INFORMATICS INSTITUTE OF TECHNOLOGY

In collaboration with

UNIVERSITY OF WESTMINSTER

5DATA001C.2 Machine Learning and Data Mining

Module Leader - Mr Nipuna

Name: Mohammed Hashim Kalam

UoW no: w1957407

Student no: 20211291

Table of Contents

Contents

Abbreviations

Short Form	Long Form
PCA	Partition Clustering Analysis
NN	Neural Network
MLP	Multi-Layer Perceptron
KM	K-Means (Clustering)
WSS	Within-Cluster Sum of Squares
BSS	Between-Cluster Sum of Squares
CH	Calinski-Harabasz Index
AR	Autoregressive Model
MA	Moving Average
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
sMAPE	Symmetric Mean Absolute Percentage Error
IO	Input / Output

Partition Clustering

1st Subtask – Before PCA

Z-Score normalization

For preprocessing the given dataset, it was normalized using the **scale()** function in R for the z-score normalization. The reason for this is to ensure that all the features contribute equally to the clustering process – which hence improves the performance of the algorithm. The reason for Z-score is that it could **help in identifying the outliers** and possibly remove them to give an even better output.

dfNormZ <- as.data.frame(scale(Whitewine_v6)) # scaling the data

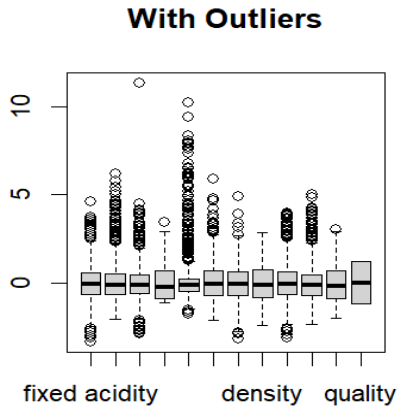
Removal of Outliers

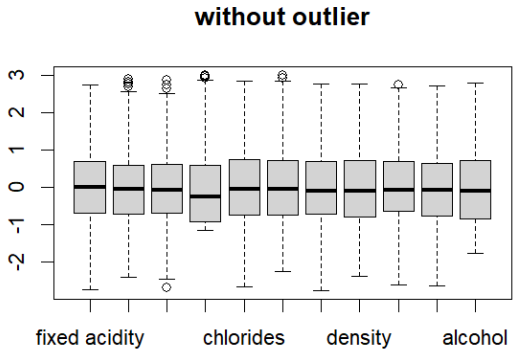
Not all the datasets are perfect – there might be few data that goes too away from the other data. To ensure consistency of the dataset and improve accuracy – removal of outliers was performed.

Outlier detection was done using dfNormZ – the z-score normalized data. After inspection, it was decided to use the Tukey's method which is a common method when it comes to removing outliers! It determines the outliers based on the interquartile range (IQR). Interquartile range basically tells you of how disperse the dataset is in the middle half of the distribution. We consider the first and the third quartile range – gets the lowest 25% of the data and the second range is anything above the 75% range.

To do this in R -> using the *quatile()* method – to get the first and third quartile. We use that to calculate the lower and upper bounds. For lower -> *qnt[1] - 1.5 * iqr* and the upper bounds -> *qnt[2] + 1.5 * iqr.* Now whatever data points lies before the lower bound and after the upper bound are considered as outliers and are removed from the dataset.

The following box-plot graphs are to show the comparison of what it was with outliers and how it is now after removing the outliers – almost cleaned dataset.





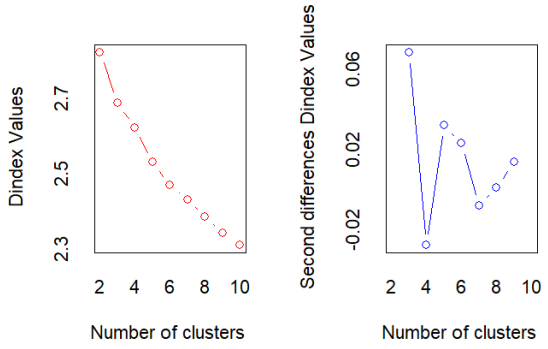
Now with this cleaned dataset - improves the performance and accuracy of the final output – clustering! Although it is not 100% cleaned as you could inspect a few outliers still available – but just with a few would not have a significant impact so it is all good!

```
# Using Tukey's method to identify outliers

# Function to detect outliers using Tukey's method
detect_outliers_tukey <- function(x) {
  qnt <- quantile(x, probs=c(0.25, 0.75), na.rm = TRUE)
  iqr <- IQR(x, na.rm = TRUE)
  fence_low <- qnt[1] - 1.5 * iqr
  fence_high <- qnt[2] + 1.5 * iqr
  outliers <- which(x < fence_low | x > fence_high)
  return(outliers)
}

# Applying Tukey's method to detect outliers for each variable
outliers_tukey <- apply(dfNormZ, 2, detect_outliers_tukey)

# Combining outliers detected by Tukey's method for all variables
all_outliers <- unique(unlist(outliers_tukey))

# Removing outliers from the dataset
dfNormZ_no_outliers <- dfNormZ[-all_outliers, ]
```

Determining K value for clusters

1. NB Clust analysis

```
# nb clust
 set.seed(10)
 nb <- NbClust(dfNormZ_no_outliers, distance = "euclidean", min.nc = 2, max.nc = 10, method
= "kmeans",     index="all")
```

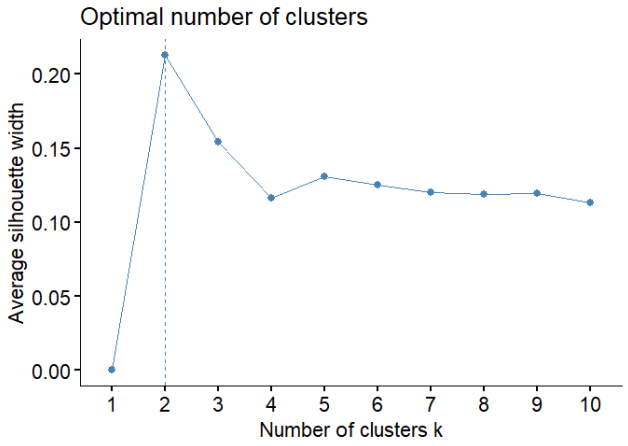



```
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 10 proposed 3 as the best number of clusters
* 2 proposed 6 as the best number of clusters
* 1 proposed 10 as the best number of clusters

                   ***** Conclusion *****

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

Over here in the NB Clust – according to the majority tule -> 2 clusters has been selected. 11 are proposed as 2 best number of clusters where with only 1 less proposed which is 3 best number of clusters coming in as second-best number of clusters!

2. Silhouette

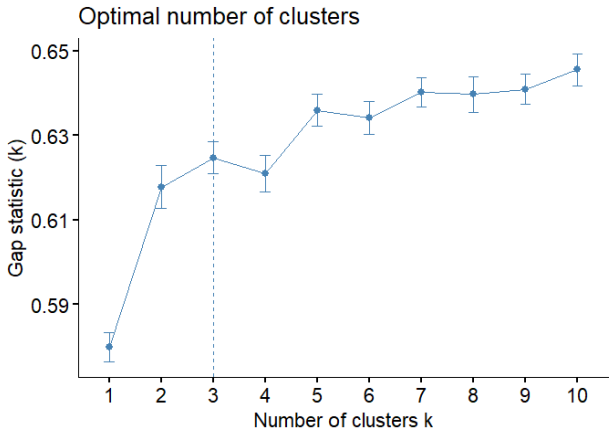
In the silhouette method – it basically determines the clusters by using the silhouette scores. The scores are basically a measure of how similar that specific data is to its own cluster than with the other clusters present.



The above diagram gives us a visual representation of how many clusters could be formed with the scaled dataset. It shows a maximum of 2 clusters while 3 clusters come in second with regards of the optimal number of clusters!

3. Gap Stats

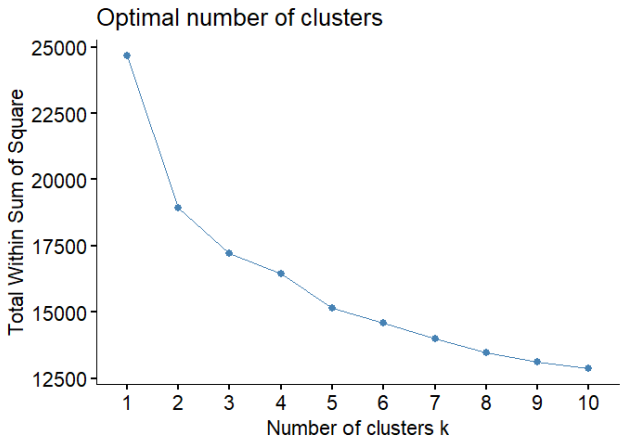
The Gap stats method basically compares the sum of intra-cluster variations of various k values with their expected values under null reference of the data. [citation]



In the gap stats method, we again see clearly a see sudden decrease at point 3 of the number of clusters. This sudden bend in the plot above means that the clustering process becomes less useful/ informative after that specific point. Due to this we could come to conclusion that in the gap stats method – the best number of clusters is 3.

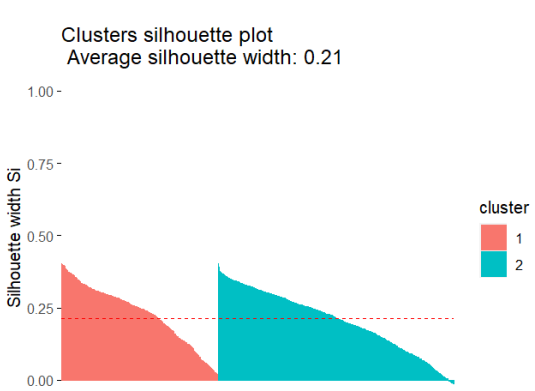
4. Elbow Method

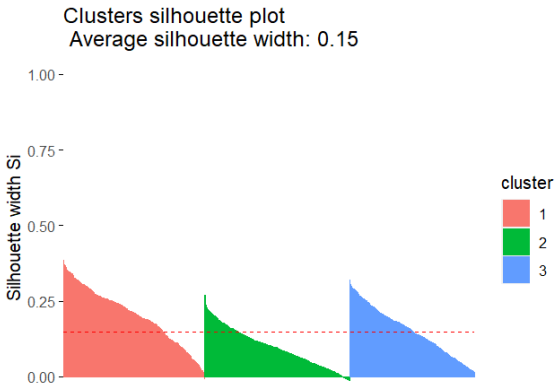
In the elbow method – it basically way to graphically represent the approx. optimal number of clusters in a dataset. It displays the within-cluster-sum-of-square values on the y axis corresponding to various values for k (number of cluster) on the x axis as shown in the below attached plot. The optimal value of clusters (k) is determined by an elbow representation in the plot.



In the above elbow graph – we could clearly see that until the 3rd cluster the lines become steeper meaning that adding more clusters improves the quality of the clustering! But after the 3rd cluster – from the 4th cluster it gets less steep! So, based on the elbow method, we could say in the above elbow diagram – that the optimal number of clusters is 3.

Since it's a tie between the above 4 automated tools -> below explanation has been considered to finalize the cluster k value.





The silhouette plot average with of cluster 2 (width-0.21) is greater than the average plot of cluster 3 (width-0.15) -> so due to this reason – as it was a tie in the auto-mated tools – considering that the greater the width, the better the algorithm has performed clustering -> cluster with the k value 2 wins and is decided to be taken forward with.

K-Means Clustering before PCA

Since now we could come to conclusion that the k value could be taken as 2 – we could move on to implementing the k means clustering. The k means clustering is done using the inbuilt function *kmeans()* with the necessary variables passed in. Move on to getting the clusters of it and finally plotting the graph.

