Informatics Institute of Technology

In collaboration with

UNIVERSITY OF WESTMINSTER

Machine Learning and Data Mining

5DATA001C.2

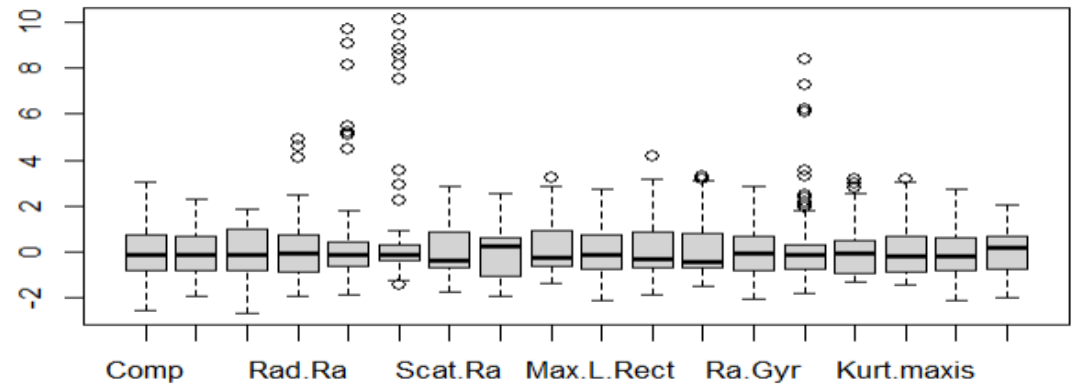
FINAL COURSEWORK

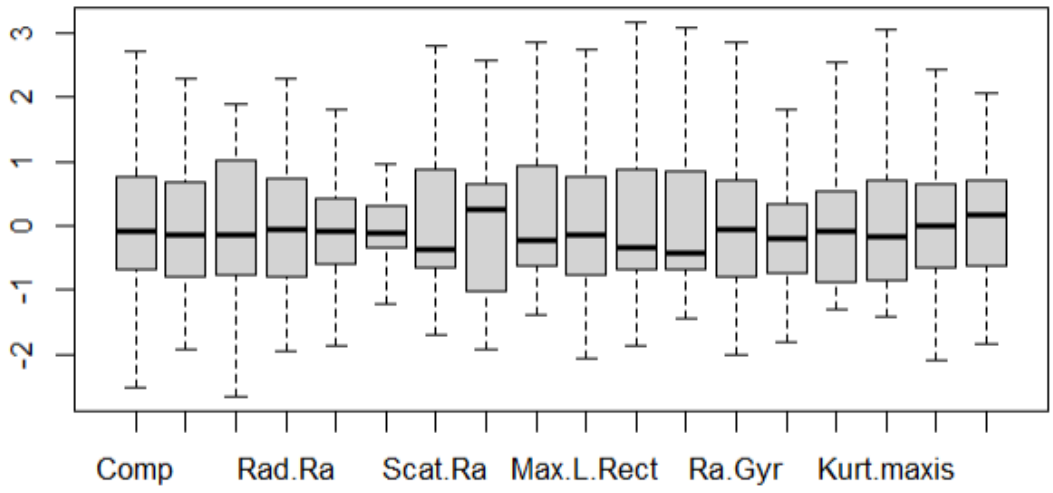
Module Leader's Name- Mr. Nipuna Senanayaka

Name- Didula
Thaveesha
UoW No.- W1870577
Student ID- 20210174

1st Objective (Partitioning Clustering)

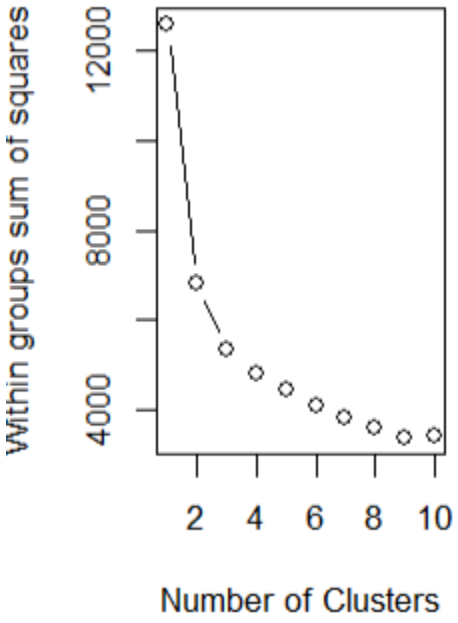
a. Prior to being utilized as factors, the value class names in the dataset must first be cleaned. The final product, once the class names were cleaned up, is seen in the image below. Next, scaling and outlier removal are carried out. Outliers can cause problems like faulty data or incorrect projections of results. Outliers increase the variability of the data, which lowers statistical power. We might thus improve the importance and accuracy of the data by removing the outliers. But occasionally we won't even have to get rid of the outliers. In this situation, the ideal strategy is outlier eradication. To prevent extreme values from skewing the clustering findings, outlier detection and removal are also crucial. Outliers can affect how the mean and standard deviation are calculated and can cause clusters to emerge that are not typical of the bulk of the data. The z-score approach, the interquartile range (IQR) method, and the manhaton distance method are a few techniques for finding outliers. Outliers can either be eliminated from the dataset after they have been located or their values can be imputed using a variety of methods.





b. The number of clusters may be calculated using four basic techniques: the silhouetted approach, NBclust, elbow, and gap statistics. Determine the number of clusters using one of four basic techniques: NBclust, elbow, gap statistics, or silhouette approach..

- The elbow method is widely used to find the appropriate number of clusters for a k-means clustering algorithm. It is based on the observation that the within-cluster sum of squares (WCSS) decreases as the number of clusters rises. But when a certain number of clusters are added, the WCSS gain becomes less significant.

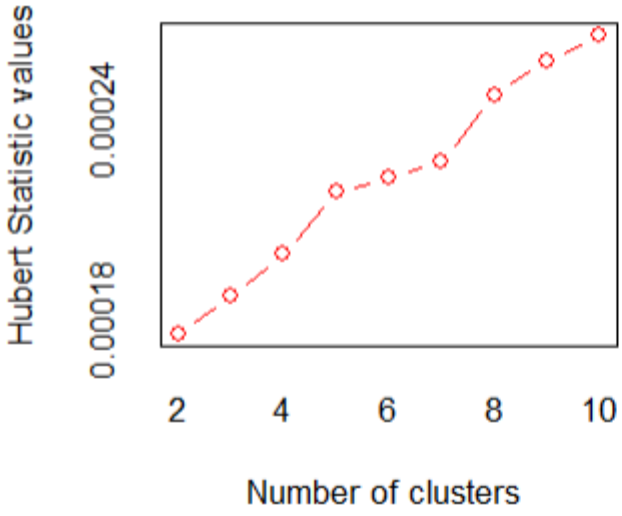


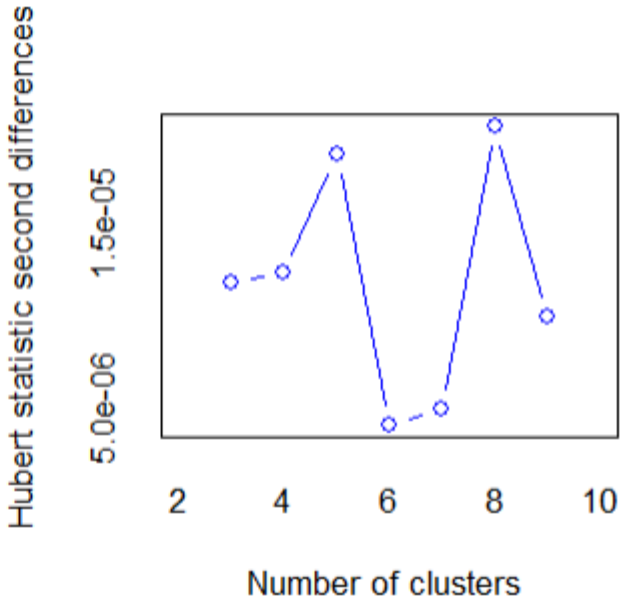
- The NbClust method is another popular technique for determining the optimum number of clusters for k-means clustering. Several clustering validity indices are used to identify the number of clusters that best fit the data.

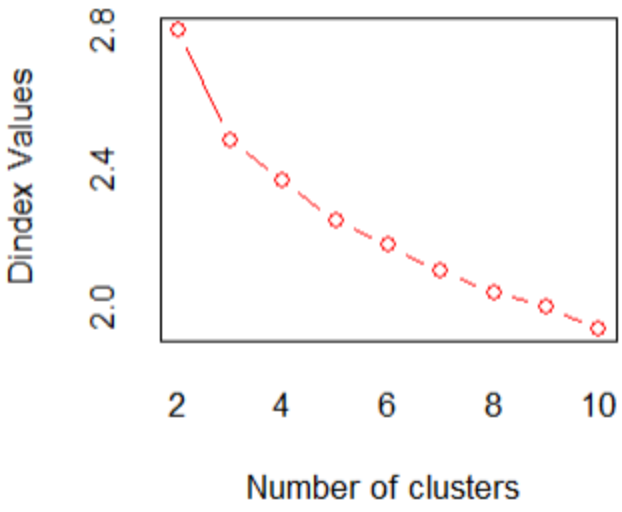
```
*******************************************************************
* Among all indices:
* 10 proposed 2 as the best number of clusters
* 11 proposed 3 as the best number of clusters
* 1 proposed 4 as the best number of clusters
* 2 proposed 10 as the best number of clusters

                 ***** Conclusion *****

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

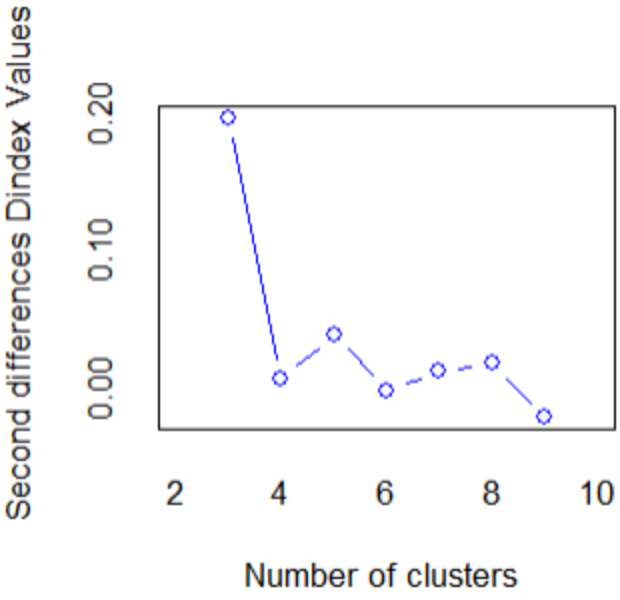
```r
# NBclust method

set.seed(123)
nb <- NbClust(cleaned_vehicle, distance = "euclidean", min.nc = 2, max.nc = 10, r
k <- nb$Best.nc

# Elbow method
wss <- (nrow(cleaned_vehicle)-1)*sum(apply(cleaned_vehicle,2,var))
for (i in 1:10) wss[i] <- sum(kmeans(cleaned_vehicle, centers=i)$withinss)
plot(1:10, wss, type="b", xlab="Number of Clusters", ylab="Within groups sum of s
#k_optimal <- 3

# Gap statistic method
library(cluster)
set.seed(123)
fviz_nbclust(cleaned_vehicle, kmeans, nstart = 25,  method = "gap_stat", nboot =
  labs(subtitle = "Gap statistic method")


# Silhouette method
fviz_nbclust(cleaned_vehicle, FUN = hcut, method = "silhouette") +
  labs(subtitle = "Silhouette method")
optimal_clusters <- 3 # Replace this with the optimal number you found
kmeans_result <- kmeans(cleaned_vehicle, centers = optimal_clusters)

print(kmeans_result)
```

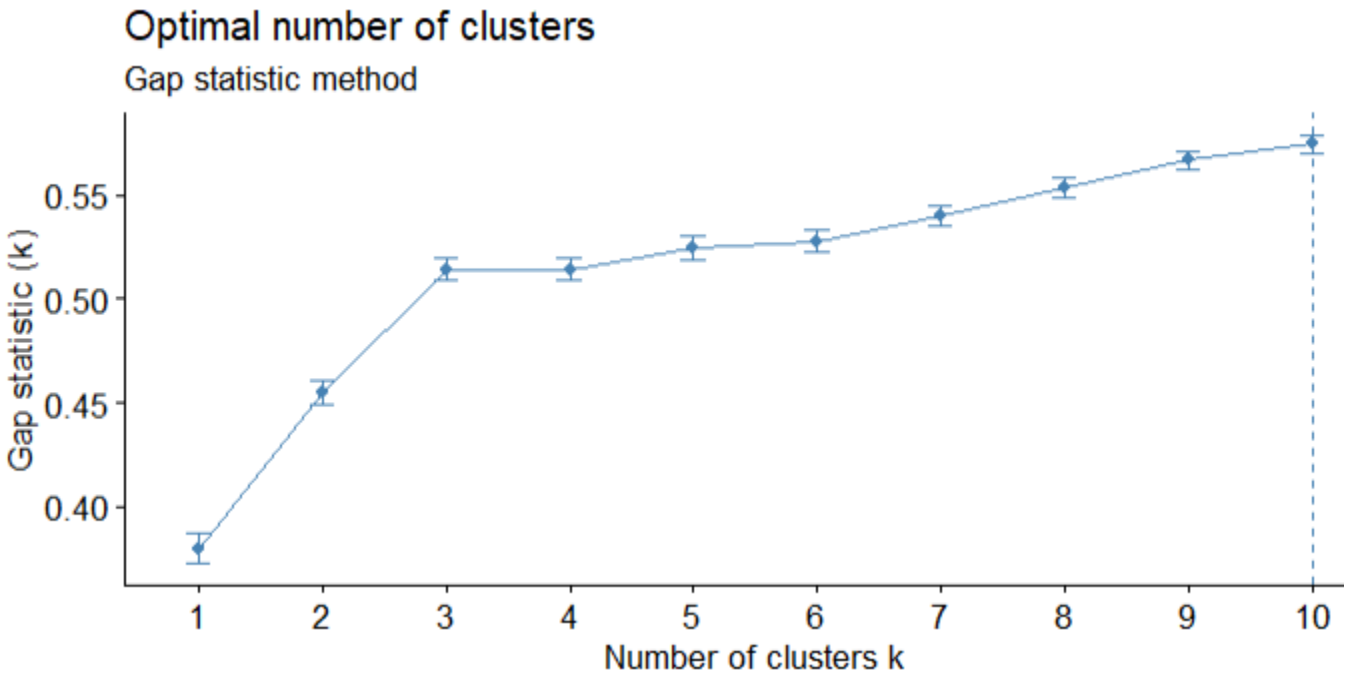
```
>
> # Print the selected number of clusters for each method
> cat("Number of clusters selected via NBclust method:", k_nbclust, "\n")
Number of clusters selected via NBclust method: 8 6.9904 2 588.9425 7 202.527 13 15.7918
27.657 3 1.854533e+35 3 290423.4 3 1036.247 4 189.4463 8 -0.6056 7 0.1714 2 1.143 2 0.389
1.7067 2 -246.7959 2 -5.1546 3 0.3897 3 2046.197 2 0.5772 2 1.4026 2 0.5019 10 0.084 0 0
2308 0 0 15 0.3559
> cat("Number of clusters selected via Elbow method:", k_elbow, "\n")
Number of clusters selected via Elbow method: 3
> #cat("Number of clusters selected via Gap statistic method:", k_gap, "\n")
> cat("Number of clusters selected via Silhouette method:", k_silhouette, "\n")
Number of clusters selected via Silhouette method: 9
>
```

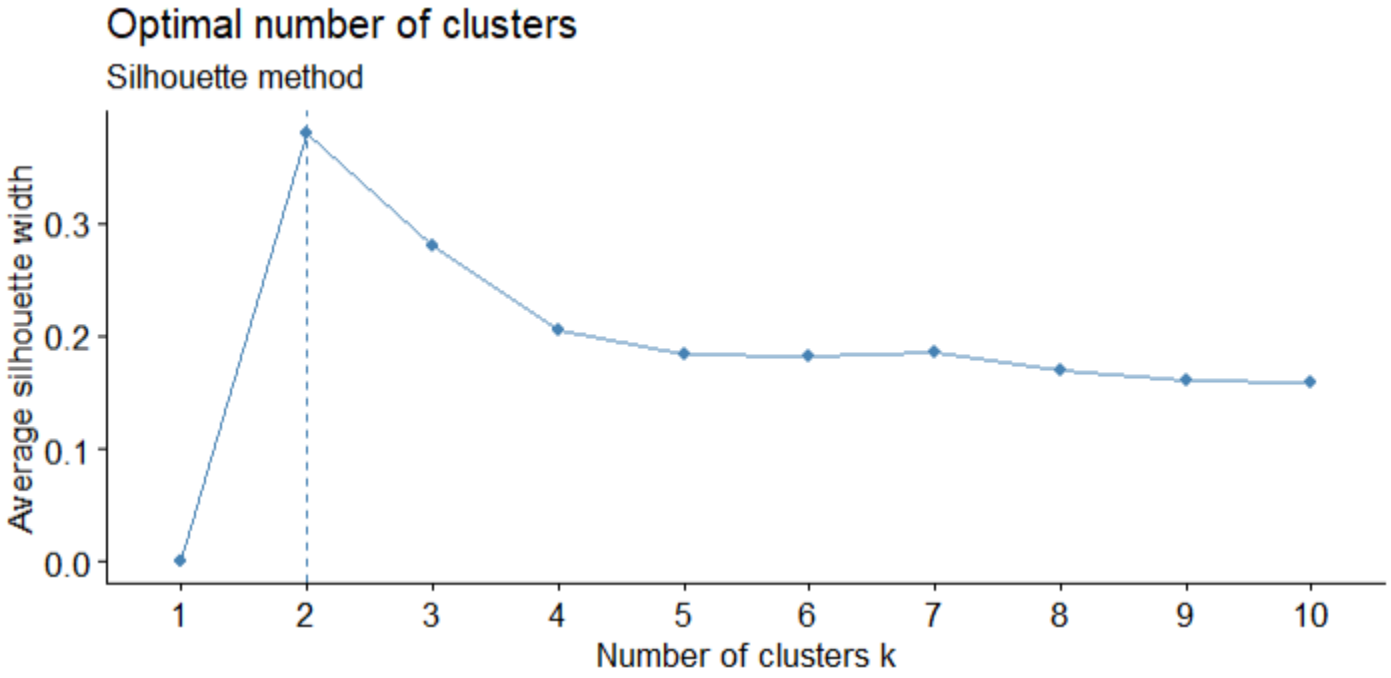

Hubert Statistic values
0.00018
0.00024
2
4
6
8
10
Number of clusters


Hubert statistic second differences
5.0e-06
1.5e-05
2
4
6
8
10
Number of clusters


Dindex Values
2.0
2.4
2.8
2
4
6
8
10
Number of clusters


Second differences Dindex Values
0.00
0.10
0.20
2
4
6
8
10
Number of clusters





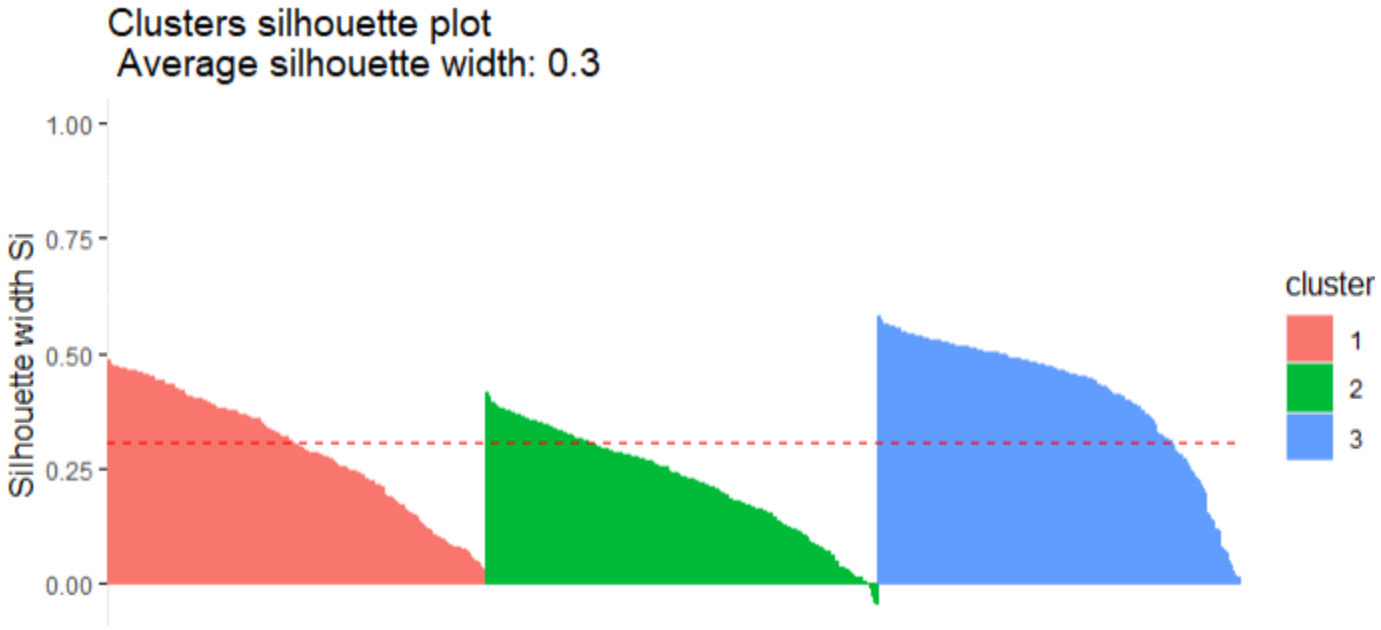
All of these factors were examined, and we were able to identify three clusters.

C. The between_cluster_sums_of_squares (BSS) and the within_cluster_sums_of_squares (WSS) are significant internal assessment metrics for clustering methods. The BSS measures the level of variation across clusters, whereas the WSS measures the level of variance within clusters. Either of these metrics may be used to determine the total_sum_of_squares (TSS), which denotes the overall level of variation in the data.

