2, io\_matrix\_3\_18th\_test[,1:5])

pred\_rslt\_18th\_2\_m1\_test <- compute(model\_18th\_2\_m1, io\_matrix\_type2\_18th\_test[,1:4])

pred\_rslt\_18th\_2\_m2\_test <- compute(model\_18th\_2\_m2, io\_matrix\_type2\_18th\_test[,1:4])

pred\_rslt\_18th\_1\_m1\_test <- compute(model\_18th\_1\_m1, io\_matrix\_type1\_18th\_test[,1:3])

pred\_rslt\_18th\_1\_m2\_test <- compute(model\_18th\_1\_m2, io\_matrix\_type1\_18th\_test[,1:3])

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

# creating again 19th hour test matrix

io\_matrix\_3\_19th\_m1\_test <- cbind(io\_matrix\_type3\_19th\_test[,1:5], pred\_rslt\_18th\_3\_m1\_test$net.result, io\_matrix\_type3\_19th\_test["output"])

io\_matrix\_3\_19th\_m2\_test <- cbind(io\_matrix\_type3\_19th\_test[,1:5], pred\_rslt\_18th\_3\_m2\_test$net.result, io\_matrix\_type3\_19th\_test["output"])

io\_matrix\_2\_19th\_m1\_test <- cbind(io\_matrix\_type2\_19th\_test[,1:4], pred\_rslt\_18th\_2\_m1\_test$net.result, io\_matrix\_type2\_19th\_test["output"])

io\_matrix\_2\_19th\_m2\_test <- cbind(io\_matrix\_type2\_19th\_test[,1:4], pred\_rslt\_18th\_2\_m2\_test$net.result, io\_matrix\_type2\_19th\_test["output"])

io\_matrix\_1\_19th\_m1\_test <- cbind(io\_matrix\_type1\_19th\_test[,1:3], pred\_rslt\_18th\_1\_m1\_test$net.result, io\_matrix\_type1\_19th\_test["output"])

io\_matrix\_1\_19th\_m2\_test <- cbind(io\_matrix\_type1\_19th\_test[,1:3], pred\_rslt\_18th\_1\_m2\_test$net.result, io\_matrix\_type1\_19th\_test["output"])

colnames(io\_matrix\_3\_19th\_m1\_test)[6] <- "previous18"

colnames(io\_matrix\_3\_19th\_m2\_test)[6] <- "previous18"

colnames(io\_matrix\_2\_19th\_m1\_test)[5] <- "previous18"

colnames(io\_matrix\_2\_19th\_m2\_test)[5] <- "previous18"

colnames(io\_matrix\_1\_19th\_m1\_test)[4] <- "previous18"

colnames(io\_matrix\_1\_19th\_m2\_test)[4] <- "previous18"

# predicting the output of 19th hour, base on given training data

pred\_rslt\_19th\_3\_m1 <- compute(model\_19th\_3\_m1, io\_matrix\_3\_19th\_m1\_test[,1:6])

pred\_rslt\_19th\_3\_m2 <- compute(model\_19th\_3\_m2, io\_matrix\_3\_19th\_m2\_test[,1:6])

pred\_rslt\_19th\_2\_m1 <- compute(model\_19th\_2\_m1, io\_matrix\_2\_19th\_m1\_test[,1:5])

pred\_rslt\_19th\_2\_m2 <- compute(model\_19th\_2\_m2, io\_matrix\_2\_19th\_m2\_test[,1:5])

pred\_rslt\_19th\_1\_m1 <- compute(model\_19th\_1\_m1, io\_matrix\_1\_19th\_m1\_test[,1:4])

pred\_rslt\_19th\_1\_m2 <- compute(model\_19th\_1\_m2, io\_matrix\_1\_19th\_m2\_test[,1:4])

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

# creating again 20th hour test matrix

io\_matrix\_type3\_20th\_m1\_test <- cbind(io\_matrix\_type3\_20th\_test[,1:5], pred\_rslt\_19th\_3\_m1$net.result, io\_matrix\_type3\_20th\_test["output"])

io\_matrix\_type3\_20th\_m2\_test <- cbind(io\_matrix\_type3\_20th\_test[,1:5], pred\_rslt\_19th\_3\_m2$net.result, io\_matrix\_type3\_20th\_test["output"])

io\_matrix\_type2\_20th\_m1\_test <- cbind(io\_matrix\_type2\_20th\_test[,1:4], pred\_rslt\_19th\_2\_m1$net.result, io\_matrix\_type2\_20th\_test["output"])

io\_matrix\_type2\_20th\_m2\_test <- cbind(io\_matrix\_type2\_20th\_test[,1:4], pred\_rslt\_19th\_2\_m2$net.result, io\_matrix\_type2\_20th\_test["output"])

io\_matrix\_type1\_20th\_m1\_test <- cbind(io\_matrix\_type1\_20th\_test[,1:3], pred\_rslt\_19th\_1\_m1$net.result, io\_matrix\_type1\_20th\_test["output"])

io\_matrix\_type1\_20th\_m2\_test <- cbind(io\_matrix\_type1\_20th\_test[,1:3], pred\_rslt\_19th\_1\_m2$net.result, io\_matrix\_type1\_20th\_test["output"])

colnames(io\_matrix\_type3\_20th\_m1\_test)[6] <- "previous19"

colnames(io\_matrix\_type3\_20th\_m2\_test)[6] <- "previous19"

colnames(io\_matrix\_type2\_20th\_m1\_test)[5] <- "previous19"

colnames(io\_matrix\_type2\_20th\_m2\_test)[5] <- "previous19"

colnames(io\_matrix\_type1\_20th\_m1\_test)[4] <- "previous19"

colnames(io\_matrix\_type1\_20th\_m2\_test)[4] <- "previous19"