```
### K MEANS CLUSTERING ###
# ASSINGING K VALUE
k_value = 2

# PASSING AND GETTING THE K MEANS
km <- kmeans(dfNormZ_no_outliers, centers= k_value, nstart= 20)
km
```

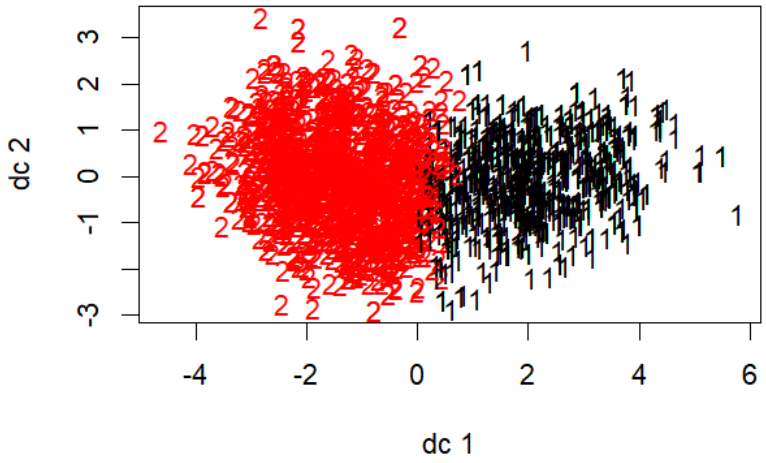
 # CALCULATING CENTROIDS

centroids <- km$centers

 # PLOTTING

library(fpc)

plotcluster(dfNormZ_no_outliers, km$cluster)



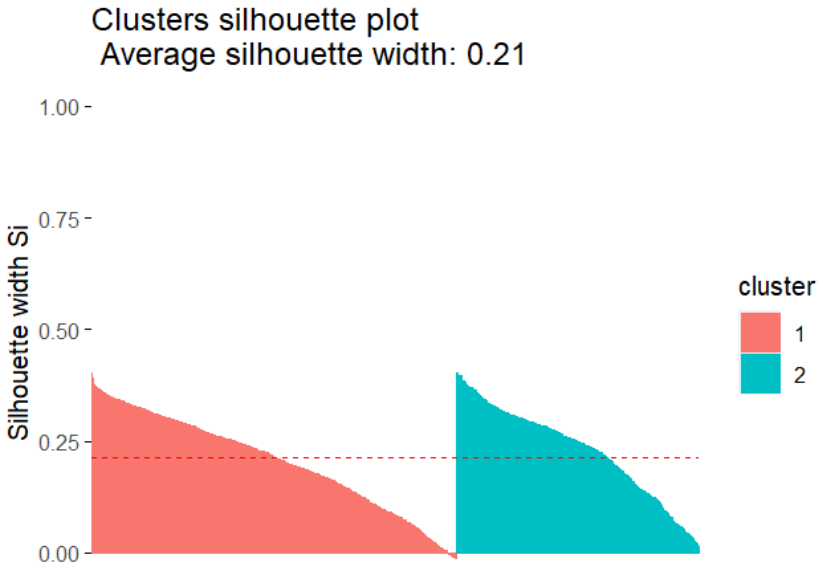
Silhouette Plot for the Width Between Clusters Before PCA

WSS -> the **within sum of squares** -> gets the total error of the cluster (error meaning the distance between each data point to the cluster centroid) and sum up the **squared** distance/error within the datapoints of the cluster.

BSS -> the **between cluster sum of squares** -> gets again the squared but this time of the distance between the cluster centroids and the overall centroid of the entire dataset!

Silhouette plot is used to measure the **quality of the clustering process** during k means. It helps to determine how well a **data point fits into its own cluster rather than the other existing clusters** within the dataset.

With these values calculated during the clustering k means process -> using *silhouette()* function to get the width for each observation according to the available clusters and the distance between the clusters. Below is the diagram of the silhouette plot using the *fviz_silhouette()* function.



Silhouette width is a measure of how similar the datapoints in the dataset to its own cluster compared to other clusters within the dataset. With the above attached image – we could notice that it has a standard average width of 0.21. A width with a higher positive value is a better value whereas a negative width value could indicate an error occurred.

2nd Subtask – PCA

PCA – also known as Principal Component Analysis – is basically a dimensionality reduction of the features to give a similar output by reducing the number of features but also retaining most of the data. This basically helps to use a smaller version of a bigger dataset and gives similar results by maintaining significant patterns. This way it will help in the computational power needed – time taken to do the clustering process but also get similar results with a much time consuming and higher requirement of computational power. [Jaadi, 2024]

In the coursework – to accomplish this – we got to firstly create the covalence matrix along by calculating the eigenvectors and eigenvalues. With this data – we then calculate the cumulative score per principal component and whatever pcs that got at least score greater than 85%. Below is the code for the above-mentioned steps.

```
# calc eigenvalues & eigenvectors
pca <- dfNormZ_no_outliers
wine.cov <- cov(pca)
wine.eigen <- eigen(wine.cov)
str(wine.cov)

phi <- wine.eigen$vectors
phi

# calc scores for all pc -> principal components
PC <- as.matrix(dfNormZ_no_outliers) %*% phi
PC

# calc scores per PC
eigenVal <- wine.eigen$values
cumulative_scores <- cumsum(eigenVal) / sum(eigenVal)
cumulative_scores
```

select_cumula_scores <- sum(cumulative_scores <= 0.85) +1

select_cumula_scores

With the above code – the selected cumulative score is 7. Meaning that out of all the features in the dataset which is 11, depending on the requirement of the scores must be at least 85% - we came down to 7 features.

Determining K value for clusters

Four "automated" tools to this new pca-based dataset.

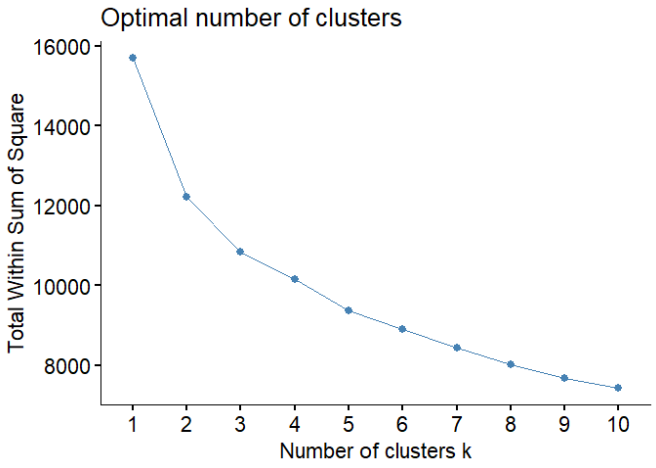
1. **NB Clust.**

```
> set.seed(10)
> nb <- NbClust(transformed, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans", index="all")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

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

*******************************************************************
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 5 proposed 3 as the best number of clusters
* 2 proposed 5 as the best number of clusters
* 1 proposed 6 as the best number of clusters
* 4 proposed 7 as the best number of clusters
* 1 proposed 10 as the best number of clusters

                   ***** Conclusion *****

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


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

 # nb clust
set.seed(10)
nb <- NbClust(transformed, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans", index="all")

Over here again in the NB Clust – according to the majority rule -> 2 clusters has been selected. 11 are proposed as 2 best number of clusters where with for 5 proposed as 3 best number of clusters coming in as second-best number of clusters!