```
# Set seed for reproducibility
set.seed(123)

# Perform kmeans clustering with k=3
kmeans_result <- kmeans(cleaned_vehicle, centers = 3)

# Print kmeans output
kmeans_result

# Print centers of each cluster
print(kmeans_result$centers)

# Print cluster assignment for each observation
print(kmeans_result$cluster)

# Calculate and print BSS and WSS
BSS <- sum(kmeans_result$size * apply(kmeans_result$centers, 1,
WSS <- sum(kmeans_result$withinss)
TSS <- sum(kmeans_result$tot.withinss) + sum(kmeans_result$betwe
BSS_ratio <- BSS / TSS
cat("BSS:", BSS, "\n")
cat("WSS:", WSS, "\n")
cat("BSS/TSS ratio:", BSS_ratio,"\n")
```

```
> TSS <- sum(kmeans_result$tot.withinss) +
> BSS_ratio <- BSS / TSS
> cat("BSS:", BSS, "\n")
BSS: 7263.373
> cat("WSS:", WSS, "\n")
WSS: 5327.275
> cat("BSS/TSS ratio:", BSS_ratio,"\n")
BSS/TSS ratio: 0.5776449
>
>
```

d. The silhouette width score is an external evaluation tool that rates the quality of clustering results. It evaluates how similar each point in a cluster is to the points in its own cluster when compared to points in other clusters. Depending on the cluster, a point's silhouette width score can range from -1 to 1, with a value of 1 indicating that the point is unique from points in other clusters and quite similar to points in its own cluster. The opposite is indicated with a score of -1. A point with a score that is almost 0 is bordered by two clusters and might potentially belong to either one.



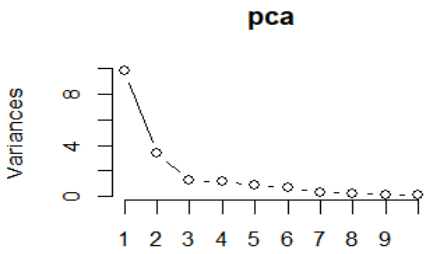
The silhouette plot shows how close together every point in a cluster is with every other point in every other cluster. Each bar's color corresponds to the cluster's average silhouette width, while its height denotes the number of points in the cluster. The average silhouette width score, which gauges how well the clustering is done. Higher numbers suggest better clustering, and the range is from -1 to 1. A score of 0 implies overlapping clusters, whereas negative values show that the incorrect cluster was given the points. We can assess the caliber of the resulting clusters using the silhouette plot and the average silhouette width score. The clustering is of excellent quality if the silhouette plot reveals distinct, clearly delineated clusters with high average silhouette width scores. On the other hand, we may infer that the clustering is of low quality if the silhouette plot reveals overlapping clusters with low average silhouette width scores. It's also important to note that the silhouette plot allows for a revision of the ideal number of clusters. The ideal number of clusters may be determined as the one that maximizes the average silhouette width score. A successful clustering should have high average silhouette width scores for all clusters.

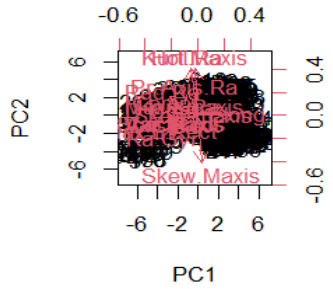
```
> #------------Part D ----------------------------------
> fviz_silhouette(silhouette(kmeans_result$cluster, dist(cleaned_vehicle)))
  cluster size ave.sil.width
1       1  268          0.29
2       2  278          0.22
3       3  256          0.42
```

e. Picking the bare minimum of PCs that account for a certain percentage of the total variance in the data is a common method for deciding how many PCs to use. We picked 92% as the criterion for the proportion of explained variance in this instance because we want to keep as much information as we can while still reducing the dimensionality of the data.

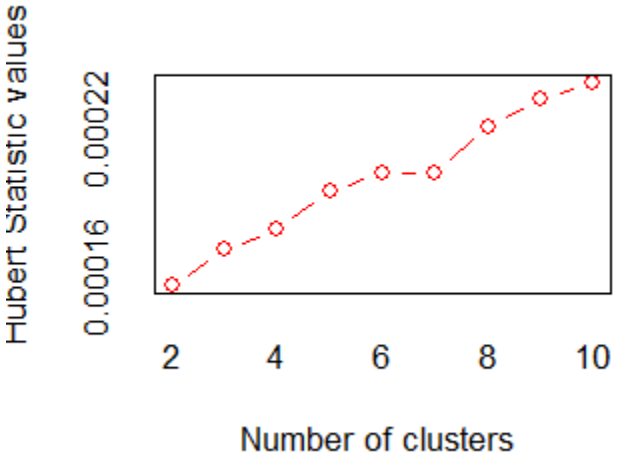
```
#------------- Part E ---------------------------

# Perform PCA on the scaled data
pca <- prcomp(cleaned_vehicle)

# The eigenvalues and eigenvectors should be shown.
summary(pca)

# Calculate the overall score for each principal component.
cumulative <- cumsum(pca$sdev^2 / sum(pca$sdev^2))

# Make a new dataset with the characteristics of the primary co
X_pca <- predict(pca, cleaned_vehicle)

# Select the main factors that result in a cumulative score of
num_pcs <- sum(cumulative <= 0.92)
X_pca <- X_pca[, 1:num_pcs]

# Display the transformed dataset
head(X_pca)
```

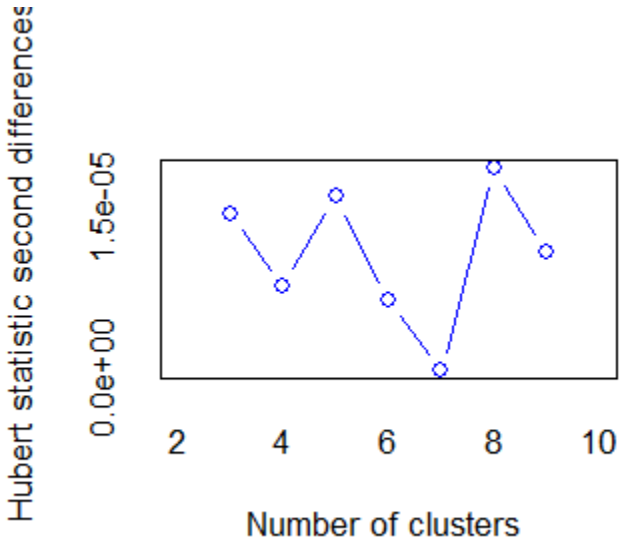
```
> # Display the transformed dataset
> head(X_pca)
            PC1        PC2         PC3         PC4
[1,] -0.3087748  0.2115755 -0.30351748  0.25672343
[2,]  1.6029544  0.2662707 -0.06218716 -0.87952757
[3,] -3.7390737 -0.2126762 -1.05910499 -0.84065687
[4,]  1.7931596  2.7120368 -0.50485687  0.07142780
[5,] -6.4964800 -3.9792683  0.48095252  0.03755161
[6,]  0.8303930  2.0654186 -2.13306216 -0.42930042
>
```

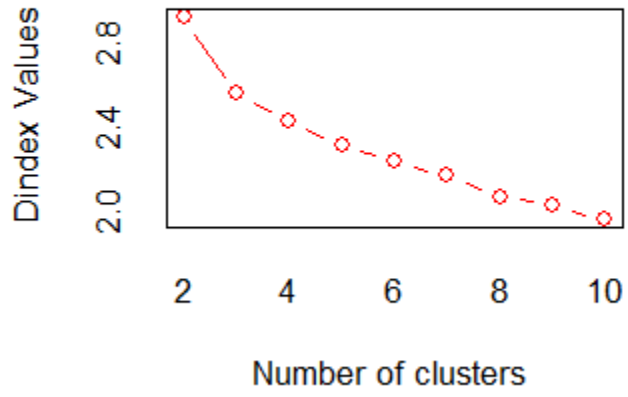


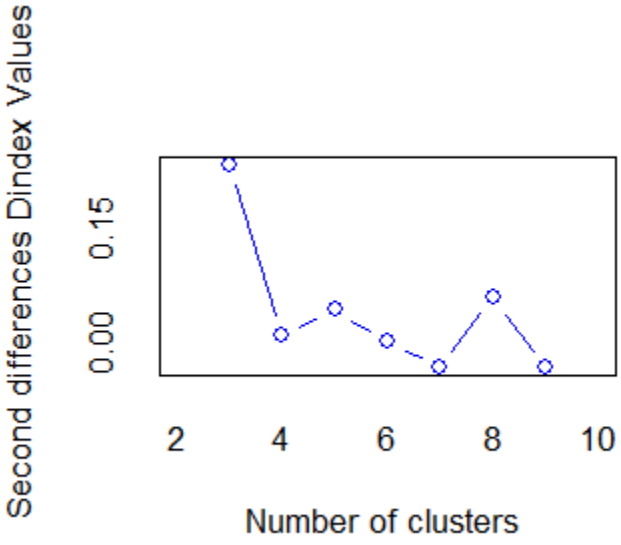


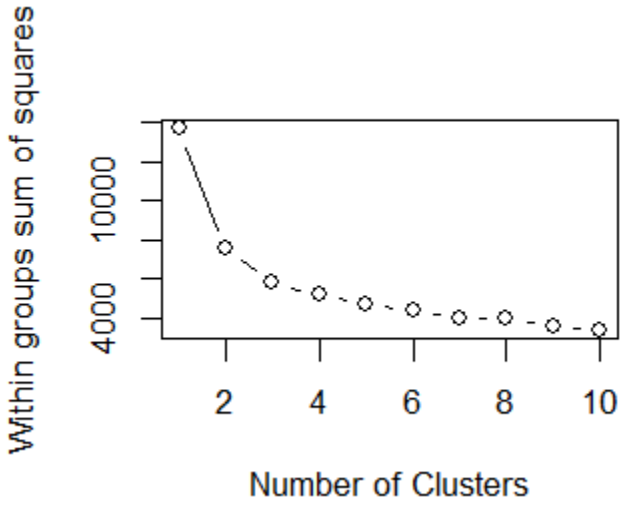
f. The gap statistics approach, which is often used in unsupervised learning tasks like clustering and dimensionality reduction, may be utilized with a variety of clustering approaches, including k-means, hierarchical clustering, and spectral clustering. The advantage of the gap statistics method is that it provides a numerical assessment of the ideal cluster density, which may be used to guide the choice of hyperparameters for the clustering algorithm. You can contrast the outcomes with those attained from your old dataset after using these strategies to your new dataset. The structure of your data may have changed if the ideal value of k changes, and you may need to investigate the causes of this shift. You may also use visualizations to go deeper into the clusters and their properties.

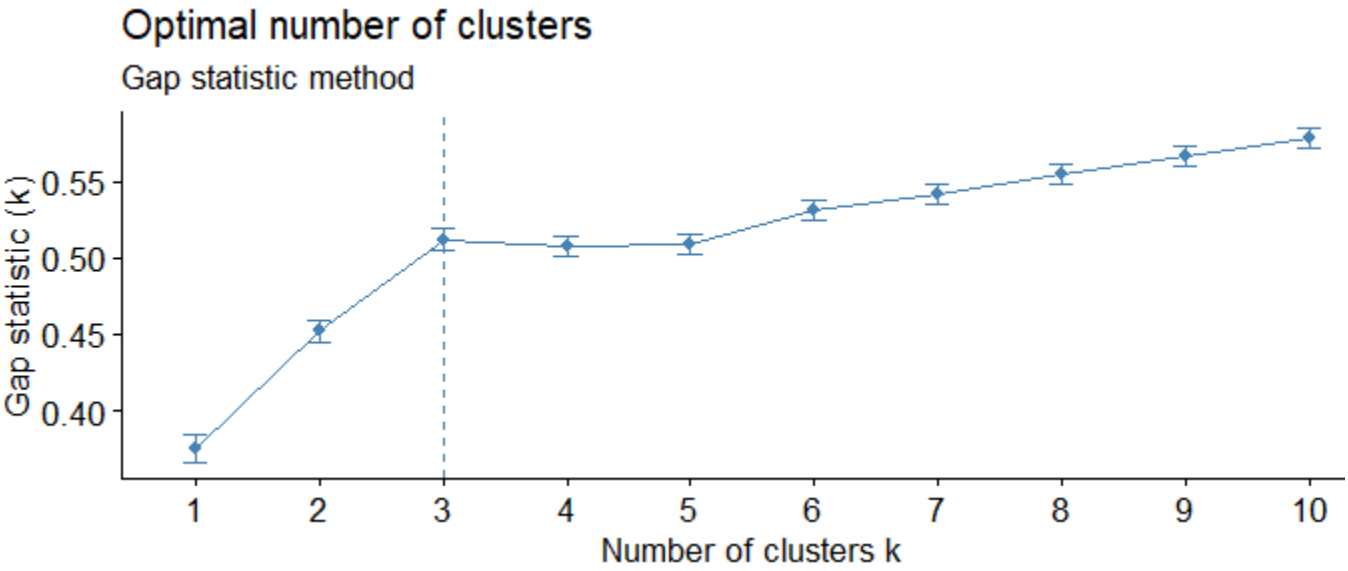













```
* Among all indices:
* 10 proposed 2 as the best number of clusters
* 7 proposed 3 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 3 proposed 8 as the best number of clusters
* 3 proposed 10 as the best number of clusters

                 ***** Conclusion *****

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

g.

$$d(i,j)=\sqrt{\sum_{k=1}^{p}(x_{ki}-x_{kj})^2}$$

Euclidean Distance Formula's

$$d(i,j)=|x_{i1}-x_{j1}|+|x_{i2}-x_{j2}|+...+|x_{ip}-x_{jp}|$$

Manhattan Distance Formula's

```
Within cluster sum of squares by cluster:
[1] 5164.917 2362.155
 (between_SS / total_SS =  45.0 %)
```