# predicting the output of 20th hour, base on given training data

pred\_rslt\_20th\_3\_m1\_test <- compute(model\_20th\_3\_m1, io\_matrix\_type3\_20th\_test[,1:6])

pred\_rslt\_20th\_3\_m2\_test <- compute(model\_20th\_3\_m2, io\_matrix\_type3\_20th\_test[,1:6])

pred\_rslt\_20th\_2\_m1\_test <- compute(model\_20th\_2\_m1, io\_matrix\_type2\_20th\_test[,1:5])

pred\_rslt\_20th\_2\_m2\_test <- compute(model\_20th\_2\_m2, io\_matrix\_type2\_20th\_test[,1:5])

pred\_rslt\_20th\_1\_m1\_test <- compute(model\_20th\_1\_m1, io\_matrix\_type1\_20th\_test[,1:4])

pred\_rslt\_20th\_1\_m2\_test <- compute(model\_20th\_1\_m2, io\_matrix\_type1\_20th\_test[,1:4])

# and find its maximum & minimum value

minimum\_of\_20th <- min(data\_20th\_hour)

maximum\_of\_20th <- max(data\_20th\_hour)

minimum\_of\_19th <- min(data\_19th\_hour)

maximum\_of\_19th <- max(data\_19th\_hour)

minimum\_of\_18th <- min(data\_18th\_hour)

maximum\_of\_18th <- max(data\_18th\_hour)

#Create the reverse of normalised function – de-normalized

denormalization <- function(value, min, max) {

return( (max - min)\*value + min )

}

predict\_3\_m1 <- denormalization(pred\_rslt\_20th\_3\_m1\_test$net.result, minimum\_of\_20th, maximum\_of\_20th)

predict\_3\_m2 <- denormalization(pred\_rslt\_20th\_3\_m2\_test$net.result, minimum\_of\_20th, maximum\_of\_20th)

predict\_2\_m1 <- denormalization(pred\_rslt\_20th\_2\_m1\_test$net.result, minimum\_of\_20th, maximum\_of\_20th)

predict\_2\_m2 <- denormalization(pred\_rslt\_20th\_2\_m2\_test$net.result, minimum\_of\_20th, maximum\_of\_20th)

predict\_1\_m1 <- denormalization(pred\_rslt\_20th\_1\_m1\_test$net.result, minimum\_of\_20th, maximum\_of\_20th)

predict\_1\_m2 <- denormalization(pred\_rslt\_20th\_1\_m2\_test$net.result, minimum\_of\_20th, maximum\_of\_20th)

original20th\_3 <- denormalization(io\_matrix\_type3\_20th\_test["output"], minimum\_of\_20th, maximum\_of\_20th)

original20th\_2 <- denormalization(io\_matrix\_type2\_20th\_test["output"], minimum\_of\_20th, maximum\_of\_20th)

original20th\_1 <- denormalization(io\_matrix\_type1\_20th\_test["output"], minimum\_of\_20th, maximum\_of\_20th)

table\_3\_m1 <- cbind(predict\_3\_m1, original20th\_3)

table\_3\_m2 <- cbind(predict\_3\_m2, original20th\_3)

table\_2\_m1 <- cbind(predict\_2\_m1, original20th\_2)

table\_2\_m2 <- cbind(predict\_2\_m2, original20th\_2)

table\_1\_m1 <- cbind(predict\_1\_m1, original20th\_1)

table\_1\_m2 <- cbind(predict\_1\_m2, original20th\_1)

colnames(table\_3\_m1)[1] <- "predict"

colnames(table\_3\_m2)[1] <- "predict"

colnames(table\_2\_m1)[1] <- "predict"

colnames(table\_2\_m2)[1] <- "predict"

colnames(table\_1\_m1)[1] <- "predict"

colnames(table\_1\_m2)[1] <- "predict"

rownames(table\_3\_m1) <- 1:nrow(table\_3\_m1)

rownames(table\_3\_m2) <- 1:nrow(table\_3\_m2)

rownames(table\_2\_m1) <- 1:nrow(table\_2\_m1)

rownames(table\_2\_m2) <- 1:nrow(table\_2\_m2)

rownames(table\_1\_m1) <- 1:nrow(table\_1\_m1)

rownames(table\_1\_m2) <- 1:nrow(table\_1\_m2)

# Evaluating the model

library(Metrics)

standard\_statistical\_indices <- function(table){

original <- unlist(table[,2])

predict <- unlist(table[,1])

print(head(table))

rmse <- rmse(original, predict)

message("rmse :",rmse)

mae <- mae(original, predict)

message("mae :",mae)

mape <- mape(original, predict)

message("mape :",mape)

smape<- smape(original, predict)

message("smape :",smape)

}

standard\_statistical\_indices(table\_3\_m1)

standard\_statistical\_indices(table\_3\_m2)

standard\_statistical\_indices(table\_2\_m1)

standard\_statistical\_indices(table\_2\_m2)

standard\_statistical\_indices(table\_1\_m1)

standard\_statistical\_indices(table\_1\_m2)

# **References**

Crone, Sven & Kourentzes, Nikolaos. (2007). Input variable selection for time series prediction with neural networks- an evaluation of visual, autocorrelation and spectral analysis for varying seasonality.

DigitalOcean. (n.d.). Outlier Analysis in R. Retrieved March 21, 2023, from

Nistrup P, Principal Component Analysis (PCA) 101 using R. Towards Data Science. Retrieved from https://towardsdatascience.com/principal-component-analysis-pca-101-using-r-361f4c53a9ff.