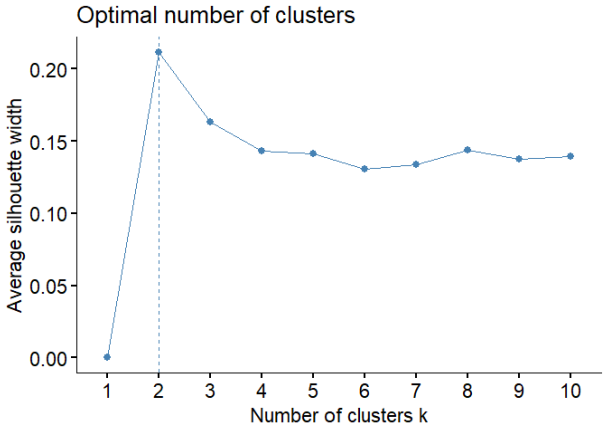
2. **Elbow method.**




```
# elbow
k = 2:10
set.seed(42)
WSS = sapply(k, function(k) {kmeans(transformed, centers=k)$tot.withinss})
plot(k, WSS, type="b", xlab= "Number of k", ylab="Within sum of squares")
```

In the above elbow graph – we could clearly see that until the 3rd cluster the lines become steeper meaning that adding more clusters improves the quality of the clustering! But after the 3rd cluster – from the 4th cluster it gets less steep! So, based on the elbow method, we could say in the above elbow diagram – that the optimal number of clusters is 3.

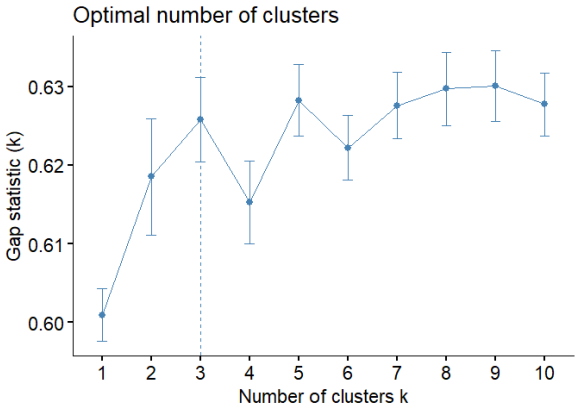
3. **Silhouette method.**



```
#silhouette
set.seed(12)
silh_nb <- fviz_nbclust(transformed, kmeans, method = "silhouette")
print(silh_nb)
```

In the above plot, we clearly see that there is a sudden bend at point 2 of the number of clusters! This shows that the clusters become less meaningful beyond 2 clusters – so due to this, we could consider 2 as the best number of clusters with the silhouette method.

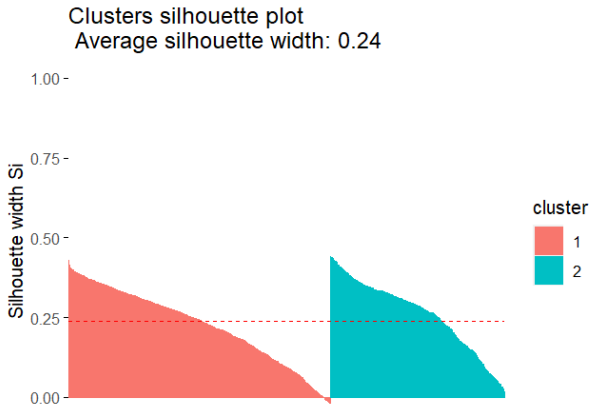
4. Gap Stats method

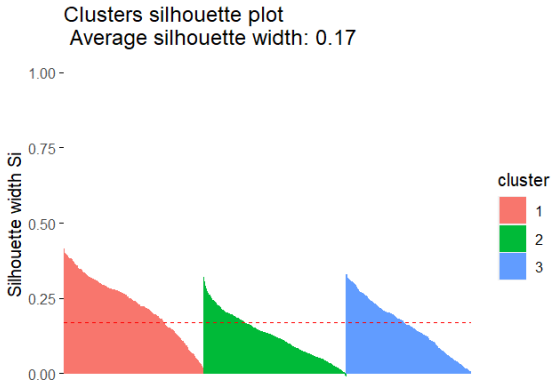



```
# gap_stat
set.seed(14)
gap_nb <- fviz_nbclust(transformed, kmeans, method = "gap_stat")
print(gap_nb)
```
I

In the gap stats method, we again see clearly a see sudden decrease at point 3 of the number of clusters. This sudden bend in the plot above means that the clustering process becomes less useful/ informative after that specific point. Due to this we could come to conclusion that in the gap stats method – the best number of clusters is 3.

Again, like before pca -> even after pca it's a tie between the above 4 automated tools -> below explanation has been considered to finalize the cluster k value.



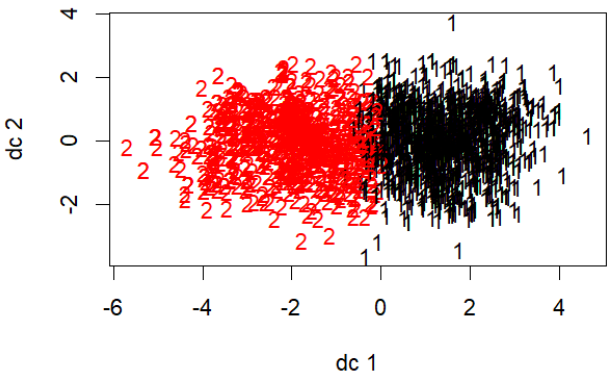


The silhouette plot average with of cluster 2 (width-0.24) is greater than the average plot of cluster 3(wdith-0.17) -> so due to this reason – as it was a tie in the auto-mated tools – considering that the greater the width, the better the algorithm has performed clustering -> cluster with the k value 2 wins and is decided to be taken forward with.

K-Means Clustering using PCA Processed Data

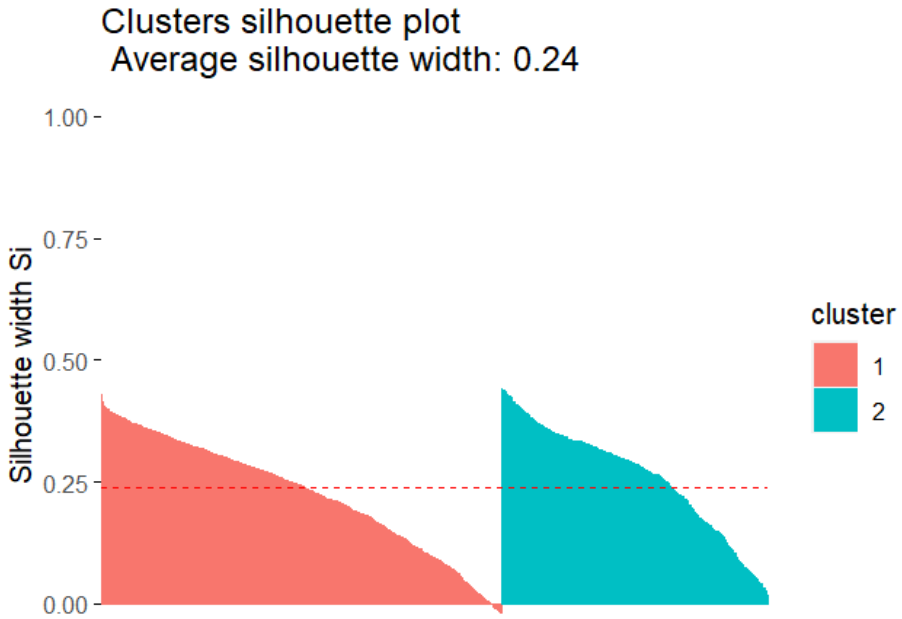
Since now we could come to conclusion that the k value could be taken as 2 – we could move on to implementing the k means clustering. The k means clustering is done using the inbuilt function *kmeans()* with the necessary variables passed in. Move on to getting the clusters of it and finally plotting the graph.

```
# k means cluster
k_value = 2

pca_km <- kmeans(transformed, centers= k_value, nstart= 20)
pca_km

# calc centroids
pca_km$centers
fviz_cluster(pca_km,data=transformed)
```



Silhouette Plot for the Width Between Clusters

Below is the diagram of the silhouette plot using the *fviz_silhouette()* function.



With the above attached image – we could notice that it has a standard average width of 0.24. The average silhouette width has increased after performing pca -> meaning that it has now clustered even better before performing pca which is a good sign. The higher the average width -> a better gap between clusters (BSS) -> meaning clustering performance has increased.