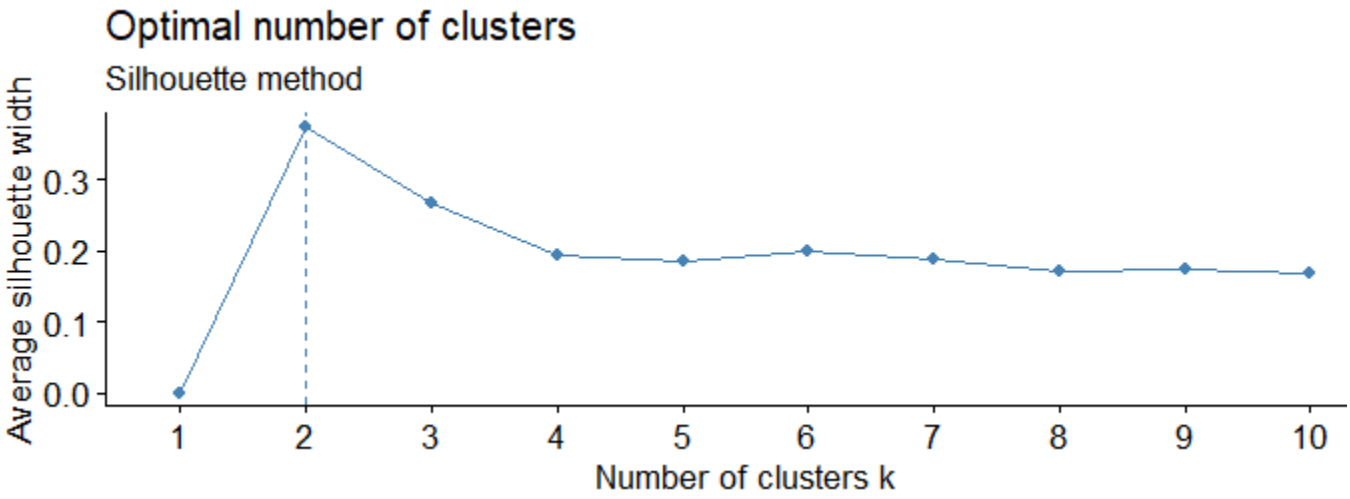
```
> WSS <- sum(kmeans_result$withinss)
> TSS <- sum(kmeans_result$tot.withinss) + sum(kmeans_result$betweenss)
> BSS_ratio <- BSS / TSS
> cat("BSS:", BSS, "\n")
BSS: NA
> cat("WSS:", WSS, "\n")
WSS: 7527.072
> cat("BSS/TSS ratio:", BSS_ratio,"\n")
BSS/TSS ratio: NA
> fviz_cluster(kmeans_result,data = newData)
```

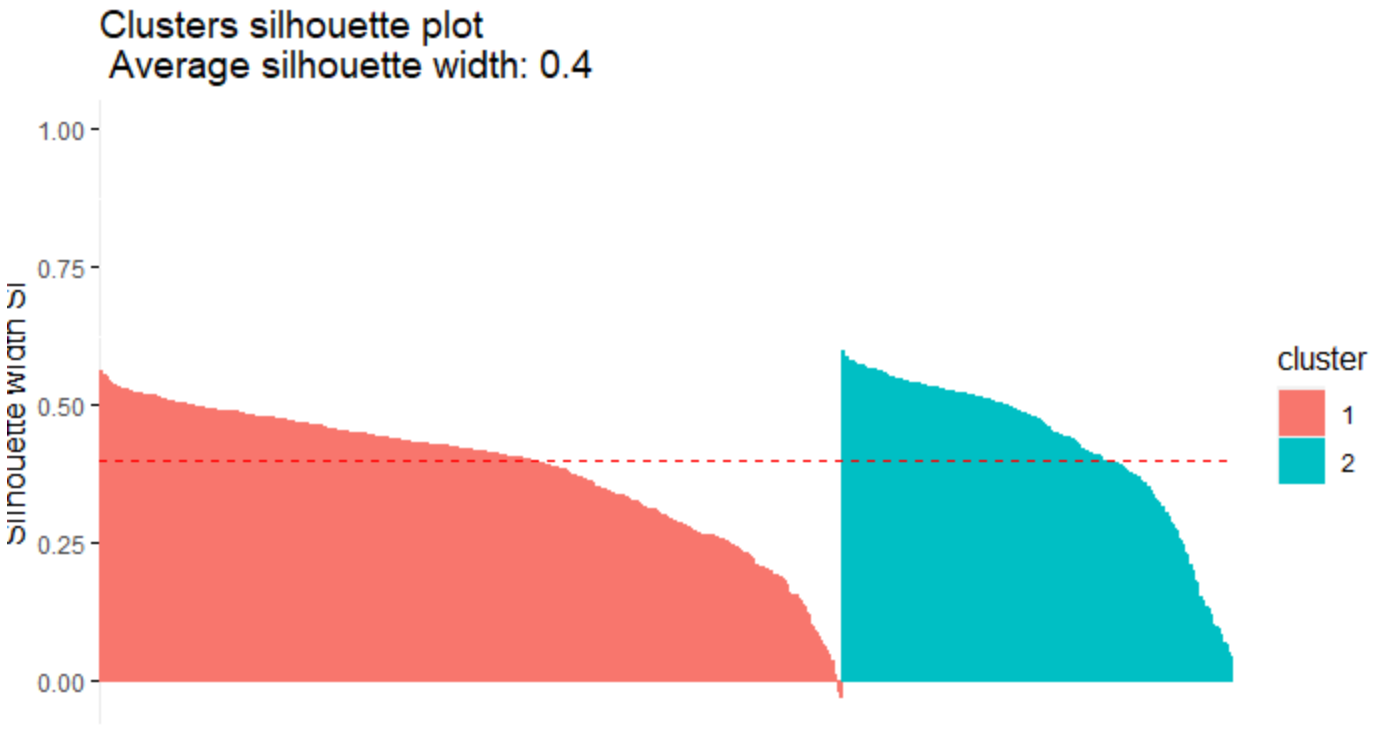
h.

The silhouette plot is a useful tool for assessing the quality of clustering since it lets you examine the distances between clusters as well as any overlapping or incorrectly separated groups. The best clustering result corresponds to the average silhouette width score (average silhouette coefficient across all points), therefore it may also be used to estimate the appropriate number of clusters.

Better clustering quality is frequently suggested by a higher average silhouette width score, which shows that the points are evenly divided into their different clusters. The silhouette coefficient, however, must always be used in conjunction with other evaluation metrics and subject-matter knowledge because its interpretation depends on the accuracy of the data and clustering method utilized.

```
#-------------------Part H-----------------------------

fviz_silhouette(silhouette(kmeans_result$cluster, dist(newData)))
cluster size ave.sil.width
      1  526          0.38
      2  276          0.43
```




Clusters silhouette plot
Average silhouette width: 0.4
Silhouette width si
1.00
0.75
0.50
0.25
0.00
cluster
1
2


Cluster plot
Dim2 (16.7%)
Dim1 (16.7%)
2
1
0
-1
-2
-3
clust

i. An internal assessment metric that is frequently used to ascertain the ideal number of clusters in a k-means clustering method is the Calinski-Harabasz Index. It calculates the difference between the variances within and between clusters. The clusters are more distinct and isolated from one another when the Calinski-Harabasz score is greater.

```
#------------------Part I---------------------------
# Calculate Calinski-Harabasz index
ch_index <- calinhara(newData, kmeans_result$cluster)
# Print Calinski-Harabasz index
print(ch_index)
```

```
> #------------------Part I---------------------------
> # Calculate Calinski-Harabasz index
> ch_index <- calinhara(newData, kmeans_result$cluster)
> # Print Calinski-Harabasz index
> print(ch_index)
[1] 653.5591
>
```

2nd Objective (MLP) Objectives/Deliverables (Multi-layer Neural Network)

a. Several input variables are used to forecast the load demand in the future and make up the input vector in MLP models for electrical load forecasting. The choice and definition of the input variables, which have a direct impact on how well results are anticipated, is a crucial phase in the model-development process.

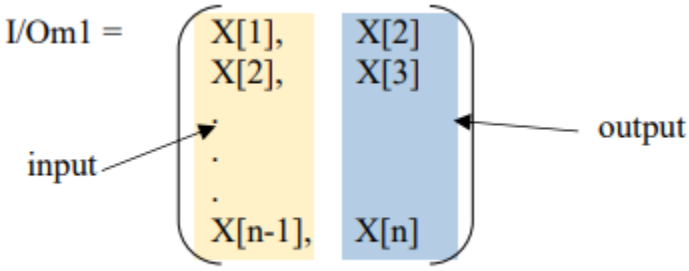
There are three crucial layers to take into account while looking for an MLP neural network. Approach utilizing autoregressive (AR) models, specifically The historical load and demand statistics are used as input variables in these models. If we are forecasting the load demand for the following hour, for example, we may use the load demand numbers from the preceding 12 or 24 hours as input variables.

This approach uses the Fourier transform to extract seasonal patterns from load demand data, which are then utilized as input variables. Multilayer Perceptron (MLP) models are a popular option for electrical load forecasting because of their ability to handle complex nonlinear relationships between input variables and the desired output. For establishing the input vector for MLP models, which is crucial to determining the model's effectiveness, numerous techniques and methodologies have been given in the field of electrical load forecasting. The autoregressive (AR) approach is one that is used in courses. The previous values of the target variable (load) make up the input vector for the AR technique. For instance, if the AR order was p, the input vector would be [load(t-1), load(t-2),..., load(t-p)]. The idea underlying the AR approach is that one may predict a system's future behavior by looking at its past behavior.

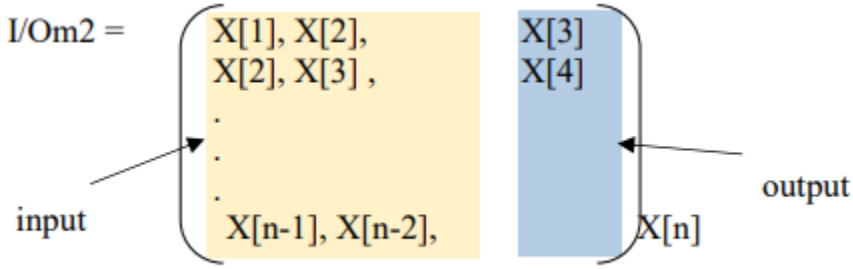
When determining the input vector for MLP models in energy load forecasting, it is necessary to review pertinent literature and assess the applicability of various schemes for the specific forecasting situation at hand. . In summary, choosing the appropriate input variables for MLP models is essential for creating accurate predictions in energy load forecasting. A variety of methods, including the AR approach, exogenous variables, wavelet transform, PCA, and hybrid approaches, may be used to create the input vector. Researchers and practitioners should carefully consider these techniques and review relevant literature before deciding on the appropriate solution for their specific forecasting problem.

b. Normalization is a crucial preprocessing step before training neural networks, especially for Multilayer Perceptron (MLP) architectures. Normalization's primary goal is to rescale input characteristics to equalize their ranges and distributions. Expanding or decreasing gradient issues may occur while neural networks are being trained. Normalization helps to solve these issues. Without normalization, some input attributes could be more significant than others, resulting in unstable model training and subpar performance. Normalization can also speed up the model's convergence and prevent overfitting. Thus, normalizing the input data to a standard scale (e.g., zero mean and unit variance) can improve the stability and efficiency of MLP networks during training, leading to higher performance on unknown data.

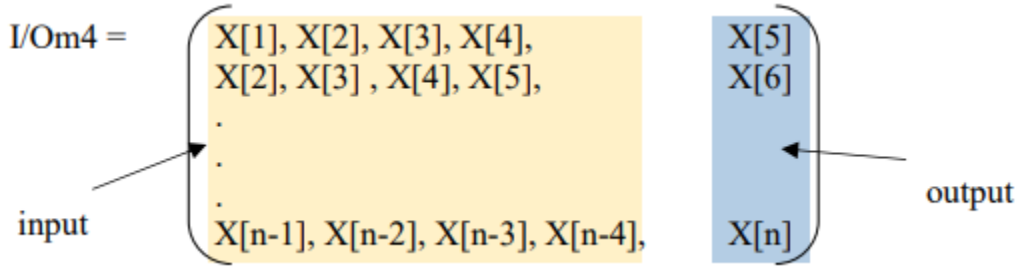
Output matrix where order of approach = 1



Output matrix where order of approach = 2,



Output matrix where order of approach = 3



An input-output (IO) matrix for a dataset of electricity usage displays the correlations between different nighttime hours and the corresponding power consumption levels. This would result in a 3x3 matrix where each cell would represent the relationship between consumption during hours 6, 7, and 8 of the evening.

A mathematical tool used to show the interdependence between different economic variables or sectors is an input-output matrix, or IO matrix. In the context of a data set of electricity use, an IO matrix would be a matrix that shows the inputs and outputs of energy use between hours six through eight of the evenings.

```r
# Create input/output matrices with different lagged time steps
time_lagged_data_1 <- data.frame(
  t_1 = lag(data$time8, 1),
  output = data$time8)[-c(1, 1), ]
time_lagged_data_2 <- data.frame(t_2 = lag(data$time8, 2),
                                 t_1 = lag(data$time8, 1),
                                 output = data$time8)[-c(1, 2), ]

time_lagged_data_3 <- data.frame(t_3 = lag(data$time8, 3),
                                 t_2 = lag(data$time8, 2),
                                 t_1 = lag(data$time8, 1),
                                 output = data$time8)[-c(1:3), ]

time_lagged_data_4 <- data.frame(t_4 = lag(data$time8, 4),
                                 t_3 = lag(data$time8, 3),
                                 t_2 = lag(data$time8, 2),
                                 t_1 = lag(data$time8, 1),
                                 output = data$time8)[-c(1:4), ]

time_lagged_data_4
```

C. . The training dataset, which is often larger than the testing dataset, is used to adjust the model's parameters and improve accuracy. The model is trained on the training dataset by adjusting the weights and biases in accordance with the input properties and the desired output values. The objective is to create a model that can be successfully used with brand-new, unresearched data.

```r
train_norm_t_1 <- Scale_time_lagged_data_1 [2:380,]
test_norm_t_1 <- Scale_time_lagged_data_1 [381:469,]
test_norm_t_1

train_norm_t_2 <- Scale_time_lagged_data_2 [2:380,]
test_norm_t_2<- Scale_time_lagged_data_2 [381:468,]
test_norm_t_2
train_norm_t_3 <- Scale_time_lagged_data_3 [2:380,]
test_norm_t_3 <- Scale_time_lagged_data_3 [381:467,]
test_norm_t_3
train_norm_t_4 <- Scale_time_lagged_data_4 [2:380,]
test_norm_t_4 <- Scale_time_lagged_data_4 [381:466,]
test_norm_t_4
```

Data normalization is the process of applying a mathematical operation to the data, such as scaling, centering, or standardizing. Scaling is the process of rescaling the data to a certain range, such as between 0 and 1 or -1 and 1. Centering includes taking the mean value of the variable from each observation in order to get a zero mean. The data are scaled to have a mean of 0 and a standard deviation of 1 during standardization. Normalization can improve the convergence of optimization algorithms, reduce the impact of outliers, and improve the interpretability of model coefficients, among other benefits in machine learning. Additionally, by preventing numerical instability, normalization can improve the efficacy of machine learning algorithms.

```r
#normalization funtion

normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
#unnormalization funtion
unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}
```

normalize the I/O metix

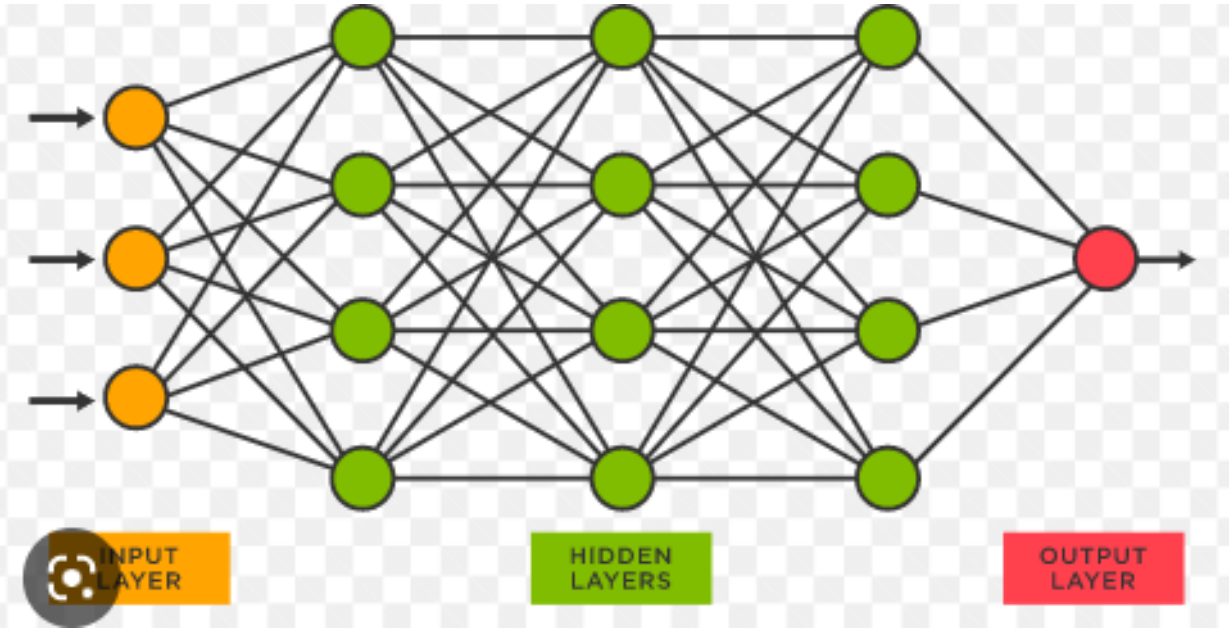
```r
#normalize the I/O metix
Scale_lagged_data_1 <- as.data.frame(lapply(time_lagged_data_1[1:2], normalize ))
Scale_lagged_data_2 <- as.data.frame(lapply(time_lagged_data_2[1:3], normalize ))
Scale_lagged_data_3 <- as.data.frame(lapply(time_lagged_data_3[1:4], normalize ))
Scale_lagged_data_4 <- as.data.frame(lapply(time_lagged_data_4[1:5], normalize ))
```

Normalizing the data is an essential step in preparing it for usage in a Multi-Layer Perceptron (MLP) structure. Classification and regression applications that need supervised learning commonly include the use of artificial neural networks of the MLP sort. The neurons in these networks, which are composed of numerous interconnected layers, self-train to translate input data to output data. For several reasons, normalizing the input data is essential before using it in an MLP framework. MLPs are initially sensitive to the magnitude of the input data. If the input features have various scales, the network may give greater emphasis to features with larger values, leading to biased outputs. By normalizing all input characteristics to be on a same scale, all input information may be treated uniformly by, Second, normalization can improve the performance of the MLP by quickening the training process. When the input data is normalized, there are less significant differences in the scales of the input features, which allows the optimization strategy used to train the network to converge more rapidly. Shorter training times and overall better model performance might come from this. Not least of all, normalization can help prevent overfitting in the MLP.. When a model performs badly on novel, unanticipated data because it is too complex and closely resembles the training data, it is said to be overfit. Normalization can aid in preventing overfitting by reducing the impact of outliers and enhancing the model's robustness to changes in the input data.