Calinski Harabasz Index Calculation

Calinski-Harabasz (CH) Index is a metrics utilized to evaluate the performance of clustering models. Meanwhile, the CH Index (also known as Variance ratio criterion) is a ratio of how of basically BSS and WSS. The ratio of how well the clusters points is separate between clusters and within clusters. Therefore, this index could be used when the data has partitioned in a clustering algorithm that has no external validation like in silhouette plots. [Unknown Wrtier, 2024]

```
> # CALINSKI HARABASZ INDEX CALCULATION
> library(fpc)
>
> calin_Harab_index <- cluster.stats(transformed, pca_km$cluster)$ch
> print(calin_Harab_index)
[1] 639.0902
```

The above screenshot – result shows that the CH index for the clustering after performing PCA is 639.0902 -> meaning that the ratio of BSS to WSS is high. This is a good sign so now it clearly shows that the clustering process has been performed exceptionally do the better separation value – width between clusters. Overall, a good high CH value is expected for a well clustered dataset!

Financial Forecasting Part

Input variables used in MLP models for exchanges rates forecasting.

a) Exchange rate forecasting is crucial especially for businesses so that they can tract and make decisions in the global market. For predicting the future exchange rates – MLP models could play a powerful tool for doing it as they could be used to identify and learn complex patterns in historical data. In MLP models, input vectors play a huge role in the accuracy in predicting the rates and for exchange rate forecasting – a few could be considered as crucial factors.

- **AR Approach** – Autoregressive (AR) approach is when it considers time-delayed values of the exchange rate as the input variables. Basically, in AR – it understands the pattern of the past exchange rates – learning from the data over time.
- **Exogenous variables** – along with the lagged values as mentioned above (past exchange rates) – it is also important to consider the variable that could affect the exchange rates. Variables like inflation rates, interest rates and more could potentially have an impact on the flow of the exchange rates pattern over time.
- **MA Approach** – Moving Average (MA) approach is like AR but not the same. In MA approach, it does not consider the past values but the past forecast errors to predict the current value. By considering the previous inaccuracies in terms of predictions – the model adjusts and improves the prediction accuracy with the past forecast errors.
- **Regularization Techniques** – when training models – it is vital to check whether it does not overfit and even underfit but train properly for a accurate output for similar but unseen data! Regularization could help prevent the overfitting – which will eventually improve the model's capabilities to generalize against similar but unseen data – that it is very important in forecasting models – as accuracy is key especially when dealing with currency ratings!

IO Matrix

To build the IO (input / output) matrices -> we **firstly build the input vectors**. In the coursework, it is mentioned to use the last column of the data set as the input variables and up to t-4 is recommended. So, using the lag() function -> the **lagged values of the first 4 of the 3rd column** is taken as input vectors!

lagged_output_1 <- lag(output_col_og_dataset, 1)

lagged_output_2 <- lag(output_col_og_dataset, 2)

lagged_output_3 <- lag(output_col_og_dataset, 3)

lagged_output_4 <- lag(output_col_og_dataset, 4)

Using the lagged values – we **bind all** the values together along **with the actual data** of the 3rd column of dataset to **one IO matrices**. By mixing up the lagged values -> four different IO matrices are built to be used in the latter process of the code!

IO_matrix_1 <- cbind(lagged_output_1, lagged_output_2, lagged_output_3, lagged_output_4, output_col_og_dataset)

IO_matrix_2 <- cbind(lagged_output_2, lagged_output_1, lagged_output_3, lagged_output_4, output_col_og_dataset)

IO_matrix_3 <- cbind(lagged_output_4, lagged_output_3, lagged_output_2, lagged_output_1, output_col_og_dataset)

IO_matrix_4 <- cbind(lagged_output_3, lagged_output_4, lagged_output_2, lagged_output_1, output_col_og_dataset)

Why normalize data before using in MLP

As in MLP – it is all about learning patterns and understanding how a set of inputs could correspond to an output. Normalizing the data at the start before passing it to the MLP model for training will generally help in speeding up the learning process which will lead to a fast and a more efficient process. It will also eventually improve the performance of the network as it will lead to an increase in the accuracy of the output. Not only accuracy, but it will help in stabilizing the training process. How stabilize? Basically, before normalization – the data points in the dataset could vary in scale – making the model harder to understand patterns. But after normalizing – it is then basically brought up to a certain low scale which now the model can easily learn the patterns in a faster and efficient way.

Following the creation of IO matrix -> firstly before normalizing with min max function -> all the NA rows are removed as it would become an issue when doing min max on a matrices that contains NA values.

```
IO_matrix_1 <- IO_matrix_1[complete.cases(IO_matrix_1),]

IO_matrix_2 <- IO_matrix_2[complete.cases(IO_matrix_2),]

IO_matrix_3 <- IO_matrix_3[complete.cases(IO_matrix_3),]

IO_matrix_4 <- IO_matrix_4[complete.cases(IO_matrix_4),]
```

After successfully removing all the NA value rows as required -> normalize (min max) has been applied to all the four matrices.

```
IO_matrix_1_normz <- normalize(IO_matrix_1)

IO_matrix_2_normz <- normalize(IO_matrix_2)

IO_matrix_3_normz <- normalize(IO_matrix_3)

IO_matrix_4_normz <- normalize(IO_matrix_4)
```

Now with the processed IO matrix -> 14 different models have been created by changing the hidden state nodes/ layers, activation function used and the linear or nonlinear output!

Understanding of the four stat. indices

1. **RMSE**

RMSE, short form for – Root Mean Square Error – is a widely utilized metrics for forecasting models like the one above. It is calculated by getting the square root of the average of squared error – difference – between the prediction and the actual value. So, the lower the error rate value – the better the model predicts the output!

2. **MAE**

MAE is another performance metric used to measure the accuracy of a model. It measures the average size/magnitude of the errors got in a sample of predictions and its observations. It is one of the most common loss functions for regression problems. [Unknown Writer, 2024] So, the lower the error rate value – the better the model predicts the output!

3. **MAPE**

MAPE is one of the many performance metrics utilized to get to know/measure how good a model is in making predictions. Basically, a measure of the model's accuracy. How it does it – it measures the average percentage of difference/errors made by the model tested with. In simple words, how inaccurate the predictions are on average. So, MAPE of 5% means – the average absolute percentage difference between the predicted values tested to its actual values is 5%. [Roberts, 2023] So, the lower the error rate value – the better the model predicts the output!

4. **SMAPE (symmetric MAPE)**

With the name itself – sMAPE is like MAPE but not! It rather calculates the mean percentage error/difference symmetrically around zero. It is widely used in predicting and forecasting models

– models with both positive and negative values in the dataset samples. So, the lower the error rate value – the better the model predicts the output!

Comparison Table of Testing Performances

```
> # PRINTING THE DATA FRAME FOR A TABLE FORMAT - EASY TO DO COMPAR
> error_stats_df
    MLP_MODEL         RMSE          MAE         MAPE        SMAPE
1   model - 1 0.006095409 0.004521968 0.003421270 0.003423940
2   model - 2 0.006110399 0.004517073 0.003417456 0.003419622
3   model - 3 0.006165682 0.004669948 0.003531184 0.003534666
4   model - 4 0.006245512 0.004677452 0.003537087 0.003539669
5   model - 5 0.006191635 0.004659528 0.003522638 0.003526435
6   model - 6 0.006290856 0.004858592 0.003671178 0.003675920
7   model - 7 0.006181300 0.004733746 0.003578312 0.003581876
8   model - 8 0.006202858 0.004726766 0.003571719 0.003575773
9   model - 9 0.006156254 0.004587161 0.003470686 0.003473613
10 model - 10 0.006217388 0.004788007 0.003618653 0.003623087
11 model - 11 0.006083447 0.004512063 0.003414040 0.003416356
12 model - 12 0.006222788 0.004717673 0.003566824 0.003570934
13 model - 13 0.006178757 0.004704432 0.003556723 0.003560144
14 model - 14 0.006389065 0.004947045 0.003737845 0.003742729
>
```

The above comparison table is a collection of all the error stat indices of all the models separately! Below is the table of how the models were built.

Model	**Inputs**	**Hidden state layer**	**Activation function**
mlp_model1	lagged1, lagged2, lagged3, lagged4	1 – c(5)	no function
mlp_model2	lagged1, lagged2, lagged3, lagged4	2 – c(4, 6)	no function
mlp_model3	lagged1, lagged3, lagged4, lagged2	2 – c(6, 8)	no function
mlp_model4	lagged1, lagged3, lagged2, lagged4	3 – c(4, 6, 8)	logistic
mlp_model5	lagged1, lagged4, lagged3, lagged2	2 – c(8, 6)	tanh
mlp_model6	lagged1, lagged4, lagged3, lagged2	2 – c(6, 8)	logistic
mlp_model7	lagged2, lagged1, lagged3", lagged4	3 – c(4, 8, 4)	logistic
mlp_model8	lagged2, lagged3, lagged4, lagged1	2 – c(6, 4)	tanh
mlp_model9	lagged2, lagged4, lagged1, lagged3	3 – c(4, 6, 8)	tanh
mlp_model10	lagged4, lagged3, lagged2, lagged1	4 – c(6, 6, 9)	tanh

mlp_model11	lagged4, lagged1, lagged3, lagged2	3 – c(6, 4, 8)	logistic
mlp_model12	Lagged3, lagged4, lagged2, lagged1	3 – c(8)	tanh
mlp_model13	lagged3, lagged1, lagged4, lagged2	2 – c(8, 4, 6)	no function
mlp_model14	lagged3, lagged2, lagged1, lagged4	3 – c(10, 6, 8)	tanh

Above is the table of all the MLP models created along with the inputs – hidden state layer and node count and the activation function used. Let's consider the MVP model 1 and MVP model 11 as we could take both of them as the best and second best as both them have the lower values compared to other models.