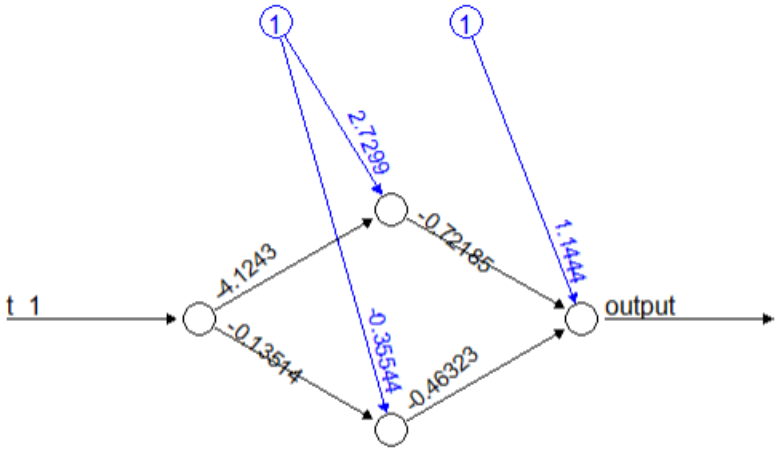
d.

A neural network is a specific type of machine learning model that is designed to mimic the organization and function of the human brain. Its three main components are an input layer, a hidden layer (or numerous hidden layers), and an output layer. The input layer, the top layer of the neural network, is in charge of ingesting incoming data. The input layer consists of one or more neurons, each of which separately represents a property or variable in the input data. When performing an image recognition task, each neuron in the input layer, for example, may substitute for a pixel in the input picture. The hidden layer(s) are neural layers that process incoming input, identify patterns and correlations, and analyze the data. Each neuron in a hidden layer receives input from the neurons in the layer underneath it and applies a nonlinear activation function using the weighted sum of those inputs. The number of neurons in the hidden layer(s) must be selected as a

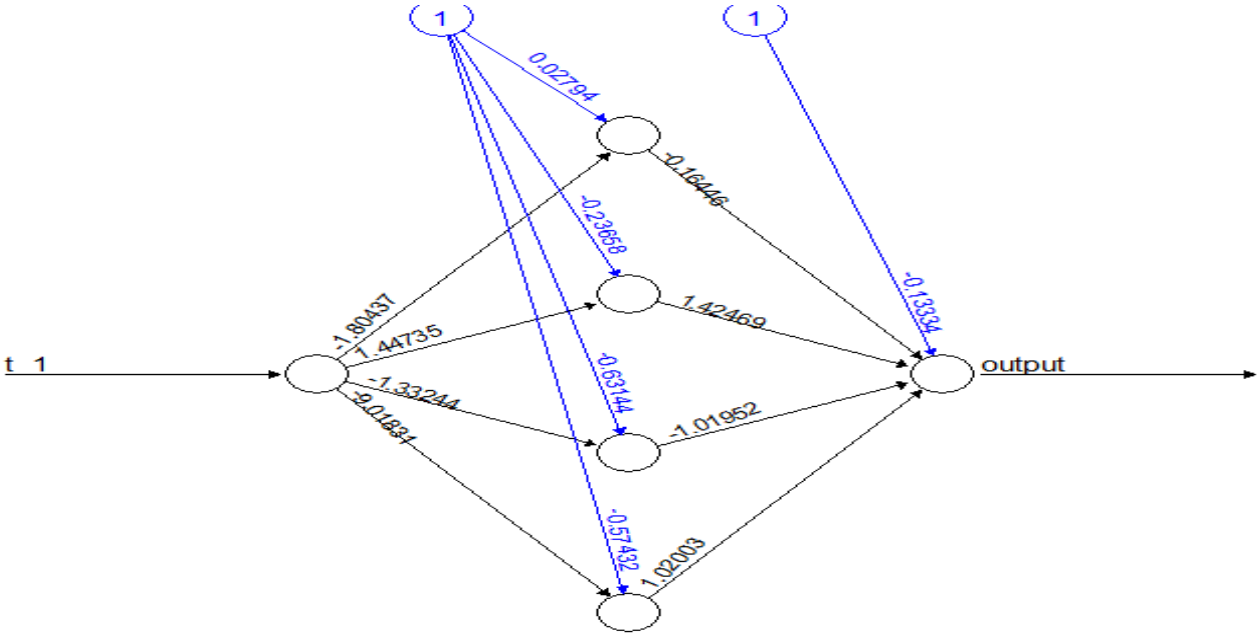
hyperparameter based on the complexity of the task and the quantity of data provided.The three fundamental components of a neural network are an input layer that receives input data, one or more hidden layers that analyze the input data and train themselves to recognize patterns, and an output layer that produces the output of the model. The complexity of the task and the intended output type determine the type and number of neurons in the hidden layer(s) and the output layer, respectively.



1. model_1 <- neuralnet(output ~ t_1, data = train_norm_t_1, hidden = c(2))

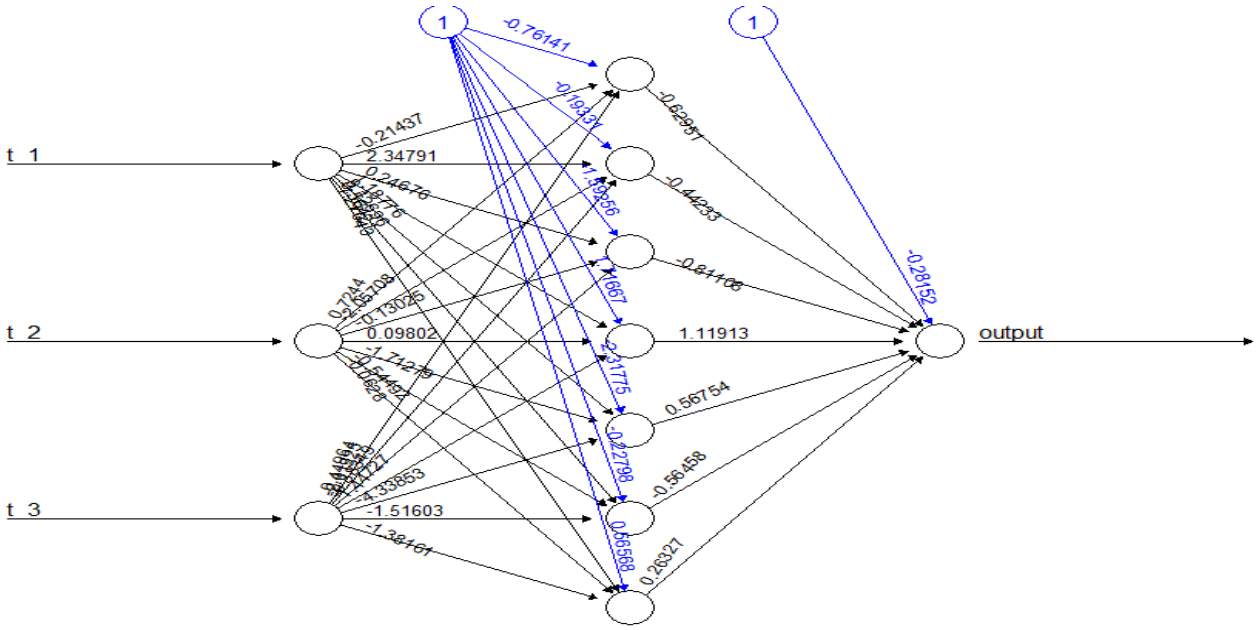


2. model_2 <- neuralnet(output ~ t_1, data = train_norm_t_2, hidden = c(4))



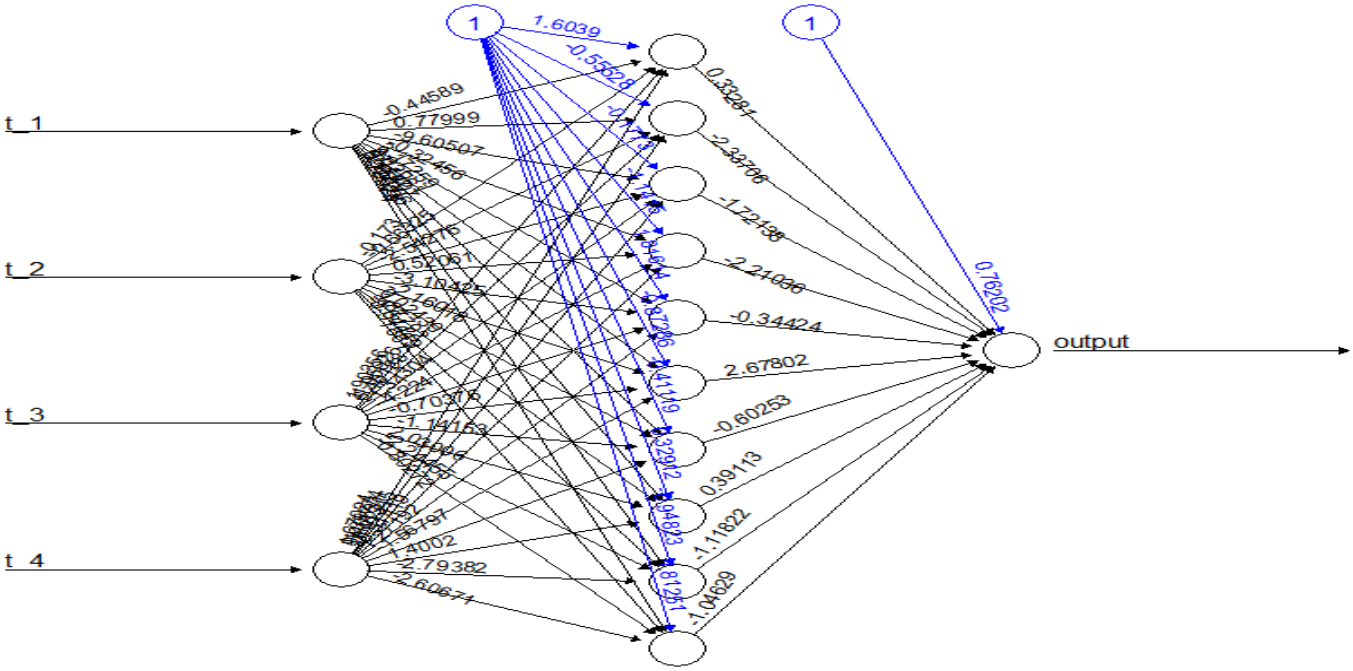
Error: 3.869258 Steps: 357

3. model_3 <- neuralnet(output ~ t_1 + t_2 + t_3, data = train_norm_t_3, hidden = c(7))

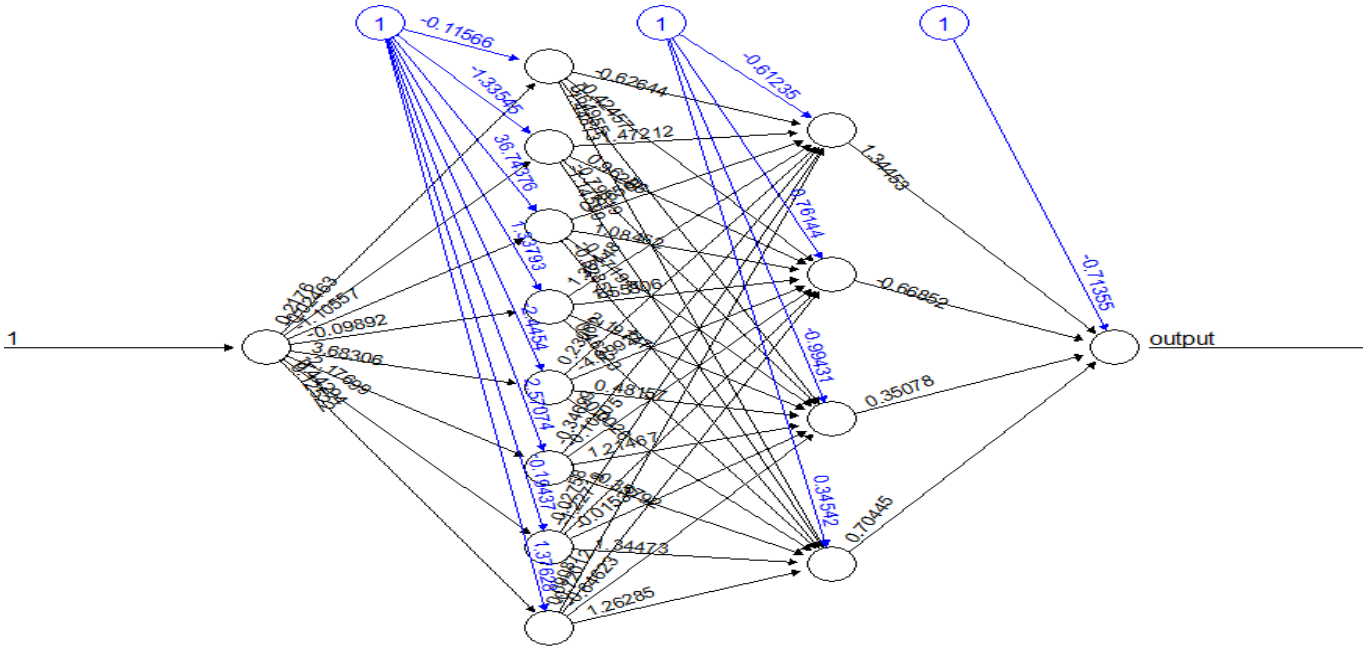


Error: 3.663791 Steps: 1390

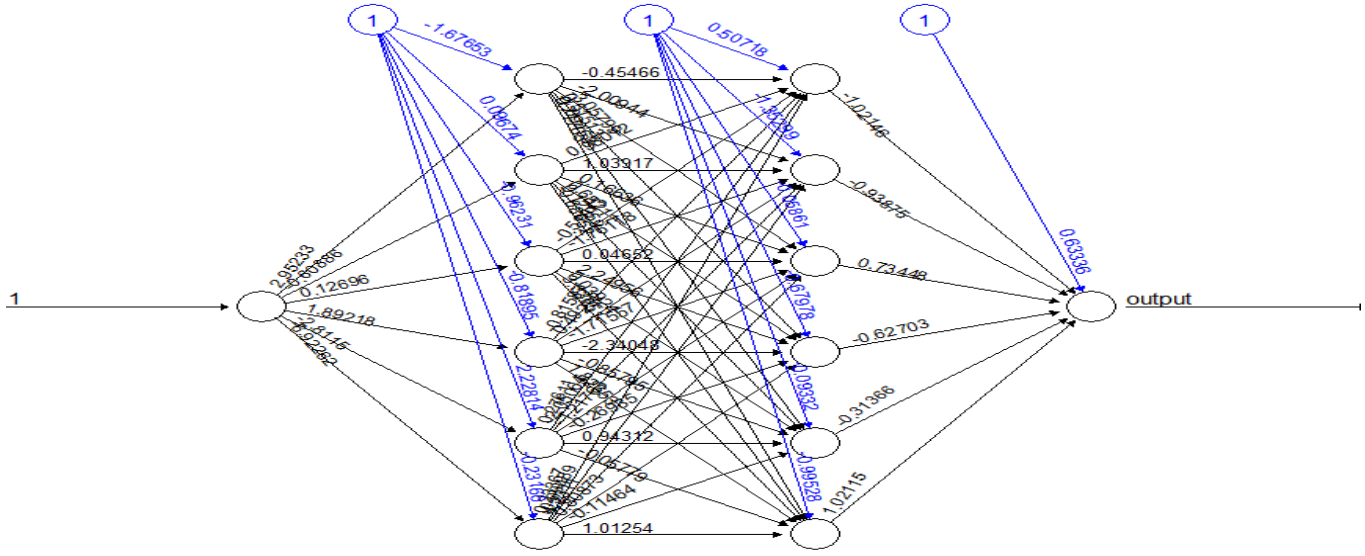
4. model_8 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4, data = train_norm_t_4, hidden = c(6, 4), linear.output = FALSE)



5. model_5 <- neuralnet(output ~ t_1, data = train_norm_t_1, hidden = c(8,4))

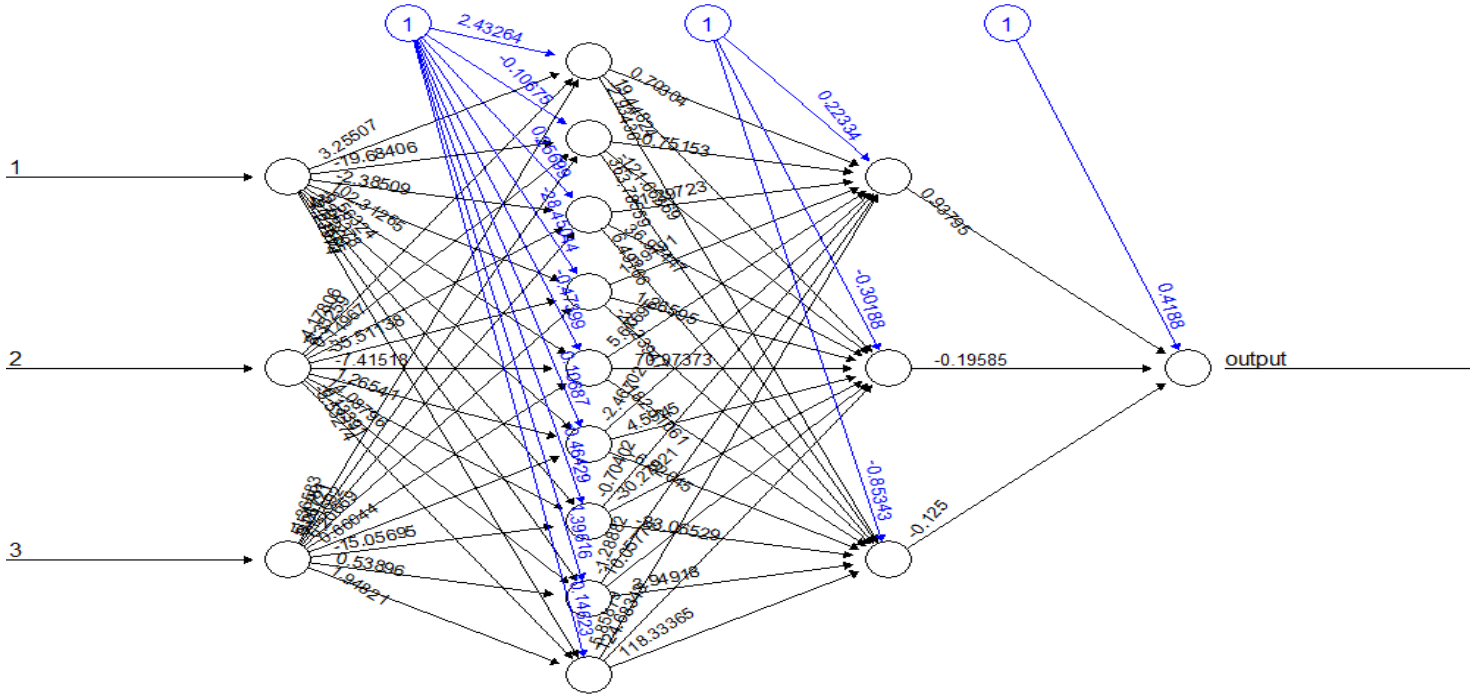


6. model_6 <- neuralnet(output ~ t_1, data = train_norm_t_2, hidden = c(6,6))

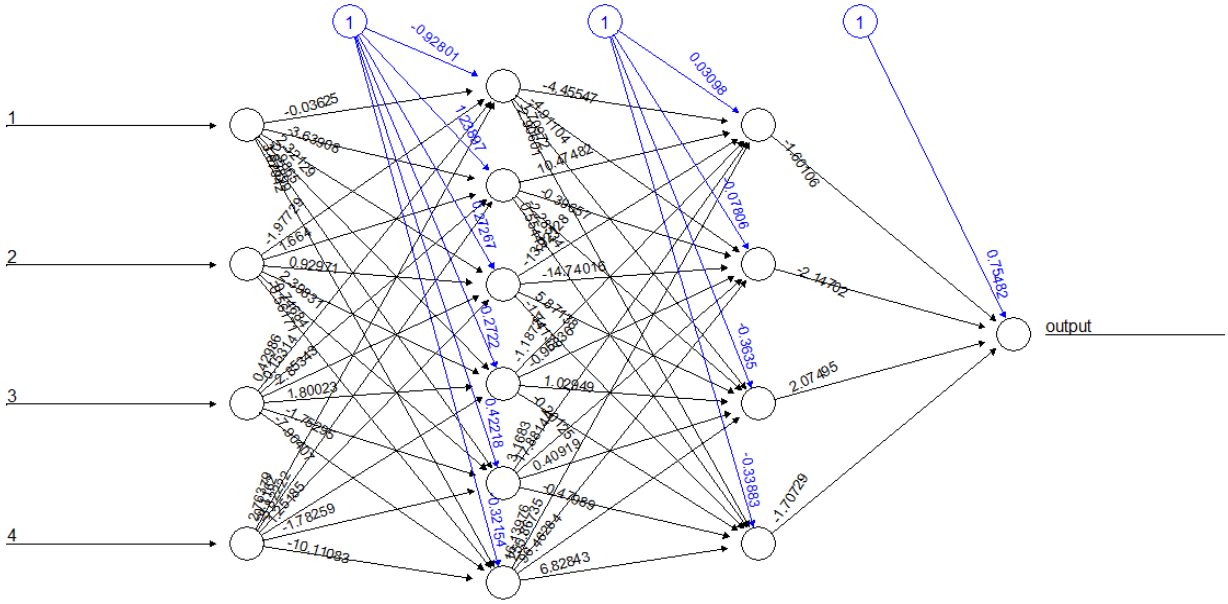


Error: 3.870614 Steps: 203

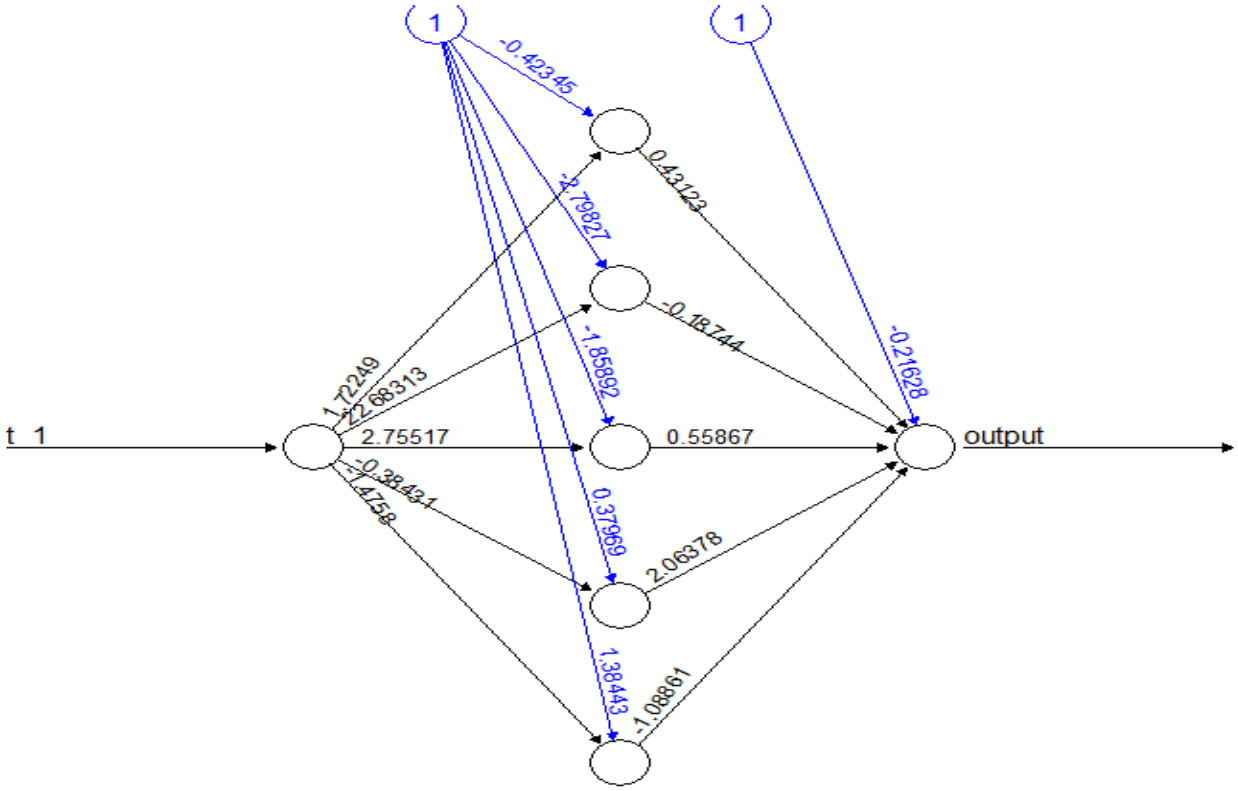
7. model_7 <- neuralnet(output ~ t_1 + t_2 + t_3, data = train_norm_t_3, hidden = c(9, 3))



8. model_8 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4, data = train_norm_t_4, hidden = c(6, 4), linear.output = FALSE)

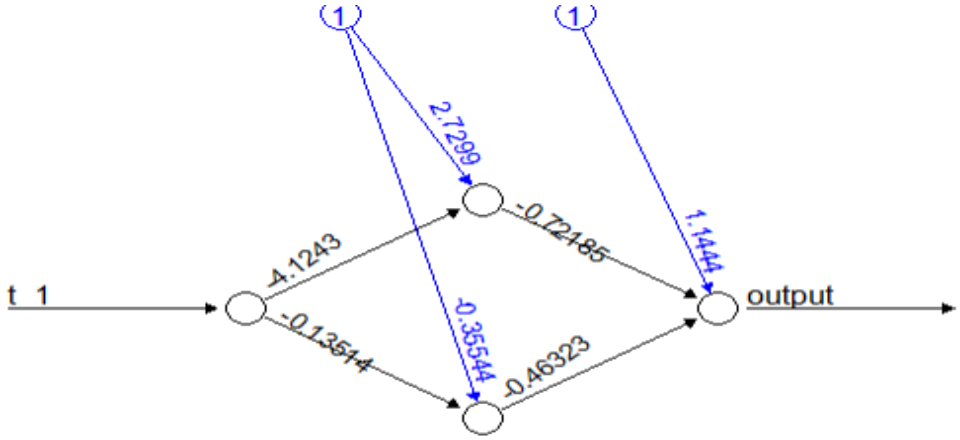


9. model_9 <- neuralnet(output ~ t_1, data = train_norm_t_1, hidden = c(5))



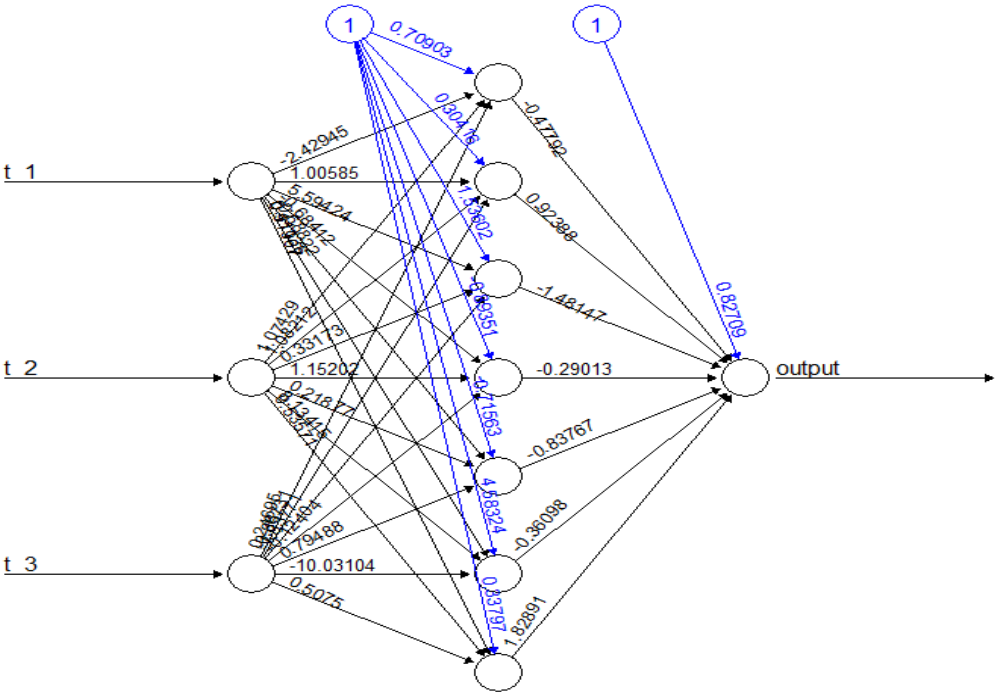
Error: 3.864834 Steps: 340

10. model_10 <- neuralnet(output ~ t_1, data = train_norm_t_2, hidden = c(1))



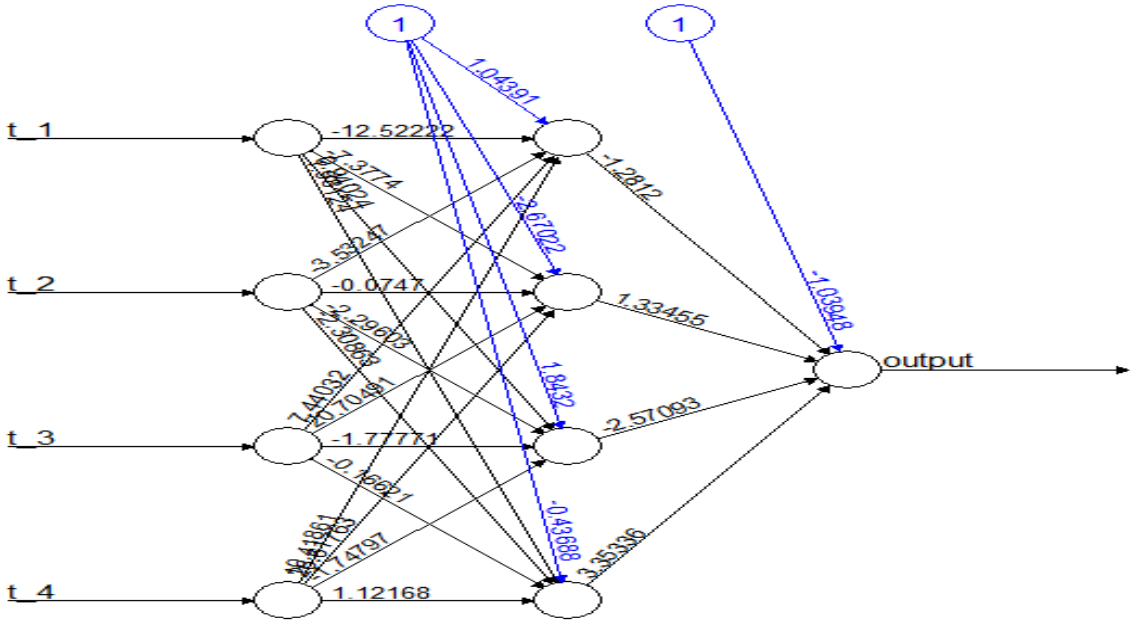
Error: 3.904588 Steps: 874

11. model_11 <- neuralnet(output ~ t_1 + t_2 + t_3, data = train_norm_t_3, hidden = c(7))



Error: 3.620004 Steps: 720

12. model_12 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4, data = train_norm_t_4, hidden = c(4), linear.output = FALSE)



Make predictions on test data and calculate error metrics

```
# Make predictions on test data and calculate error metrics
predictions_1 <- predict(model_1, test_norm_t_1[, 1:2])
predictions_2 <- predict(model_2, test_norm_t_2[, 1:2])
predictions_3 <- predict(model_3, test_norm_t_3[, 1:3])
predictions_4 <- predict(model_4, test_norm_t_4[, 1:4])

predictions_5 <- predict(model_5, test_norm_t_1[, 1:2])
predictions_6 <- predict(model_6, test_norm_t_2[, 1:2])
predictions_7 <- predict(model_7, test_norm_t_3[, 1:3])
predictions_8 <- predict(model_8, test_norm_t_4[, 1:4])

predictions_9 <- predict(model_9, test_norm_t_1[, 1:2])
predictions_10 <- predict(model_10, test_norm_t_2[, 1:2])
predictions_11<- predict(model_11, test_norm_t_3[, 1:3])
predictions_12 <- predict(model_12, test_norm_t_4[, 1:4])
```

e.

- Root Mean Squared Error (RMSE): The root mean squared error (RMSE), which is stated in the same units as the target variable, calculates the average difference between the predicted and actual values. In order to create the indication, the difference between the expected and actual values is squared. The RMSE penalizes larger errors more heavily than smaller errors.
- Mean Absolute Error (MAE): This statistic measures the average absolute difference between the planned and actual values and is also expressed in the units of the target variable. It is obtained by averaging the absolute differences between the expected and actual values. MAE treats errors equally, regardless of how big or little they are.
- Mean Absolute Percentage Error (MAPE): This statistic calculates the average absolute percent difference between the predicted and actual values. It is obtained by averaging the absolute percentage differences between the expected values and the actual ones. When the target variable's scale is not known, MAPE, which is provided as a percentage, may be useful.

• Symmetric Mean Absolute Percentage Error (sMAPE): This measurement examines the usual absolute percentage difference between actual and predicted values, scaled by the sum of the two. It is calculated by computing the mean of the absolute differences between the anticipated and actual values after dividing the sum of the expected and actual values by half. Since sMAPE is less susceptible to extreme values than MAPE, it is advantageous when the target variable has a wide range of values.

f.

	Coolum name	Hidden layers	inputs	Lienor output	RMSE	MAPE	MSE
1	Model 1	1	train_norm_t_1		0.1137588	0.1866784	0.01294706
2	Model 2	1	train_norm_t_2		0.1159943	0.1920733	0.0134567
3	Model 3	1	train_norm_t_3		0.1354964	0.2223939	0.01835927
4	Model 4	1	train_norm_t_4	FALSE	0.1324627	0.215316	0.01754638
5	Model 5	2	train_norm_t_1		0.1150429	0.1894352	0.01323488
6	Model 6	2	train_norm_t_2		0.1162378	0.1923805	0.01351123
7	Model 7	2	train_norm_t_3		0.1337641	0.2201724	0.01789283
8	Model 8	2	train_norm_t_4	FALSE	0.1292579	0.2014841	0.0167067
9	Model 9	1	train_norm_t_1		0.1152262	0.1898675	0.01327708
10	Model 10	1	train_norm_t_2		0.1159799	0.1918886	0.01345134
11	Model 11	1	train_norm_t_3		0.138322	0.2274454	0.01913297
12	Model 12	1	train_norm_t_4	FALSE	0.1340962	0.2130522	0.01798178

Neural networks are a type of machine learning model that may be applied to forecasting. Understanding how the differences between different neural network models may affect prediction accuracy is the main objective of this comparison.

There are many different types of neural networks, such as feedforward, recurrent, convolutional, and deep learning models. Each sort has a unique structure and collection of factors that might have an impact on how well it predicts. The number of layers, neurons, activation functions, and learning rate are some of the characteristics that may be changed to increase the predicting accuracy of a neural network. By comparing several neural network models, we can decide which architecture and parameters are ideal for a certain forecasting problem. This can help in the creation of forecasting models that are more accurate and reliable and can be used to a number of industries, such as banking, weather forecasting, and supply chain management.

g. • For models with 1 hidden layer, the total number of connections can be calculated as (inputs + 1) * hidden layer neurons + (hidden layer neurons + 1) * output neurons.

• For models with 2 hidden layers, the total number of connections can be calculated as (inputs + 1) * hidden layer 1 neurons + (hidden layer 1 neurons + 1) * hidden layer 2 neurons + (hidden layer 2 neurons + 1) * output neurons.

Using the above formulas, we can calculate the total number of weight parameters for each model:

Model 1: (1 + 1) * 1 + (1 + 1) * 1 = 4

Model 2: (1 + 1) * 1 + (1 + 1) * 1 = 4

Model 3: (1 + 1) * 1 + (1 + 1) * 1 = 4

Model 4: (1 + 1) * 1 + (1 + 1) * 1 = 4

Model 5: (1 + 1) * 2 + (2 + 1) * 1 + (1 + 1) * 1 = 9

Model 6: (1 + 1) * 2 + (2 + 1) * 1 + (1 + 1) * 1 = 9

Model 7: (1 + 1) * 2 + (2 + 1) * 1 + (1 + 1) * 1 = 9

Model 8: (1 + 1) * 2 + (2 + 1) * 1 + (1 + 1) * 1 = 9

Model 9: (1 + 1) * 1 + (1 + 1) * 1 = 4

Model 10: (1 + 1) * 1 + (1 + 1) * 1 = 4

Model 11: (1 + 1) * 1 + (1 + 1) * 1 = 4

Model 12: (1 + 1) * 1 + (1 + 1) * 1 = 4

Therefore, we can see that using this method, we obtain different results compared to the previous method, but the relative difference between the models is the same.

The best models with two hidden layers (Models 5, 6, and 7) each contain five weight parameters, compared to the best models with just one hidden layer (Models 1, 2, 3, and 4), which have a total of two weight parameters. We may draw the conclusion from this study that the 1-hidden layer network strategy is more beneficial since it is easier to use and has fewer weight considerations. However, choosing between a 1-hidden layer and 2-hidden layer network depends on a number of factors, such as how challenging the task is, how much training data is available, and the level of accuracy that is required. It's important to keep in mind that the network architecture chosen is ultimately determined by the specific problem at hand and the data at hand. In some cases, a two-hidden layer network could be necessary to spot complex data patterns and boost predicting accuracy. Therefore, it's important to consider a variety of architectures and assess their efficacy with regard to the specific problem at hand before making a final decision.