Parameter counts for the hidden state.

MVP model1 -> 1 layer with 5 nodes -> 5params.

MVP model11 -> 3 layers with 6 to 4 to 8 nodes ratio -> 24+ 32 -> 56 params.

```
> calc_error_values_helper(predicted_values_c
$RMSE
[1] 0.006095409

$MAE
[1] 0.004521968

$MAPE
[1] 0.00342127

$SMAPE
[1] 0.00342394
```

```
> calc_error_values_helper(predicted_values_de
$RMSE
[1] 0.006083447

$MAE
[1] 0.004512063

$MAPE
[1] 0.00341404

$SMAPE
[1] 0.003416356
```

Model1 got 5 number of params in hidden state while model11 got 56. Both the models have lower error values compared to others but obviously since model11 has more params than model1 – eventually making it learn better and have a lower loss rate lower than that of model1 as in average.

0.00436564675 - model 1

0.0043564765 - model 11

Meaning model11 is the better model as now it is prone to error less likely than compared to model1. **The image on the left hand is for model1 and the image at the right hand is for model11.**

Model11 is taken as the best model due to having the lowest average error values compared to all the other models trained!

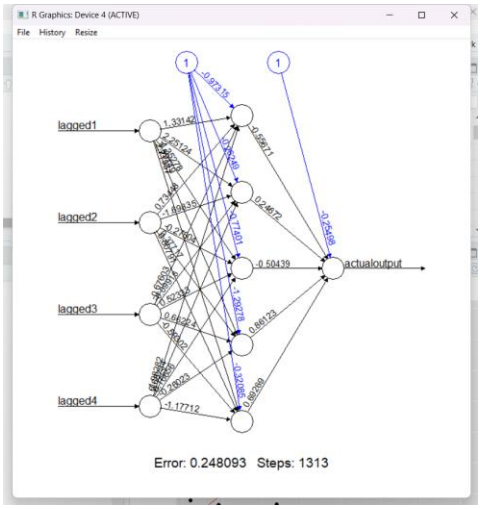
Efficiency Explanation Using One and Two Hidden Layer MLP Models

** I did not consider the activation function when calculating the parameters (cazt too much calculation) – just the parameters within the hidden state **

One hidden layer MLP models

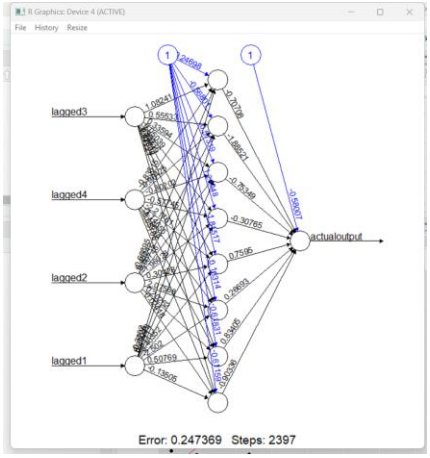
Calculating Number of Params

Model1



`5 nodes in node layer -> 5 params in nodes + bias as 1 -> so total 10 params`

Model12



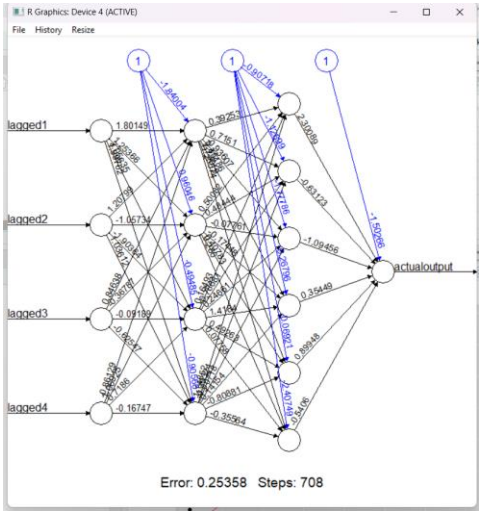
`8 nodes in node layer -> 8 params in nodes + bias as 1 -> so total 16 params`

For the MLP model with one hidden layer -> **model1 is more efficient** than model12 as in model1 it has less params than model12.

Two hidden layer MLP models

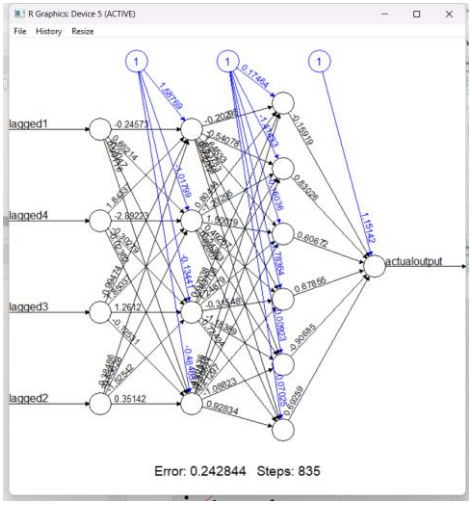
Calculating Number of Params

Model2



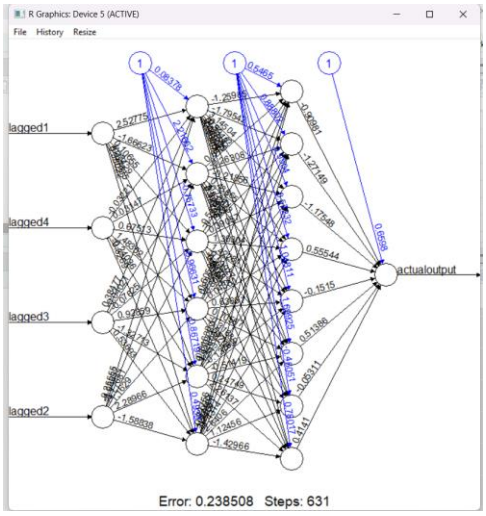

```
2 node layer -> 4 to 6 ratio -> so 24 params in node layer + bias as 1 -> 24 +
4 + 6 -> so total 34 params
```

Model5



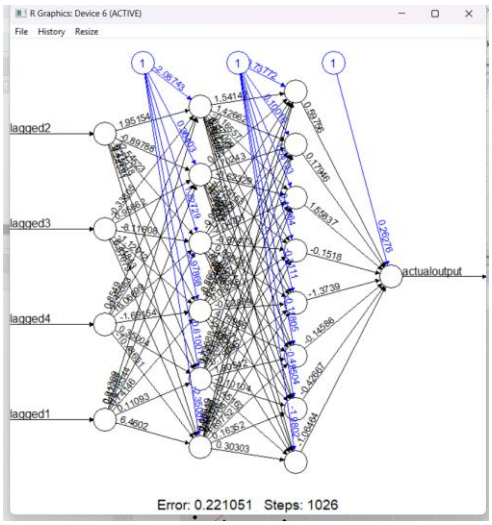

```
2 node layer -> 4 to 6 ratio -> so 24 params in node layer + bias as 1 -> 24 +
4 + 6 -> so total 34 params
```

Model6




```
2 node layer -> 6 to 8 ratio -> so 48 params in node layer + bias as 1 -> 48 +
6 + 8 -> so total 60 params
```

Model8



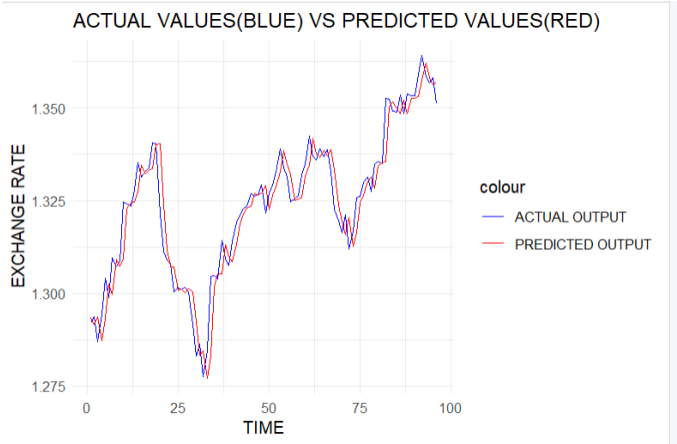

```
2 node layer -> 6 to 8 ratio -> so 48 params in node layer + bias as 1 -> 48 +
6 + 8 -> so total 60 params
```

For the MLP model with two hidden layers -> **model2 and model3 is more efficient** than model6 and model8 as they have lesser parameters.

Talking about efficiency of the models. The lower the parameters within the model – the better the higher the efficiency. In one hidden layer -> model1 has 10 params and model12 had 16 params – and in two hidden layer -> model2 and model5 has 34 params while model6 and model8 has 60

params -> meaning that the models will low count of params works efficiently. The lower the param count -> the simpler the model's structure. Meaning that only less time and computational power is required to run the model properly. Models will less params like model1 for one hidden layer and both model2 and model5 in two hidden layer works with higher efficiency and even performance as it will faster too. Also, models will lower parameters generalize to unseen data better!