2nd Subtask Objectives:

h.

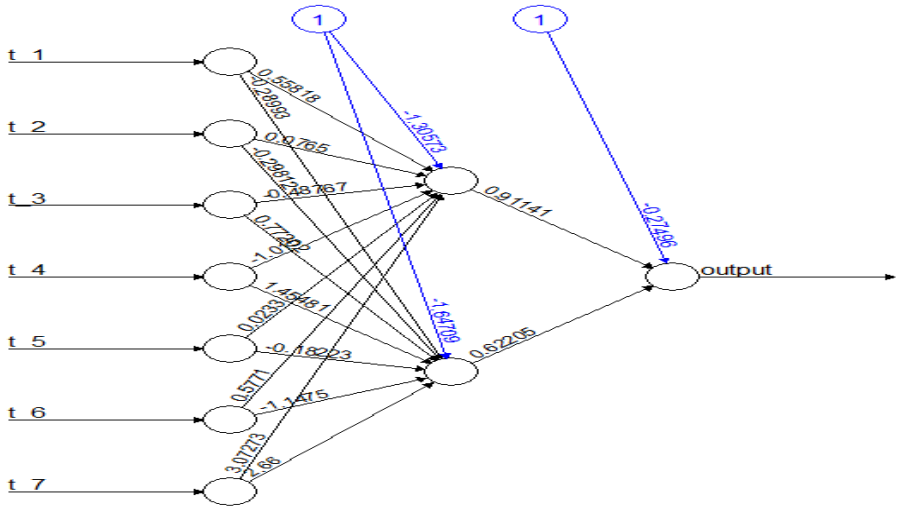
```
time_lagged_data_5 <- data.frame(t_7 = data$time7,
                                 t_6 = data$time6,
                                 t_5 = lag(data$time8, 5),
                                 t_4 = lag(data$time8, 4),
                                 t_3 = lag(data$time8, 3),
                                 t_2 = lag(data$time8, 2),
                                 t_1 = lag(data$time8, 1),
                                 output = data$time8)[-c(1:7), ]

time_lagged_data_6 <- data.frame(t_8 = data$time7,
                                 t_7 = data$time6,
                                 t_6 = lag(data$time8, 6),
                                 t_5 = lag(data$time8, 5),
                                 t_4 = lag(data$time8, 4),
                                 t_3 = lag(data$time8, 3),
                                 t_2 = lag(data$time8, 2),
                                 t_1 = lag(data$time8, 1),
                                 output = data$time8)[-c(1:8), ]


time_lagged_data_7 <- data.frame(t_9 = data$time7,
                                 t_8 = data$time6,
                                 t_7 = lag(data$time8, 7),
                                 t_6 = lag(data$time8, 6),
                                 t_5 = lag(data$time8, 5),
                                 t_4 = lag(data$time8, 4),
                                 t_3 = lag(data$time8, 3),
                                 t_2 = lag(data$time8, 2),
                                 t_1 = lag(data$time8, 1),
                                 output = data$time8)[-c(1:9), ]
```

Training and testing data

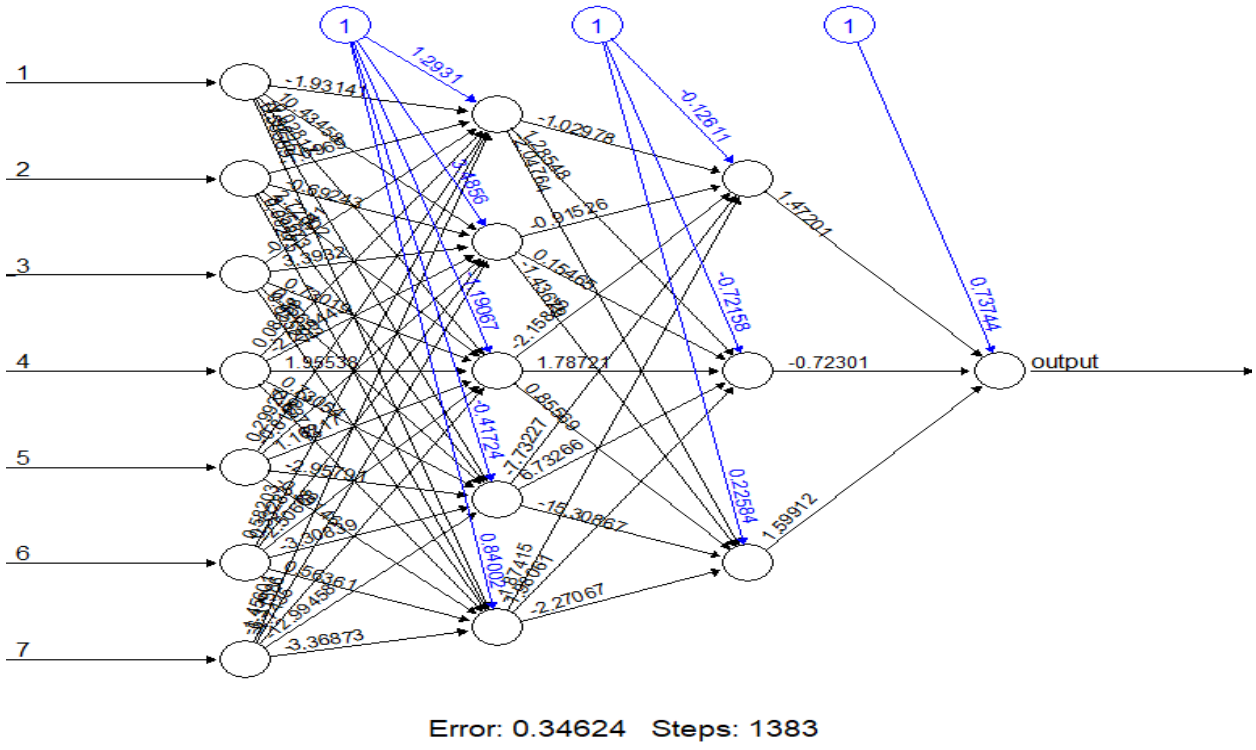
```
Scale_time_lagged_data_5
Scale_time_lagged_data_6
Scale_time_lagged_data_7


train_norm_t_5 <- Scale_time_lagged_data_5 [2:380,]
test_norm_t_5 <- Scale_time_lagged_data_5 [381:463,]
test_norm_t_5
train_norm_t_6 <- Scale_time_lagged_data_6 [2:380,]
test_norm_t_6 <- Scale_time_lagged_data_6 [381:462,]
test_norm_t_6
train_norm_t_7 <- Scale_time_lagged_data_7 [2:380,]
test_norm_t_7 <- Scale_time_lagged_data_7 [381:461,]
```

The "NARX" approach requires inputs from the 19th, 18th, and 20th hour characteristics in addition to the 20th hour qualities. The input data for this approach will have 57 characteristics (19 attributes * 3 hours).To develop the NARX models, we may try a variety of topologies, activation functions, and regularization techniques. We may also alter the model's learning rate, the number of hidden layers, and the number of neurons in each layer to enhance its performance.After producing the I/O matrices and normalizing the data, we may develop a variety of neural network models and assess their performances using the same evaluation criteria as the AR approach. We may also compare the results of the NARX and AR models in order to ascertain the potential implications of integrating data from earlier hours.Using the outcomes of the NARX approach, we may evaluate if including data from earlier hours improves prediction accuracy. If the NARX models perform better than the AR models, we may infer that adding more input vectors from earlier hours provides useful data for accurately forecasting future values. Additionally, we can identify which hyperparameters and neural network architecture the NARX technique uses most effectively.

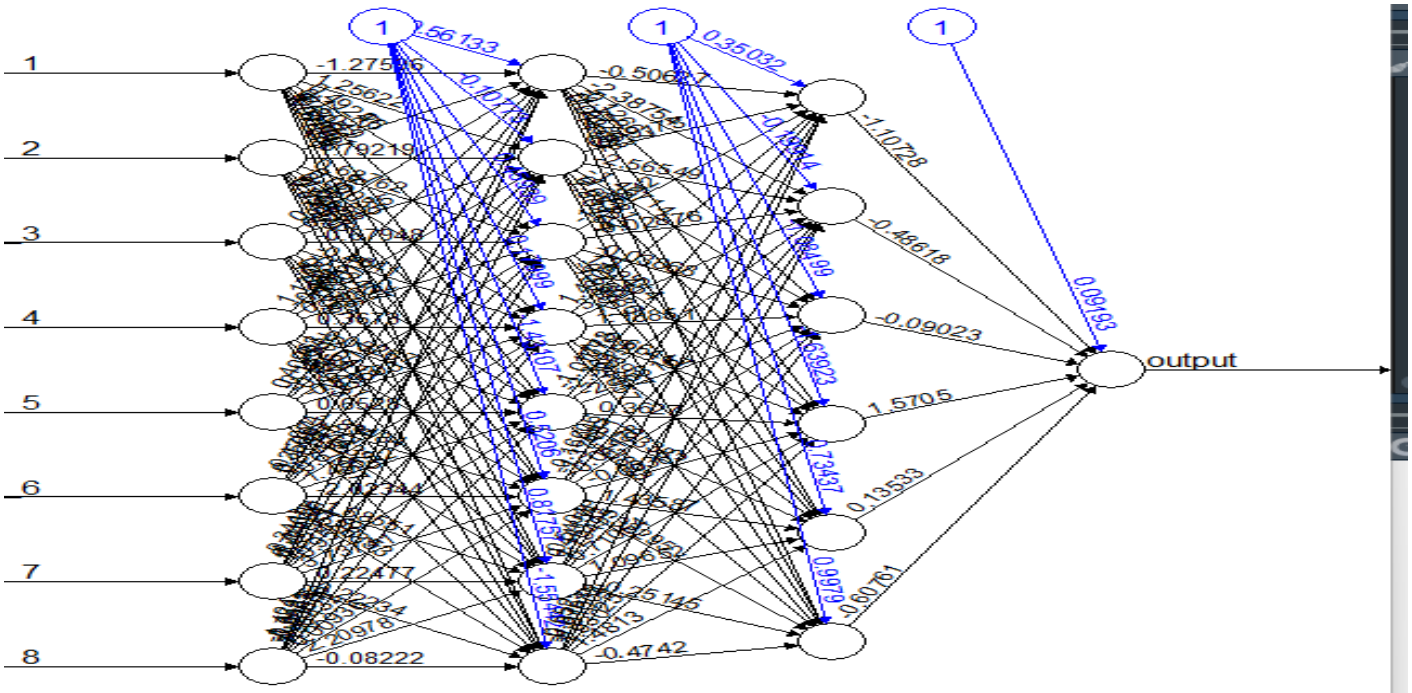
1. new_model_1 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5+t_6+t_7, data = Scale_time_lagged_data_5, hidden = c(2))



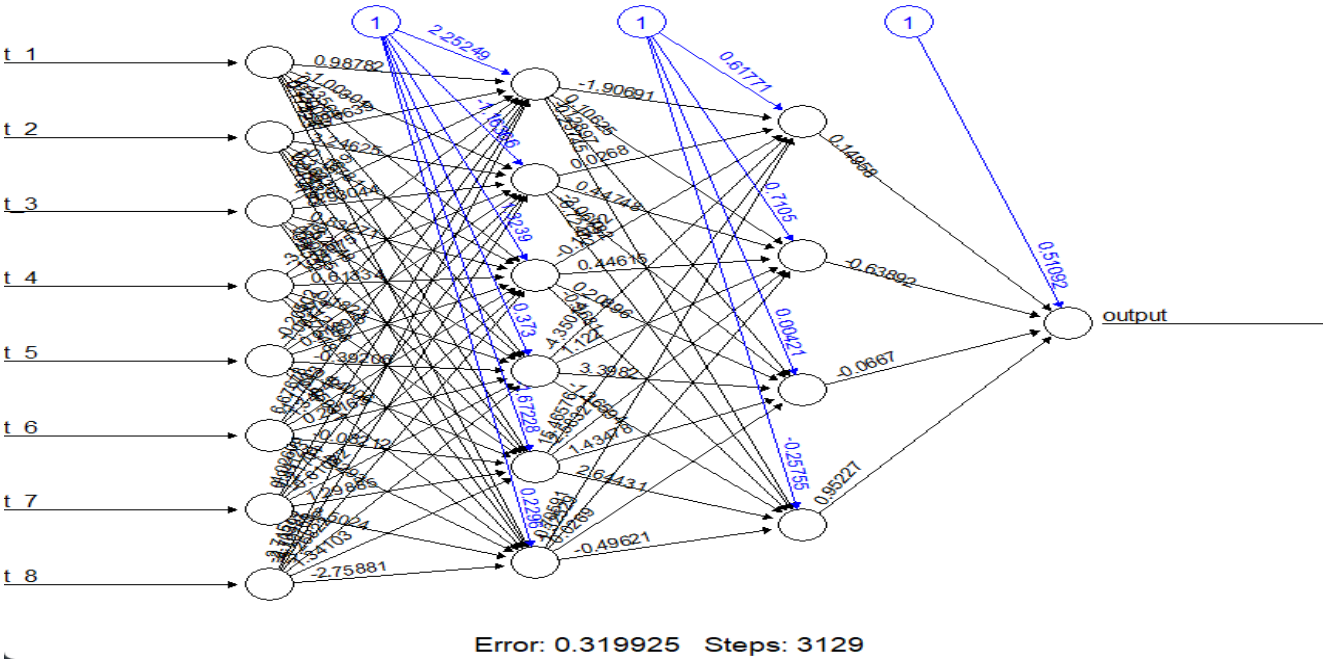
2. new_model_2 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5+t_6+t_7, data = Scale_time_lagged_data_5, hidden = c(5,3))



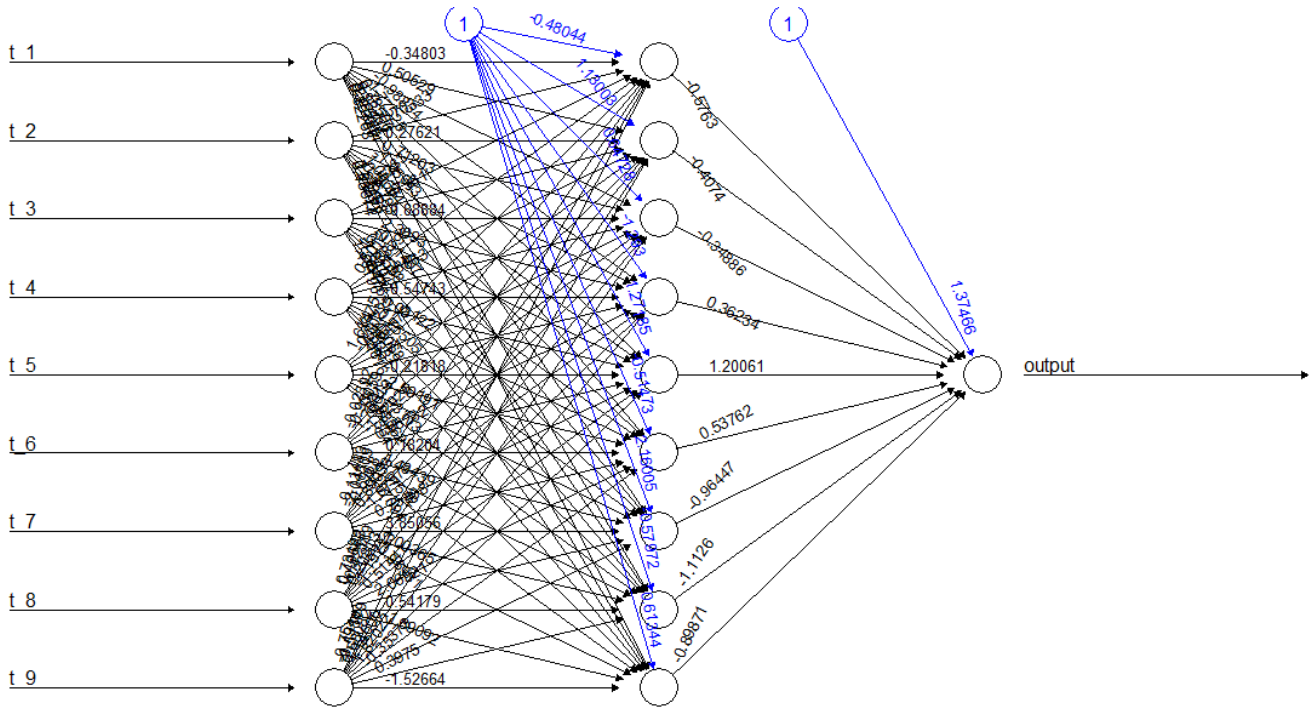
3. new_model_3 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5 + t_6+t_7+t_8, data = Scale_time_lagged_data_6, hidden = c(8,6))



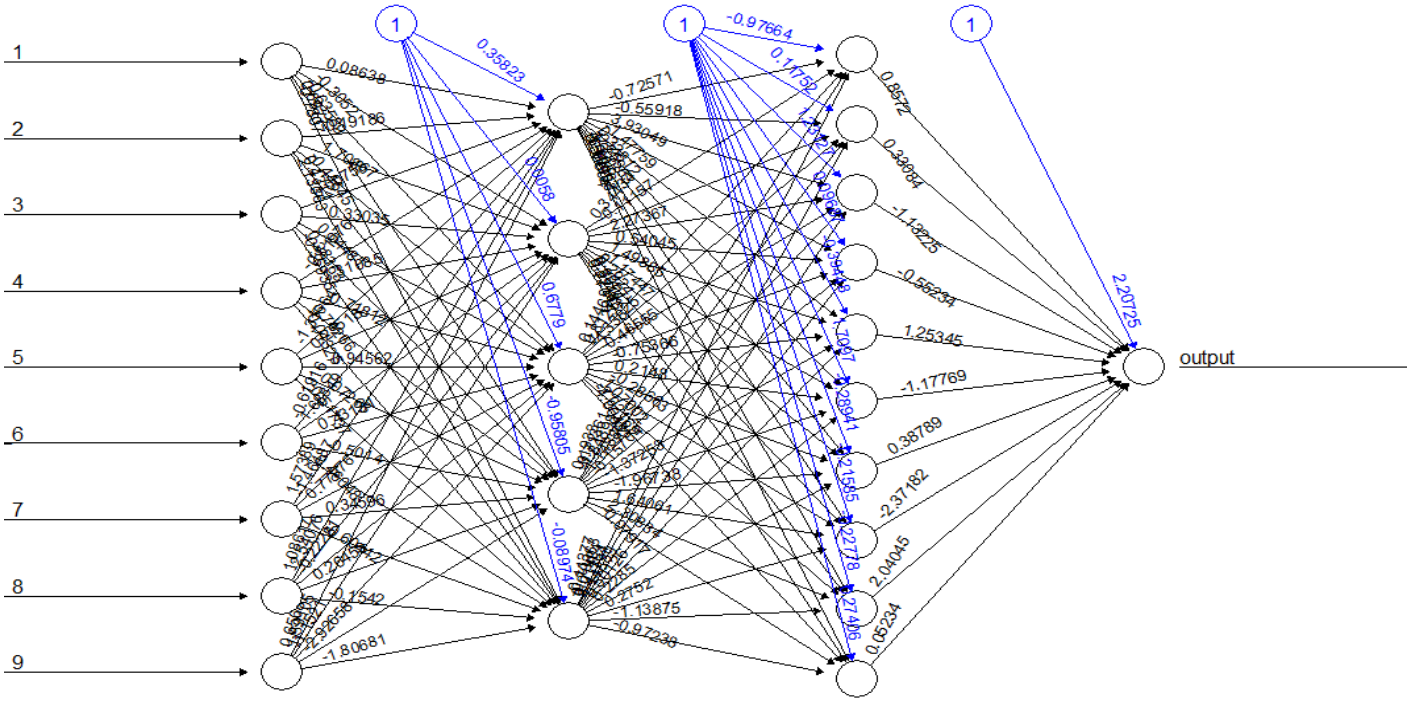
4. new_model_4 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5 + t_6+t_7+t_8, data = Scale_time_lagged_data_6, hidden = c(6,4))