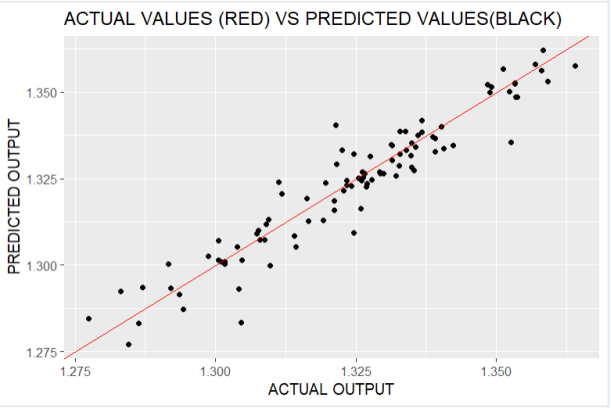
Time Series Plot for Predicted and Actual values



Model11 is taken as the best model due to having the lowest average error values compared to all the other models trained!

Above are the plots for both the predicted and actual values in one graph in a time series format. As you can see, both have similar shape and edges but not exactly as it is not a 100% accurate model. The above diagrams plot was done using **model11 as it was considered as the best model** out of all the models set up during the code run. As the shape is almost the same in both the actual and predicted TS plot -> we can conclude that the model prediction is mostly accurate.

Plot for Predicted and Actual Values




```
$RMSE
[1] 0.006083447

$MAE
[1] 0.004512063

$MAPE
[1] 0.00341404

$SMAPE
[1] 0.003416356
```

Model11 is taken as the best model due to having the lowest average error values compared to all the other models trained!

The above diagram represents – prediction values vs actual values. The red line represents the actual values, and the dots represent the predicted values. The further the dot is from the red line – shows the error / how it has not predicted the values 100%. Fortunately, when inspecting, we could see that most of the dots are close to the red line – so we could conclude that the model has predicted to some extent a low loss rate as shown in the index matrices!

Appendix (Code)

```
##### K MEANS CLUSTERING #####
library(ggplot2) # FOR PLOTTING
library(factoextra)
library(NbClust) # FOR K VALUE FIINDING NB CLUST
library(cluster)

# READING DATASET
library(readxl)
Whitewine_v6 <- read_excel("C:/My Files/ml cw/Whitewine_v6.xlsx")

# CHECKING STRUCURE OF DATASET
str(Whitewine_v6)

# REMOVING IF ANY NULL VALUES PRESENT
df <- na.omit(Whitewine_v6)

# CONVERTING DATASET TO DATAFRAME
dfNormZ <- as.data.frame((Whitewine_v6[1:11]))



### USING TUKEYS METHOD TO IDENTIFY OUTLIERS ###
# FUNCTION TO DETECT OUTLIERS
detect_outliers_tukey <- function(x) {
  qnt <- quantile(x, probs=c(0.25, 0.75), na.rm = TRUE)
  iqr <- IQR(x, na.rm = TRUE)
  fence_low <- qnt[1] - 1.5 * iqr
```

```
  fence_high <- qnt[2] + 1.5 * iqr
  outliers <- which(x < fence_low | x > fence_high)
  return(outliers)
}

# APPLYING THE ABOVE FUNTCION TO EACH VARIABLE/FEATURE IN THE DATASET
outliers_tukey <- apply(dfNormZ, 2, detect_outliers_tukey)

# COMBINE OUTLEIRS DETECTED FOR ALL FEATURES
all_outliers <- unique(unlist(outliers_tukey))

# REMOVING THE DETECTED OUTLIERS FROM THE DATASET
dfNormZ_no_outliers <- dfNormZ[-all_outliers, ]

# SCALING THE CLEANED DATASET
dfNormZ_no_outliers <- scale(dfNormZ_no_outliers)

# CONVERTING IT TO A DATA FRAME NOW
dfNormZ_no_outliers <- as.data.frame(dfNormZ_no_outliers)

# BOXPLOTTNG BOTH THE DATA FRAME WITH OUTLIERS AND WITHOUT OUTLIERS
boxplot(dfNormZ, main = "With Outliers")
boxplot(dfNormZ_no_outliers, main = "Without Outliers")

# VISUALIZING USING GPLOT BOTH THE DATA FRAME WITH OUTLIERS AND WITHOUT OUTLIERS
ggplot(data = dfNormZ, aes(x = dfNormZ[,9], y = dfNormZ[,9])) +
```

```
  geom_point() +
  labs(title = "With Outliers", x = "Variable 1", y = "Variable 2")
ggplot(data = dfNormZ_no_outliers, aes(x = dfNormZ_no_outliers[,9], y =
dfNormZ_no_outliers[,9])) +
  geom_point() +
  labs(title = "Without Outliers", x = "Variable 1", y = "Variable 2")



## PERFORM NB CLUST ##
# EUCLIDEAN
set.seed(10)
nb <- NbClust(dfNormZ_no_outliers, distance = "euclidean", min.nc = 2, max.nc = 10,
            method = "kmeans", index="all")

# MANHATTAN
set.seed(8)
nb <- NbClust(dfNormZ_no_outliers, distance = "manhattan", min.nc = 2, max.nc = 10, method
=
              "kmeans",     index="all")

# MAXIMUM
set.seed(6)
nb <- NbClust(dfNormZ_no_outliers, distance = "maximum", min.nc = 2, max.nc = 10, method
=
              "kmeans",     index="all")



# SILHOUTTE
set.seed(12)
```

```
silh_nb <- fviz_nbclust(dfNormZ_no_outliers, kmeans, method = "silhouette")
print(silh_nb)

# GAP STATS
set.seed(14)
gap_nb <- fviz_nbclust(dfNormZ_no_outliers, kmeans, method = "gap_stat")
print(gap_nb)

# ELBOW
k = 2:10
set.seed(42)
WSS = sapply(k, function(k) {kmeans(dfNormZ_no_outliers, centers=k)$tot.withinss})
plot(k, WSS, type="b", xlab= "Number of k", ylab="Within sum of squares")


# ELBOW FOR OPTIMUM CLUSTERS
wss_nb <- fviz_nbclust(dfNormZ_no_outliers, kmeans, method = "wss")
print(wss_nb)


### K MEANS CLUSTERING ###
# ASSINGING K VALUE
k_value = 2

# PASSING AND GETTING THE K MEANS
km <- kmeans(dfNormZ_no_outliers, centers= k_value, nstart= 20)
km
```

```
# CALCULATING CENTROIDS
centroids <- km$centers

# PLOTTING
# fviz_cluster(km,data=dfNormZ_no_outliers)

# PLOTTING
library(fpc)
plotcluster(dfNormZ_no_outliers, km$cluster)

# CALCULATING WSS AND BSS
wine_wss = km$tot.withinss
wine_wss
wine_bss = km$betweenss
wine_bss


# CALCULATING TSS -> SUM OF BOTH WSS AND BSS
wine_tss= wine_wss+wine_bss
wine_tss

# RATIO BETWEEN WSS AND TSS
ratio_wss_bss <- wine_wss / wine_tss
ratio_wss_bss

# RATIO BETWEEN BSS AND TSS
ratio_wss_tss <- wine_bss / wine_tss
ratio_wss_tss
```

```
# SILHOUTTE PLOT
silh <- silhouette(km$cluster, dist(dfNormZ_no_outliers))
fviz_silhouette(silh)

# AVERGAE WIDTH EXTRACTING AND PRINTING
avg_silh_width <- mean(silh[, "sil_width"])
print(paste("Average silhoette width score is ", avg_silh_width))




################## SUB TAST 2 - PCA ##################
# CALCULATING EIGEN VALUES AND EIGEN VECTORS
pca <- dfNormZ_no_outliers
wine.cov <- cov(pca)
# VALUES
wine.eigen <- eigen(wine.cov)
str(wine.cov)

# VECTORS
vectors <- wine.eigen$vectors
phi

# CALCULATING SCORES FOR ALL PC -> PRINCIPAL COMPONENTS
PC <- as.matrix(dfNormZ_no_outliers) %*% phi
PC
```

```
# CALCULATING SCORE PER PC
eigenVal <- wine.eigen$values
cumulative_scores <- cumsum(eigenVal) / sum(eigenVal)
cumulative_scores

# CALCULATING SELECTED CUMULATIVE SCORE - ANYTHING BELOW 0.85 THE THRESHOLD GIVEN
select_cumula_scores <- which(cumulative_scores > 0.85)[1]
select_cumula_scores

# CREATE A DATASET WITH THE PCA PROCESSED DATA
transformed <- as.data.frame(PC[, 1:select_cumula_scores])
head(transformed)
head(dfNormZ_no_outliers)



## PERFORM NB CLUST ##
# EUCLIDEAN
set.seed(10)
nb <- NbClust(transformed, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans", index="all")

# MANHATTAN
set.seed(8)
nb <- NbClust(transformed, distance = "manhattan", min.nc = 2, max.nc = 10, method = "kmeans", index="all")

# MAXIMUM
```

```
set.seed(6)
nb <- NbClust(transformed, distance = "maximum", min.nc = 2, max.nc = 10, method = "kmeans", index="all")

# ELBOW METHOD
k = 2:10
set.seed(42)
WSS = sapply(k, function(k) {kmeans(transformed, centers=k)$tot.withinss})
plot(k, WSS, type="b", xlab= "Number of k", ylab="Within sum of squares")

# ELBOW FOR OPTIMAL CLUSTER
wss_nb <- fviz_nbclust(transformed, kmeans, method = "wss")
print(wss_nb)

# SILHOUETTE
set.seed(12)
silh_nb <- fviz_nbclust(transformed, kmeans, method = "silhouette")
print(silh_nb)

# GAP STATS
set.seed(14)
gap_nb <- fviz_nbclust(transformed, kmeans, method = "gap_stat")
print(gap_nb)

### K MEANS CLUSTERING ###
# K VALUE
k_value = 2

# K MEANS
```

```
pca_km <- kmeans(transformed, centers= k_value, nstart= 20)
pca_km

# CALCULATE CENTROIDS
centroid_pca <- pca_km$centers

# PLOTTING
# fviz_cluster(pca_km,data=transformed)
plotcluster(transformed, pca_km$cluster)



# CALCULATE WSS AND BSS
wine_pca_wss = pca_km$tot.withinss
wine_pca_wss
wine_pca_bss = pca_km$betweenss
wine_pca_bss



# CALCULATING TSS -> SUM OF BOTH WSS AND BSS
wine_pca_tss= wine_pca_wss+wine_pca_bss
wine_pca_tss

# RATIO BETWEEN WSS AND TSS
ratio_wss_tss <- wine_pca_wss / wine_pca_tss
ratio_wss_tss

# RATIO BETWEEN BSS AND TSS
ratio_bss_tss <- wine_pca_bss / wine_pca_tss
```

```
ratio_bss_tss



# SILHOUTTE PLOT
silh <- silhouette(pca_km$cluster, dist(transformed))
fviz_silhouette(silh)



# AVERGAE WIDTH EXTRACTING AND PRINTING
avg_silh_width <- mean(silh[, "sil_width"])
print(paste("Average silhoette width score is ", avg_silh_width))

# CONVERTING DATASET INTO A FORMAT FOR CALINSKI HARABASZ
transformed <- dist(transformed)
str(transformed)

# CALINSKI HARABASZ INDEX CALCULATION
library(fpc)

calin_Harab_index <- cluster.stats(transformed, pca_km$cluster)$ch
print(calin_Harab_index)

##### NEURAL NETWORK #####
library(neuralnet) # NEURAL NETWORK
library(MLmetrics) # ERROR VALUES
library(readxl) # READ DATASET
library(ggplot2) # FOR PLOT
```

```
# SMAPE INBUILT FUNCTION DID NOT WORK SO HAD TO SET IT UP
smape <- function(actual_values, predictions) {
  return (mean(2 * abs(actual_values - predictions) / (abs(actual_values) + abs(predictions))))
}

# DEFINING NORMALIZATION AND DENORMALIZAITON
normalize <- function(x) {return((x - min(x)) / (max(x) - min(x)))}
unnormalize <- function(x, min, max) {return( (max - min)*x + min )}

# READING DATASET
ExchangeUSD <- read_excel("C:/My Files/ml cw/ExchangeUSD.xlsx")

# ADDING APRROPRIATE COL NAMES
colnames(ExchangeUSD) <- c("Input_1", "Input_2", "Output")

output_col_og_dataset <- ExchangeUSD[3]
head(output_col_og_dataset)

# GETTING LAGGED VALUES FOR FIRST FOUR OF THE THRID COL OF DATASET
lagged_output_1 <- lag(output_col_og_dataset, 1)
lagged_output_2 <- lag(output_col_og_dataset, 2)
lagged_output_3 <- lag(output_col_og_dataset, 3)
lagged_output_4 <- lag(output_col_og_dataset, 4)

# CREATING IO MATRICES WITH VARYING COMBINATIONS OF LAGGED VALUES
IO_matrix_1 <- cbind(lagged_output_1, lagged_output_2, lagged_output_3, lagged_output_4,
output_col_og_dataset)
IO_matrix_2 <- cbind(lagged_output_2, lagged_output_1, lagged_output_3, lagged_output_4,
output_col_og_dataset)
```

```
IO_matrix_3 <- cbind(lagged_output_4, lagged_output_3, lagged_output_2, lagged_output_1, output_col_og_dataset)

IO_matrix_4 <- cbind(lagged_output_3, lagged_output_4, lagged_output_2, lagged_output_1, output_col_og_dataset)


# PRITNING THE FIRST FEW TO INSPECT

head(IO_matrix_1)

head(IO_matrix_2)

head(IO_matrix_3)

head(IO_matrix_4)


# REMOVING NA VALUES OF THE IO MATRICES

IO_matrix_1 <- IO_matrix_1[complete.cases(IO_matrix_1),]

IO_matrix_2 <- IO_matrix_2[complete.cases(IO_matrix_2),]

IO_matrix_3 <- IO_matrix_3[complete.cases(IO_matrix_3),]

IO_matrix_4 <- IO_matrix_4[complete.cases(IO_matrix_4),]


# PRINTING THE FIRST FEW TO INSPECT

head(IO_matrix_1)

head(IO_matrix_2)

head(IO_matrix_3)

head(IO_matrix_4)


# NORMALIZING THE DATA USING MIN MAX

IO_matrix_1_normz <- normalize(IO_matrix_1)

IO_matrix_2_normz <- normalize(IO_matrix_2)

IO_matrix_3_normz <- normalize(IO_matrix_3)

IO_matrix_4_normz <- normalize(IO_matrix_4)
```

```
# PRINTING THE FIRST FEW TO INSPECT
head(IO_matrix_1_normz)
head(IO_matrix_2_normz)
head(IO_matrix_3_normz)
head(IO_matrix_4_normz)

str(IO_matrix_4_normz)
IO_matrix_4_normz

# RENAMING COL NAMES
colnames(IO_matrix_1_normz) <- c("lagged1", "lagged2","lagged3","lagged4", "actualOutput")
colnames(IO_matrix_2_normz) <- c("lagged1", "lagged2","lagged3","lagged4", "actualOutput")
colnames(IO_matrix_3_normz) <- c("lagged1", "lagged2","lagged3","lagged4", "actualOutput")
colnames(IO_matrix_4_normz) <- c("lagged1", "lagged2","lagged3","lagged4", "actualOutput")

# INSPECTING JUST ONE TO CHECK WHETHER COLUMN NAME CHANGED
head(IO_matrix_1_normz)


### DATASPLIT -> TRAINING 400 - REST FOR TESTING & CCONVERT TO
DATAFRAMES TO BE USED IN NN
# TRAINING PART
training_set_1 <- as.data.frame(IO_matrix_1_normz[1:400,])
training_set_2 <- as.data.frame(IO_matrix_2_normz[1:400,])
training_set_3 <- as.data.frame(IO_matrix_3_normz[1:400,])
training_set_4 <- as.data.frame(IO_matrix_4_normz[1:400,])

# TESTING PART
testing_set_1 <- as.data.frame(IO_matrix_1_normz[401:nrow(IO_matrix_1_normz),])
```

```
testing_set_2 <- as.data.frame(IO_matrix_2_normz[401:nrow(IO_matrix_2_normz),])
testing_set_3 <- as.data.frame(IO_matrix_3_normz[401:nrow(IO_matrix_3_normz),])
testing_set_4 <- as.data.frame(IO_matrix_4_normz[401:nrow(IO_matrix_4_normz),])


# PRINT FIRST FEW JUST TO CHECK WHETHER PROPERLY SPLITTED
nrow(training_set_1)
nrow(training_set_2)
nrow(training_set_3)
nrow(training_set_4)
nrow(testing_set_1)
nrow(testing_set_2)
nrow(testing_set_3)
nrow(testing_set_4)


##### CREATING MLP MODELS ####

# FUNCTION TO SET UP MODELS EASILY
train_mlp_model <- function(seed, input, train_data, hidden_layers, activation) {
  set.seed(seed)
  input_formula <- as.formula(paste("actualOutput ~", paste(input, collapse = " + ")))
  model <- neuralnet(input_formula, data = train_data, hidden = hidden_layers,
                     act.fct = activation, linear.output = TRUE)
  return(model)
}
```

```
# FUNCTION TO SET UP MODELS EASILY WWITHOUT FUNCTION
train_mlp_model_no_func <- function(seed, input, train_data, hidden_layers) {
  set.seed(seed)
  input_formula <- as.formula(paste("actualOutput ~", paste(input, collapse = " + ")))
  model <- neuralnet(input_formula, data = train_data, hidden = hidden_layers, linear.output =
TRUE)
  return(model)
}



# MODEL1
mlp_model1 <- train_mlp_model_no_func(12, c("lagged1", "lagged2", "lagged3", "lagged4"),
                          training_set_1, c(5))

# MODEL2
mlp_model2 <- train_mlp_model(14, c("lagged1", "lagged2", "lagged3", "lagged4"),
                          training_set_1, c(4, 6), "logistic")

# MODEL3
mlp_model3<- train_mlp_model_no_func(16, c("lagged1", "lagged3", "lagged4", "lagged2"),
                          training_set_1, c(6, 8))

# MODEL4
mlp_model4<- train_mlp_model(18, c("lagged1", "lagged3", "lagged2", "lagged4"),
                         training_set_1, c(4, 6, 8), "logistic")

# MODEL5
mlp_model5<- train_mlp_model(20, c("lagged1", "lagged4", "lagged3", "lagged2"),
                    training_set_1, c(8, 6), "tanh")
```

```
# MODEL6
mlp_model6<- train_mlp_model(22, c("lagged1", "lagged4", "lagged3", "lagged2"),
                    training_set_1, c(6, 8), "logistic")

# MODEL7
mlp_model7<- train_mlp_model(24, c("lagged2", "lagged1", "lagged3", "lagged4"),
                    training_set_2, c(4, 8, 4), "logistic")

# MODEL8
mlp_model8<- train_mlp_model(26, c("lagged2", "lagged3", "lagged4", "lagged1"),
                      training_set_2, c(6, 4), "tanh")

# MODEL9
mlp_model9<- train_mlp_model(28, c("lagged2", "lagged4", "lagged1", "lagged3"),
                      training_set_2, c(4, 6, 8), "tanh")

# MODEL10
mlp_model10<- train_mlp_model(30, c("lagged4", "lagged3", "lagged2", "lagged1"),
                      training_set_3, c(6, 6, 9), "tanh")

# MODEL11
mlp_model11<- train_mlp_model(32, c("lagged4", "lagged1", "lagged3", "lagged2"),
                       training_set_3, c(6, 4, 8), "logistic")

# MODEL12
mlp_model12<- train_mlp_model(34, c("lagged3", "lagged4", "lagged2", "lagged1"),
                       training_set_4, c(8), "tanh")
```

```
# MODEL13
mlp_model13<- train_mlp_model_no_func(36, c("lagged3", "lagged1", "lagged4", "lagged2"),
                        training_set_4, c(8, 4, 6))

# MODEL14
mlp_model14<- train_mlp_model(38, c("lagged3", "lagged2", "lagged1", "lagged4"),
                                training_set_4, c(10, 6, 8), "tanh")



# PLOTTING FOR INPECTION
plot(mlp_model1)
plot(mlp_model2)
plot(mlp_model3)
plot(mlp_model4)
plot(mlp_model5)
plot(mlp_model6)
plot(mlp_model7)
plot(mlp_model8)
plot(mlp_model9)
plot(mlp_model10)
plot(mlp_model11)
plot(mlp_model12)
plot(mlp_model13)
plot(mlp_model14)


#### MAKING PREDICTIONS ####
```

```
## DENORMALIZING THE PREDICTED VALUES AND ACTUAL VALUES ##
training_original_data <- output_col_og_dataset[1:400, ]
nrow(training_original_data)

# GETTING THE MIN AND MAX VALUE OF THE OUTPUT COL TO BE PASSED TO THE DENORMALIZING FUNCTION
min_output <- min(training_original_data$Output)
max_output <- max(training_original_data$Output)
paste("min output: " ,min_output)
paste("maz output: " ,max_output)

# CREATED THIS AS THE SAME CODE WILL BE USED TWICE IN THE LATTER PART OF THE CODE
calc_error_values_helper <- function(predicted, actual) {
  ### CALCULATING ERROR VALUES ####
  # CALC RMSE
  rmse <- RMSE(predicted, actual)

  # CALC MAE
  mae <- MAE(predicted, actual)

  # CALC MAPE
  mape <- MAPE(predicted, actual)

  # CALC SMAPE
  smape <- smape(predicted, actual)

  # RETURN LIST OF ALL THE ERROR STATS
```

```
  return(list(RMSE = rmse, MAE = mae, MAPE = mape, SMAPE = smape))
}


calc_error_stats <- function(model, testing_set) {
  # GENERATING THE PREDICTIONS USING THE MODEL AND THE TEST DATASET
  predictions <- predict(model, testing_set)

  # EXTRACTING PREDICTED VALUES
  predicted_values <- as.vector(predictions)
  actual_values <- testing_set$actualOutput

  # DDENORMALIZING THE PREDICTED VALUES
  predicted_values_denorm <- unnormalize(predicted_values, min_output, max_output)

  # DENORMALIZING THE ACTUAL VALUES
  actual_values_denorm <- unnormalize(actual_values, min_output, max_output)

  return (calc_error_values_helper(predicted_values_denorm, actual_values_denorm))

}


error_stats <- list()

is.na(testing_set_1)

# PASSING IN THE MODEL AND ITS RELEVANT TESTING SET
```

```
error_stats[[1]] <- calc_error_stats(mlp_model1, testing_set_1)
error_stats[[2]] <- calc_error_stats(mlp_model2, testing_set_1)
error_stats[[3]] <- calc_error_stats(mlp_model3, testing_set_1)
error_stats[[4]] <- calc_error_stats(mlp_model4, testing_set_1)
error_stats[[5]] <- calc_error_stats(mlp_model5, testing_set_1)
error_stats[[6]] <- calc_error_stats(mlp_model6, testing_set_1)
error_stats[[7]] <- calc_error_stats(mlp_model7, testing_set_2)
error_stats[[8]] <- calc_error_stats(mlp_model8, testing_set_2)
error_stats[[9]] <- calc_error_stats(mlp_model9, testing_set_2)
error_stats[[10]] <- calc_error_stats(mlp_model10, testing_set_3)
error_stats[[11]] <- calc_error_stats(mlp_model11, testing_set_3)
error_stats[[12]] <- calc_error_stats(mlp_model12, testing_set_4)
error_stats[[13]] <- calc_error_stats(mlp_model13, testing_set_4)
error_stats[[14]] <- calc_error_stats(mlp_model14, testing_set_4)

# DISPLAYIN IT AS A DATA FRAME FOR BETTER COMPARISON
error_stats_df <- data.frame(MLP_MODEL=paste("model -", 1:14),
                             RMSE = sapply(error_stats, function(x) x$RMSE),
                             MAE = sapply(error_stats, function(x) x$MAE),
                             MAPE = sapply(error_stats, function(x) x$MAPE),
                             SMAPE = sapply(error_stats, function(x) x$SMAPE))


# PRINTING THE DATA FRAME FOR A TABLE FORMAT - EASY TO DO COMPARISON
error_stats_df

# GENERATING THE PREDICTIONS USING THE MODEL AND THE TEST DATASET
predictions <- predict(mlp_model11, testing_set_3)
```

```
# EXTRACTING PREDICTED VALUES
predicted_values <- as.vector(predictions)
actual_values <- testing_set_2$actualOutput
nrow(testing_set_2)

# DDENORMALIZING THE PREDICTED VALUES
predicted_values_denorm <- unnormalize(predicted_values, min_output, max_output)

# DENORMALIZING THE ACTUAL VALUES
actual_values_denorm <- unnormalize(actual_values, min_output, max_output)



# PRIINTING TO COMPARE
predicted_values_denorm
actual_values_denorm
nrow(as.data.frame(predicted_values_denorm))
nrow(as.data.frame(actual_values_denorm))



# PLOT USING G PLOT FOR PREDICTION OUTPUT VS DESIRED OUTPUT
plot_data <- data.frame(Actual = actual_values_denorm, Predicted = predicted_values_denorm)
ggplot(plot_data, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  labs(x = "ACTUAL OUTPUT", y = "PREDICTED OUTPUT", title = "ACTUAL VALUES
(RED) VS PREDICTED VALUES(BLACK)")
```

```
# DATAFRAME CREATION FOR THE TIME SERIES PLOT
time_series_data <- data.frame(
  Index = 1:nrow(testing_set_2),
  Actual = actual_values_denorm,
  Predicted = predicted_values_denorm
)

# TIME SERIES OF BOTH IN ONE PLOT
ggplot(time_series_data, aes(x = Index)) +
  geom_line(aes(y = Actual, color = "ACTUAL OUTPUT")) +
  geom_line(aes(y = Predicted, color = "PREDICTED OUTPUT")) +
  labs(x = "TIME", y = "EXCHANGE RATE", title = "ACTUAL VALUES(BLUE) VS PREDICTED VALUES(RED)") +
  scale_color_manual(values = c("ACTUAL OUTPUT" = "blue", "PREDICTED OUTPUT" = "red")) +
  theme_minimal()


# GETTING THE ERROR VALUES INDIVIDUALLY AGAIN AS REQUIRED
calc_error_values_helper(predicted_values_denorm, actual_values_denorm)
```