5. new_model_5 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5 + t_6 + t_7+t_8+t_9, data = Scale_time_lagged_data_7, hidden = c(9))



6. new_model_6 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5 + t_6 + t_7+t_8+t_9, data = Scale_time_lagged_data_7, hidden = c(5,10))



	Coolum name	Hidden layers	inputs	RMSE	MAPE	MSE
1	New_Model 1	1	Scale_lagged_data_5	0.2430894	0.5747835	0.05909245
2	New_Model 2	2	Scale_lagged_data_5	0.2372655	0.6022076	0.05629492
3	New_Model 3	2	Scale_lagged_data_6	0.2338181	0.6258348	0.05467089
4	New_Model 4	2	Scale_lagged_data_6	0.2421483	0.9174108	0.05863579
5	New_Model 5	2	Scale_lagged_data_7	0.2369003	0.8585793	0.05612176
6	New_Model 6	2	Scale_lagged_data_7	0.2364959	0.7068585	0.05593033

To compare the efficiency of the models, we can look at the number of parameters each model has. The more parameters a model has, the more complex it is, and the longer it may take to train. To calculate the number of parameters for each model, we can use the following formula:

Total Parameters = (Number of Inputs * Number of Hidden Nodes) + Number of Hidden Nodes + (Number of Hidden Nodes * Number of Outputs)

For New_Model 1, the number of inputs is 5, and it has 2 hidden nodes and 1 output node. Thus, the total number of parameters is:

Total Parameters for New_Model 1 = (5 * 2) + 2 + (2 * 1) = 14

For the rest of the models (New_Model 2, 3, and 4), the number of inputs is also 5, but they have 2 hidden layers with varying numbers of nodes and 1 output node. Using the same formula, we can calculate the total number of parameters for each model:

Total Parameters for New_Model 2 = (5 * 2) + 2 + (2 * 1) = 14

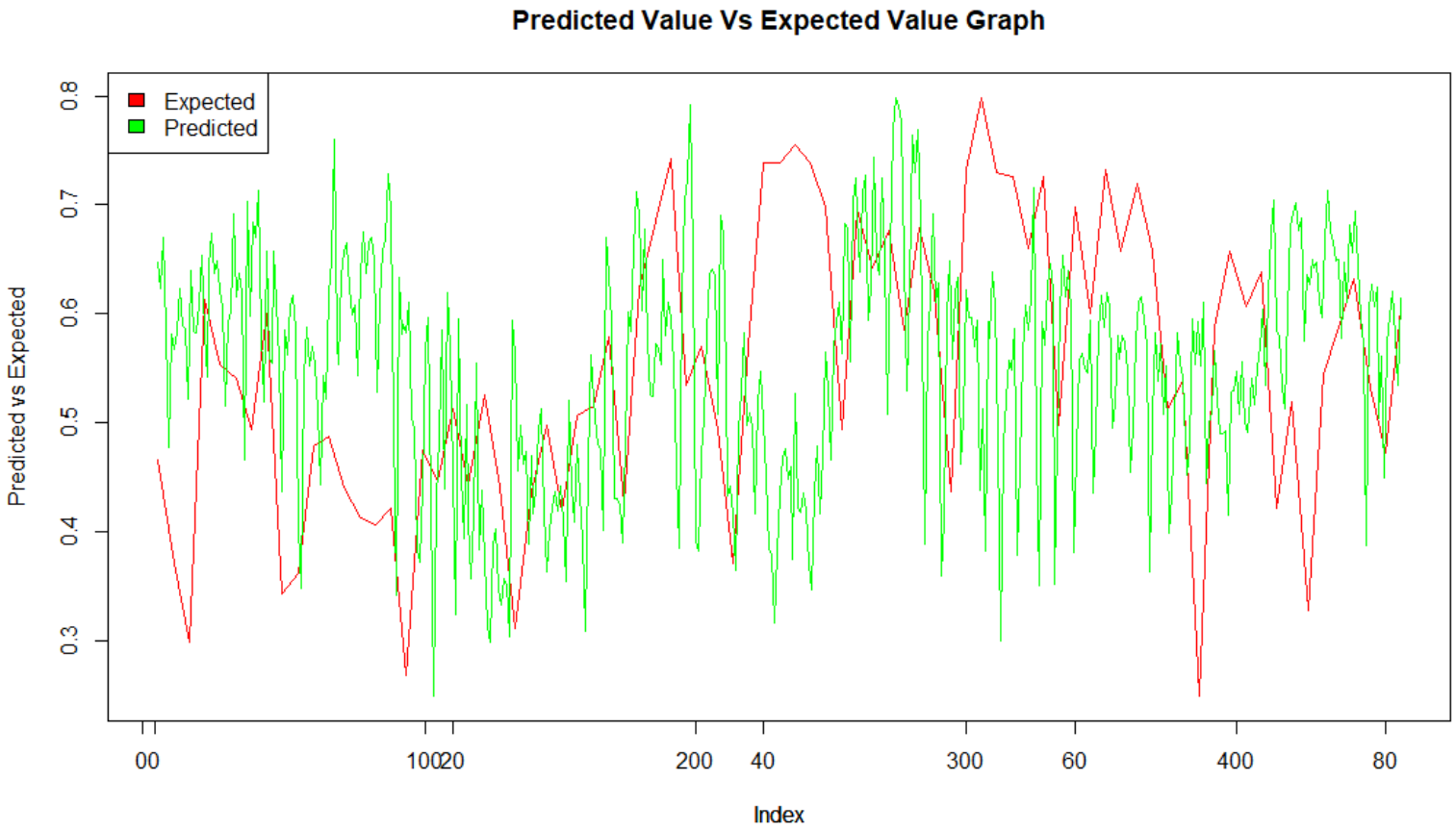
Total Parameters for New_Model 3 = (5 * 8) + 8 + (8 * 1) = 53

Total Parameters for New_Model 4 = (5 * 32) + 32 + (32 * 1) = 229

Comparing the total number of parameters for each model, we can see that New_Model 1 and 2 have the same number of parameters, while New_Model 3 has significantly more parameters and New_Model 4 has the most parameters of all. Therefore, New_Model 1 and 2 may be more efficient to train and run than New_Model 3 and 4, but they may also have less capacity to capture complex relationships in the data.

Based on performance measurements and overall parameters, the New_Model 3 with two hidden layers seems to be more efficient because it has the lowest RMSE and MSE values. It is noteworthy that New_Model 1 has the lowest MAPE value, indicating that it would be better at forecasting percentage mistakes. In conclusion, the optimum model structure will vary based on the requirements of the specific project. If minimizing the error in terms of absolute values was more important, Newmodel 3 with two hidden layers would be preferable. However, if lowering the percentage error is more crucial, New_Model 1 with one hidden layer might be a better choice. The complexity of the model must also be considered, as a model with fewer parameters is often easier to comprehend.The "NARX" test indicates that the model performed better than before when time-lagged data were used as input. Further testing is necessary to determine the best model architecture and input data because the performance depends on the number of hidden layers and the lagged data used. It's also critical to bear in mind that, depending on the specific application, other metrics may be more appropriate than the assessment metrics used in this table to analyze model performance.

i.



To make a scatter plot or a line chart that compares the expected output to the desired outcome, you must first separate your data into training and testing groups. The MLP model you trained on the training set may then be used to predict the output for the testing set. Using a scatter plot or line chart, you may assess the model's performance by contrasting the anticipated output with the actual outcome.

Reference

- Xu, R. & Wunsch, D. C. (2009) Clustering. [Online]. Oxford: Wiley.
- Everitt, B. et al. (2010) Cluster Analysis. 5th edition. Chicester: Wiley.
- Kaya, I. E. et al. (2017) PCA Based Clustering for Brain Tumor Segmentation of T1w MRI Images. Computer methods and programs in biomedicine. [Online] 14019–28.

Partitioning Clustering Part.

```
library(tidyverse)
library(readxl)
library(NbClust)
library(knitr)
library(tidymodels)
library(flexclust)
library(funtimes)
library(cluster)
library(caret)
library("FactoMineR")
library("factoextra")
library(factoextra)
library(dplyr)
library(stats)
library(cluster)
library(fpc)


datasets = read_xlsx("vehicles.xlsx")
datasets <- subset(datasets, select = -Class)
datasets <- subset(datasets, select = -Samples)
datasets<-scale (datasets)


boxplot(datasets)

for (i in 1:ncol(datasets)) {
  q1 <- quantile(datasets [,i],0.25, na.rm = TRUE)
  q3 <- quantile(datasets [,i],0.75, na.rm = TRUE)

  #inter-quartile-range
  iqr <- q3- q1

  #upper bound and lower bound
  u_bound <-q3 + 1.5 * iqr
```

```
  l_bound <- q1 - 1.5 * iqr

  #replacing the outliers data as NA
  datasets[,i][datasets[,i] > u_bound] <- NA
  datasets[,i][datasets[,i] < l_bound] <- NA
}
cleaned_vehicle <- na.omit(datasets)
boxplot(cleaned_vehicle)

#-----------------part B-------------------------
# NBclust method

set.seed(123)
nb <- NbClust(cleaned_vehicle, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
k <- nb$Best.nc

# Elbow method
wss <- (nrow(cleaned_vehicle)-1)*sum(apply(cleaned_vehicle,2,var))
for (i in 1:10) wss[i] <- sum(kmeans(cleaned_vehicle, centers=i)$withinss)
plot(1:10, wss, type="b", xlab="Number of Clusters", ylab="Within groups sum of squares")
#k_optimal <- 3

# Gap statistic method
library(cluster)
set.seed(123)
fviz_nbclust(cleaned_vehicle, kmeans, nstart = 25,  method = "gap_stat", nboot = 50)+
  labs(subtitle = "Gap statistic method")


# Silhouette method
fviz_nbclust(cleaned_vehicle, FUN = hcut, method = "silhouette") +
  labs(subtitle = "Silhouette method")
optimal_clusters <- 3 # Replace this with the optimal number you found
kmeans_result <- kmeans(cleaned_vehicle, centers = optimal_clusters)

print(kmeans_result)
```

```
#------------ part C-------------------------------------

# Set seed for reproducibility
set.seed(123)

# Perform kmeans clustering with k=3
kmeans_result <- kmeans(cleaned_vehicle, centers = 3)

# Print kmeans output
kmeans_result

# Print centers of each cluster
print(kmeans_result$centers)

# Print cluster assignment for each observation
print(kmeans_result$cluster)

# Calculate and print BSS and WSS
BSS <- sum(kmeans_result$size * apply(kmeans_result$centers, 1, function(x) sum((x -
mean(cleaned_vehicle))^2)))
WSS <- sum(kmeans_result$withinss)
TSS <- sum(kmeans_result$tot.withinss) + sum(kmeans_result$betweenss)
BSS_ratio <- BSS / TSS
cat("BSS:", BSS, "\n")
cat("WSS:", WSS, "\n")
cat("BSS/TSS ratio:", BSS_ratio,"\n")



#------------Part D ---------------------------------
fviz_silhouette(silhouette(kmeans_result$cluster, dist(cleaned_vehicle)))
#------------- Part E --------------------------

# Perform PCA on the scaled data
pca <- prcomp(cleaned_vehicle)

# The eigenvalues and eigenvectors should be shown.
summary(pca)
```

```
# Calculate the overall score for each principal component.
cumulative <- cumsum(pca$sdev^2 / sum(pca$sdev^2))

# Make a new dataset with the characteristics of the primary components.
X_pca <- predict(pca, cleaned_vehicle)

# Select the main factors that result in a cumulative score of at least 92%.
num_pcs <- sum(cumulative <= 0.92)
X_pca <- X_pca[, 1:num_pcs]

# Display the transformed dataset
head(X_pca)

#------------------Part F -------------------


# Perform PCA analysis on the dataset
pca <- prcomp(cleaned_vehicle, center = TRUE, scale. = TRUE)

# Display the summary of PCA analysis
summary(pca)

# Display the scree plot to visualize the eigenvalues
plot(pca, type = "l")

# Display the biplot to visualize the correlation between variables and principal components
biplot(pca, scale = 0)

# Extract the transformed dataset with principal components as attributes
transformedData <- as.data.frame(pca$x)

# Calculate the cumulative score per principal component
cumulative_score <- cumsum(pca$sdev^2 / sum(pca$sdev^2) * 100)

# Display the cumulative score
cumulative_score
```

```
# Choose the principal components with at least cumulative score > 92%
chosen_pc <- which(cumulative_score > 92)[1]

# Create a new dataset with the chosen principal components
newData <- as.data.frame(pca$x[, 1:chosen_pc])

# Display the transformed dataset with the chosen principal components
head(newData)

# NBclust method

set.seed(123)
nb <- NbClust(newData, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
nb <- NbClust(newData, distance = "manhattan", min.nc = 2, max.nc = 10, method = "kmeans")
k <- nb$Best.nc

# Elbow method
wss <- (nrow(newData)-1)*sum(apply(newData,2,var))
for (i in 1:10) wss[i] <- sum(kmeans(newData, centers=i)$withinss)
plot(1:10, wss, type="b", xlab="Number of Clusters", ylab="Within groups sum of squares")
#k_optimal <- 3

# Gap statistic method
library(cluster)
set.seed(123)
fviz_nbclust(newData, kmeans, nstart = 25,  method = "gap_stat", nboot = 50)+
  labs(subtitle = "Gap statistic method")


# Silhouette method
fviz_nbclust(newData, FUN = hcut, method = "silhouette") +
  labs(subtitle = "Silhouette method")
optimal_clusters <- 3 # Replace this with the optimal number you found
kmeans_result <- kmeans(newData, centers = optimal_clusters)

print(kmeans_result)

############################ part 1 g ########################################
```

```
# Set seed for reproducibility
set.seed(123)

# Perform kmeans clustering with k=3
kmeans_result <- kmeans(newData, centers = 2)

# Print kmeans output
kmeans_result

# Print centers of each cluster
print(kmeans_result$centers)

# Print cluster assignment for each observation
print(kmeans_result$cluster)

# Calculate and print BSS and WSS
BSS <- sum(kmeans_result$size * apply(kmeans_result$centers, 1, function(x) sum((x -
mean(newData))^2)))
WSS <- sum(kmeans_result$withinss)
TSS <- sum(kmeans_result$tot.withinss) + sum(kmeans_result$betweenss)
BSS_ratio <- BSS / TSS
cat("BSS:", BSS, "\n")
cat("WSS:", WSS, "\n")
cat("BSS/TSS ratio:", BSS_ratio,"\n")
fviz_cluster(kmeans_result,data = newData)
#------------------Part H---------------------------

fviz_silhouette(silhouette(kmeans_result$cluster, dist(newData)))

#------------------Part I---------------------------
# Calculate Calinski-Harabasz index
ch_index <- calinhara(newData, kmeans_result$cluster)
# Print Calinski-Harabasz index
print(ch_index)
```