Reference

K-means Clustering Analysis. Available on - https://uc-r.github.io/kmeans_clustering [last accessed 2nd April 2024]

Jaadi, Z. (23rd Feb 2024). A Step-by-Step Explanation of Principal Component Analysis (PCA). Available on - https://builtin.com/data-science/step-step-explanation-principal-component-analysis [last accessed 5th April 2024]

Calinski-Harabasz Index – Cluster Validity indices | Set 3 (25th April 2022) Available on - https://www.geeksforgeeks.org/calinski-harabasz-index-cluster-validity-indices-set-3/ [last accessed 14th April 2024]

Roberts, A. (2nd of Feb 2023). Mean Absolute Percentage Error (MAPE): What You Need To Know. Available on - https://arize.com/blog-course/mean-absolute-percentage-error-mape-what-you-need-to-know/#:~:text=point%20in%20time).-,Mean%20absolute%20percentage%20error%20measures%20the%20average%20magnitude%20of%20error,and%20the%20actuals%20is%2020%25 [last accessed 23rd of April 2024]

What is Mean Absolute Error (MAE)? Available on - (https://c3.ai/glossary/data-science/mean-absolute-error/#:~:text=What%20is%20Mean%20Absolute%20Error,true%20value%20of%20that%20observation [last accessed on 28th April 2024]