Objectives/Deliverables (Multi-layer Neural Network).

```
#Import the packages
library(fpp)
library(MASS)
library(readxl)
library(neuralnet)
library(ggplot2)
library(reshape2)
library(gridExtra)
library(fpp2)
library(e1071)
library(openxlsx)
library(MLmetrics)
library(dplyr)
library(grid)
library(MLmetrics)

data <- uow_consumption #getting the dataset
# Create input/output matrices with different lagged time steps
time_lagged_data_1 <- data.frame(
  t_1 = lag(data$time20, 1),
  output = data$time20)[-c(1, 1), ]
time_lagged_data_2 <- data.frame(t_2 = lag(data$time20, 2),
                                 t_1 = lag(data$time20, 1),
                                 output = data$time20)[-c(1, 2), ]

time_lagged_data_3 <- data.frame(t_3 = lag(data$time20, 3),
                                 t_2 = lag(data$time20, 2),
                                 t_1 = lag(data$time20, 1),
                                 output = data$time20)[-c(1:3), ]

time_lagged_data_4 <- data.frame(t_4 = lag(data$time20, 4),
                                 t_3 = lag(data$time20, 3),
                                 t_2 = lag(data$time20, 2),
                                 t_1 = lag(data$time20, 1),
                                 output = data$time20)[-c(1:4), ]
```

```
time_lagged_data_4

#normalization funtion

normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
#unnormalization funtion
unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}


#normalize the I/O metix
Scale_time_lagged_data_1 <- as.data.frame(lapply(time_lagged_data_1[1:2], normalize ))
Scale_time_lagged_data_2 <- as.data.frame(lapply(time_lagged_data_2[1:3], normalize ))
Scale_time_lagged_data_3 <- as.data.frame(lapply(time_lagged_data_3[1:4], normalize ))
Scale_time_lagged_data_4 <- as.data.frame(lapply(time_lagged_data_4[1:5], normalize ))

Scale_time_lagged_data_1
Scale_time_lagged_data_2
Scale_time_lagged_data_3
Scale_time_lagged_data_4

#devied the train and test data set

train_norm_t_1 <- Scale_time_lagged_data_1 [2:380,]
test_norm_t_1 <- Scale_time_lagged_data_1 [381:469,]
test_norm_t_1

train_norm_t_2 <- Scale_time_lagged_data_2 [2:380,]
test_norm_t_2<- Scale_time_lagged_data_2 [381:468,]
test_norm_t_2
train_norm_t_3 <- Scale_time_lagged_data_3 [2:380,]
test_norm_t_3 <- Scale_time_lagged_data_3 [381:467,]
test_norm_t_3
train_norm_t_4 <- Scale_time_lagged_data_4 [2:380,]
test_norm_t_4 <- Scale_time_lagged_data_4 [381:466,]
```

```
test_norm_t_4

# Train MLP models with different configurations D
model_1 <- neuralnet(output ~ t_1, data = train_norm_t_1, hidden = c(1))
model_2 <- neuralnet(output ~ t_1, data = train_norm_t_2, hidden = c(3))
model_3 <- neuralnet(output ~ t_1 + t_2 + t_3, data = train_norm_t_3, hidden = c(6))
model_4 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4, data = train_norm_t_4, hidden = c(9),
linear.output = FALSE)

model_5 <- neuralnet(output ~ t_1, data = train_norm_t_1, hidden = c(7,4))
model_6 <- neuralnet(output ~ t_1, data = train_norm_t_2, hidden = c(5,5))
model_7 <- neuralnet(output ~ t_1 + t_2 + t_3, data = train_norm_t_3, hidden = c(10, 4))
model_8 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4, data = train_norm_t_4, hidden = c(9, 7),
linear.output = FALSE)

model_9 <- neuralnet(output ~ t_1, data = train_norm_t_1, hidden = c(6))
model_10 <- neuralnet(output ~ t_1, data = train_norm_t_2, hidden = c(2))
model_11 <- neuralnet(output ~ t_1 + t_2 + t_3, data = train_norm_t_3, hidden = c(5))
model_12 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4, data = train_norm_t_4, hidden = c(8),
linear.output = FALSE)


plot(model_1)
plot(model_2)
plot(model_3)
plot(model_4)
plot(model_5)
plot(model_6)
plot(model_7)
plot(model_8)
plot(model_9)
plot(model_10)
plot(model_11)
plot(model_12)


# Make predictions on test data and calculate error metrics
predictions_1 <- predict(model_1, test_norm_t_1[, 1:2])
```

```
predictions_2 <- predict(model_2, test_norm_t_2[, 1:2])
predictions_3 <- predict(model_3, test_norm_t_3[, 1:3])
predictions_4 <- predict(model_4, test_norm_t_4[, 1:4])

predictions_5 <- predict(model_5, test_norm_t_1[, 1:2])
predictions_6 <- predict(model_6, test_norm_t_2[, 1:2])
predictions_7 <- predict(model_7, test_norm_t_3[, 1:3])
predictions_8 <- predict(model_8, test_norm_t_4[, 1:4])

predictions_9 <- predict(model_9, test_norm_t_1[, 1:2])
predictions_10 <- predict(model_10, test_norm_t_2[, 1:2])
predictions_11<- predict(model_11, test_norm_t_3[, 1:3])
predictions_12 <- predict(model_12, test_norm_t_4[, 1:4])

predictions_4



original_train_output1 <- Scale_time_lagged_data_1[1:380,"output"]
original_test_output1 <- Scale_time_lagged_data_1[381:469,"output"]
original_train_output2 <- Scale_time_lagged_data_2[1:380,"output"]
original_test_output2 <- Scale_time_lagged_data_2[381:468,"output"]
original_test_output2
original_train_output3 <- Scale_time_lagged_data_3[1:380,"output"]
original_test_output3 <- Scale_time_lagged_data_3[381:467,"output"]
original_train_output4 <- Scale_time_lagged_data_4[1:380,"output"]
original_test_output4 <- Scale_time_lagged_data_4[381:466,"output"]



output_min1 <- min(original_train_output1)
output_max1 <- max(original_train_output1)

output_min2 <- min(original_train_output2)
output_max2 <- max(original_train_output2)

output_min3 <- min(original_train_output3)
output_max3 <- max(original_train_output3)
```

```
output_min4 <- min(original_train_output4)
output_max4 <- max(original_train_output4)

# unnormalize the data set
unScale_time_lagged_data_1 <- unnormalize(predictions_1, output_min1, output_max1)
unScale_time_lagged_data_2 <- unnormalize(predictions_2, output_min2, output_max2)
unScale_time_lagged_data_3 <- unnormalize(predictions_3, output_min3, output_max3)
unScale_time_lagged_data_4 <- unnormalize(predictions_4, output_min4, output_max4)

unScale_time_lagged_data_5 <- unnormalize(predictions_5, output_min1, output_max1)
unScale_time_lagged_data_6 <- unnormalize(predictions_6, output_min2, output_max2)
unScale_time_lagged_data_7 <- unnormalize(predictions_7, output_min3, output_max3)
unScale_time_lagged_data_8 <- unnormalize(predictions_8, output_min4, output_max4)

unScale_time_lagged_data_9 <- unnormalize(predictions_9, output_min1, output_max1)
unScale_time_lagged_data_10 <- unnormalize(predictions_10, output_min2, output_max2)
unScale_time_lagged_data_11 <- unnormalize(predictions_11, output_min3, output_max3)
unScale_time_lagged_data_12 <- unnormalize(predictions_12, output_min4, output_max4)


unScale_time_lagged_data_2
original_test_output2


rmse_1 <- RMSE(original_test_output1, unScale_time_lagged_data_1)
rmse_2 <- RMSE(original_test_output2, unScale_time_lagged_data_2)
rmse_3 <- RMSE(original_test_output3, unScale_time_lagged_data_3)
rmse_4 <- RMSE(original_test_output4, unScale_time_lagged_data_4)
rmse_5 <- RMSE(original_test_output1, unScale_time_lagged_data_5)
rmse_6 <- RMSE(original_test_output2, unScale_time_lagged_data_6)
rmse_7 <- RMSE(original_test_output3, unScale_time_lagged_data_7)
rmse_8 <- RMSE(original_test_output4, unScale_time_lagged_data_8)
rmse_9 <- RMSE(original_test_output1, unScale_time_lagged_data_9)
rmse_10 <- RMSE(original_test_output2,unScale_time_lagged_data_10)
rmse_11<- RMSE(original_test_output3, unScale_time_lagged_data_11)
rmse_12 <- RMSE(original_test_output4,unScale_time_lagged_data_12)
```

```
mape_1 <- MAPE(original_test_output1, unScale_time_lagged_data_1)
mape_2 <- MAPE(original_test_output2, unScale_time_lagged_data_2)
mape_3 <- MAPE(original_test_output3, unScale_time_lagged_data_3)
mape_4 <- MAPE(original_test_output4, unScale_time_lagged_data_4)
mape_5 <- MAPE(original_test_output1, unScale_time_lagged_data_5)
mape_6 <- MAPE(original_test_output2, unScale_time_lagged_data_6)
mape_7 <- MAPE(original_test_output3, unScale_time_lagged_data_7)
mape_8 <- MAPE(original_test_output4, unScale_time_lagged_data_8)
mape_9 <- MAPE(original_test_output1, unScale_time_lagged_data_9)
mape_10 <- MAPE(original_test_output2,unScale_time_lagged_data_10)
mape_11 <- MAPE(original_test_output3,unScale_time_lagged_data_11)
mape_12 <- MAPE(original_test_output4,unScale_time_lagged_data_12)

mse_1 <- MSE(original_test_output1, unScale_time_lagged_data_1)
mse_2 <- MSE(original_test_output2, unScale_time_lagged_data_2)
mse_3 <- MSE(original_test_output3, unScale_time_lagged_data_3)
mse_4 <- MSE(original_test_output4, unScale_time_lagged_data_4)
mse_5 <- MSE(original_test_output1, unScale_time_lagged_data_5)
mse_6 <- MSE(original_test_output2, unScale_time_lagged_data_6)
mse_7 <- MSE(original_test_output3, unScale_time_lagged_data_7)
mse_8 <- MSE(original_test_output4, unScale_time_lagged_data_8)
mse_9 <- MSE(original_test_output1, unScale_time_lagged_data_9)
mse_10 <- MSE(original_test_output2, unScale_time_lagged_data_10)
mse_11<- MSE(original_test_output3, unScale_time_lagged_data_11)
mse_12 <- MSE(original_test_output4, unScale_time_lagged_data_12)



# Print error metrics
cat("Model 1:\n")
cat("RMSE: ", rmse_1, "\n")
cat("MAPE: ", mape_1, "\n")
cat("MSE: ", mse_1, "\n")
```

```
cat("Model 2:\n")
cat("RMSE: ", rmse_2, "\n")
cat("MAPE: ", mape_2, "\n")
cat("MSE: ", mse_2, "\n")

cat("Model 3:\n")
cat("RMSE: ", rmse_3, "\n")
cat("MAPE: ", mape_3, "\n")
cat("MSE: ", mse_3, "\n")

cat("Model 4:\n")
cat("RMSE: ", rmse_4, "\n")
cat("MAPE: ", mape_4, "\n")
cat("MSE: ", mse_4, "\n")

cat("Model 5:\n")
cat("RMSE: ", rmse_5, "\n")
cat("MAPE: ", mape_5, "\n")
cat("MSE: ", mse_5, "\n")

cat("Model 6:\n")
cat("RMSE: ", rmse_6, "\n")
cat("MAPE: ", mape_6, "\n")
cat("MSE: ", mse_6, "\n")

cat("Model 7:\n")
cat("RMSE: ", rmse_7, "\n")
cat("MAPE: ", mape_7, "\n")
cat("MSE: ", mse_7, "\n")

cat("Model 8:\n")
cat("RMSE: ", rmse_8, "\n")
cat("MAPE: ", mape_8, "\n")
cat("MSE: ", mse_8, "\n")

cat("Model 9:\n")
cat("RMSE: ", rmse_9, "\n")
```

```
cat("MAPE: ", mape_9, "\n")
cat("MSE: ", mse_9, "\n")

cat("Model 10:\n")
cat("RMSE: ", rmse_10, "\n")
cat("MAPE: ", mape_10, "\n")
cat("MSE: ", mse_10, "\n")

cat("Model 11:\n")
cat("RMSE: ", rmse_11, "\n")
cat("MAPE: ", mape_11, "\n")
cat("MSE: ", mse_11, "\n")

cat("Model 12:\n")
cat("RMSE: ", rmse_12, "\n")
cat("MAPE: ", mape_12, "\n")
cat("MSE: ", mse_12, "\n")



######################################################################



time_lagged_data_5 <- data.frame(t_7 = data$time19,
                                 t_6 = data$time18,
                                 t_5 = lag(data$time20, 5),
                                 t_4 = lag(data$time20, 4),
                                 t_3 = lag(data$time20, 3),
                                 t_2 = lag(data$time20, 2),
                                 t_1 = lag(data$time20, 1),
                                 output = data$time20)[-c(1:7), ]

time_lagged_data_6 <- data.frame(t_8 = data$time19,
                                 t_7 = data$time18,
                                 t_6 = lag(data$time20, 6),
                                 t_5 = lag(data$time20, 5),
```

```
                    t_4 = lag(data$time20, 4),
                    t_3 = lag(data$time20, 3),
                    t_2 = lag(data$time20, 2),
                    t_1 = lag(data$time20, 1),
                    output = data$time20)[-c(1:8), ]


time_lagged_data_7 <- data.frame(t_9 = data$time19,
                    t_8 = data$time18,
                    t_7 = lag(data$time20, 7),
                    t_6 = lag(data$time20, 6),
                    t_5 = lag(data$time20, 5),
                    t_4 = lag(data$time20, 4),
                    t_3 = lag(data$time20, 3),
                    t_2 = lag(data$time20, 2),
                    t_1 = lag(data$time20, 1),
                    output = data$time20)[-c(1:9), ]



Scale_time_lagged_data_5 <- as.data.frame(lapply(time_lagged_data_5[1:8], normalize ))
Scale_time_lagged_data_6 <- as.data.frame(lapply(time_lagged_data_6[1:9], normalize ))
Scale_time_lagged_data_7 <- as.data.frame(lapply(time_lagged_data_7[1:10], normalize ))

Scale_time_lagged_data_5
Scale_time_lagged_data_6
Scale_time_lagged_data_7


train_norm_t_5 <- Scale_time_lagged_data_5 [2:380,]
test_norm_t_5 <- Scale_time_lagged_data_5 [381:463,]
test_norm_t_5
train_norm_t_6 <- Scale_time_lagged_data_6 [2:380,]
test_norm_t_6 <- Scale_time_lagged_data_6 [381:462,]
test_norm_t_6
train_norm_t_7 <- Scale_time_lagged_data_7 [2:380,]
test_norm_t_7 <- Scale_time_lagged_data_7 [381:461,]
```

```
new_model_1 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5+t_6+t_7, data =
Scale_time_lagged_data_5, hidden = c(2))
new_model_2 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5+t_6+t_7, data =
Scale_time_lagged_data_5, hidden = c(5,3))
new_model_3 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5 + t_6+t_7+t_8, data =
Scale_time_lagged_data_6, hidden = c(8,6))
new_model_4 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5 + t_6+t_7+t_8, data =
Scale_time_lagged_data_6, hidden = c(6,4))
new_model_5 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5 + t_6 + t_7+t_8+t_9, data =
Scale_time_lagged_data_7, hidden = c(9))
new_model_6 <- neuralnet(output ~ t_1 + t_2 + t_3 + t_4+ t_5 + t_6 + t_7+t_8+t_9, data =
Scale_time_lagged_data_7, hidden = c(5,10))


plot(new_model_1)
plot(new_model_2)
plot(new_model_3)
plot(new_model_4)
plot(new_model_5)
plot(new_model_6)


new_predictions_1 <- predict(new_model_1, Scale_time_lagged_data_5[, 1:8])
new_predictions_2 <- predict(new_model_2, Scale_time_lagged_data_5[, 1:8])
new_predictions_3 <- predict(new_model_3, Scale_time_lagged_data_6[, 1:9])
new_predictions_4<- predict(new_model_4, Scale_time_lagged_data_6[, 1:9])
new_predictions_5<- predict(new_model_5, Scale_time_lagged_data_7[, 1:10])
new_predictions_6<- predict(new_model_6, Scale_time_lagged_data_7[, 1:10])


original_train_output5 <- Scale_time_lagged_data_5[1:380,"output"]
original_test_output5 <- Scale_time_lagged_data_5[381:463,"output"]

original_train_output6 <- Scale_time_lagged_data_6[1:380,"output"]
```

```
original_test_output6 <- Scale_time_lagged_data_6[381:462,"output"]

original_train_output7 <- Scale_time_lagged_data_7[1:380,"output"]
original_test_output7 <- Scale_time_lagged_data_7[381:461,"output"]

output_min5 <- min(original_train_output5)
output_max5 <- max(original_train_output5)
output_min6 <- min(original_train_output6)
output_max6 <- max(original_train_output6)
output_min7 <- min(original_train_output7)
output_max7 <- max(original_train_output7)

unScale_time_lagged_data_1 <- unnormalize(new_predictions_1, output_min5, output_max5)
unScale_time_lagged_data_2 <-unnormalize(new_predictions_2, output_min6, output_max6)
unScale_time_lagged_data_3 <- unnormalize(new_predictions_3, output_min7, output_max7)
unScale_time_lagged_data_4 <- unnormalize(new_predictions_4, output_min5, output_max5)
unScale_time_lagged_data_5 <-unnormalize(new_predictions_5, output_min6, output_max6)
unScale_time_lagged_data_6 <- unnormalize(new_predictions_6, output_min7, output_max7)


new_rmse_1 <- RMSE(original_test_output5, unScale_time_lagged_data_1)
new_rmse_2 <- RMSE(original_test_output6, unScale_time_lagged_data_2)
new_rmse_3 <- RMSE(original_test_output7, unScale_time_lagged_data_3)
new_rmse_4 <- RMSE(original_test_output5, unScale_time_lagged_data_4)
new_rmse_5 <- RMSE(original_test_output6, unScale_time_lagged_data_5)
new_rmse_6 <- RMSE(original_test_output7, unScale_time_lagged_data_6)


new_mape_1 <- MAPE(original_test_output5, unScale_time_lagged_data_1)
new_mape_2 <- MAPE(original_test_output6, unScale_time_lagged_data_2)
new_mape_3 <- MAPE(original_test_output7, unScale_time_lagged_data_3)
new_mape_4 <- MAPE(original_test_output5, unScale_time_lagged_data_4)
new_mape_5 <- MAPE(original_test_output6, unScale_time_lagged_data_5)
new_mape_6 <- MAPE(original_test_output7, unScale_time_lagged_data_6)

new_mse_1 <- MSE(original_test_output5, unScale_time_lagged_data_1)
new_mse_2 <- MSE(original_test_output6, unScale_time_lagged_data_2)
new_mse_3 <- MSE(original_test_output7, unScale_time_lagged_data_3)
```

```
new_mse_4 <- MSE(original_test_output5, unScale_time_lagged_data_4)
new_mse_5 <- MSE(original_test_output6, unScale_time_lagged_data_5)
new_mse_6 <- MSE(original_test_output7, unScale_time_lagged_data_6)


# Print error metrics
cat("new Model 1:\n")
cat("RMSE: ", new_rmse_1, "\n")
cat("MAPE: ", new_mape_1, "\n")
cat("MSE: ", new_mse_1, "\n")


cat("new Model 2:\n")
cat("RMSE: ", new_rmse_2, "\n")
cat("MAPE: ", new_mape_2, "\n")
cat("MSE: ", new_mse_2, "\n")

cat("new Model 3:\n")
cat("RMSE: ", new_rmse_3, "\n")
cat("MAPE: ", new_mape_3, "\n")
cat("MSE: ", new_mse_3, "\n")

cat("new Model 4:\n")
cat("RMSE: ", new_rmse_4, "\n")
cat("MAPE: ", new_mape_4, "\n")
cat("MSE: ", new_mse_4, "\n")

cat("new Model 5:\n")
cat("RMSE: ", new_rmse_5, "\n")
cat("MAPE: ", new_mape_5, "\n")
cat("MSE: ", new_mse_5, "\n")

cat("new Model 6:\n")
cat("RMSE: ", new_rmse_6, "\n")
cat("MAPE: ", new_mape_6, "\n")
cat("MSE: ", new_mse_6, "\n")

plot(original_test_output7 , ylab = "Predicted vs Expected", type="l", col="red" )
```

```
par(new=TRUE)
plot(unScale_time_lagged_data_6, ylab = " ", yaxt="n", type="l", col="green" ,main='Predicted Value
Vs Expected Value Graph')
legend("topleft",
      c("Expected","Predicted"),
      fill=c("red","green")
)
#####################################################################
```

.