Machine Learning Coursework

Name: Om Sadigale

ID: 1943544

Year: 2nd

Course: Data Science and Analytics

Lecturer: Tom Oliver

Contents

Partitioning Clustering

1st Subtask Observations:

Preprocessing task:

The processing task involves both scaling and outlier detection. Scaling helps to manage data with varying units, while outlier detection is crucial for identifying data points that could significantly affect the results of various operations. The affect could mean and standard deviation. This is important because outliers can distort the accuracy of analyses and calculations.

The Interquartile Range (IQR) method is used to detect and remove outliers. This approach involves splitting the dataset into quartiles to measure its variability. By calculating the range between the first and third quartiles, we can identify potential outliers lying outside this range and subsequently remove them from the dataset. This method helps ensure that our data is more reliable and representative for further analysis.

Scaling data was utilized as part of the outlier detection process and showed multiple outliers for each column. This has reduced the no of observation to 2243.

The output of preprocessing:

```
> str(cleaned_data)
'data.frame':   2243 obs. of  11 variables:
 $ fixed acidity       : num  -0.53 -0.53 0.323 0.323 -0.896 ...
 $ volatile acidity    : num  -2.07 -2.07 -2.07 -2.07 -1.86 ...
 $ citric acid         : num  -0.0631 -0.0631 1.3108 1.3108 0.6238 ...
 $ residual sugar      : num  -0.874 -0.874 -0.995 -0.995 -1.055 ...
 $ chlorides           : num  -0.865 -0.865 0.208 0.208 -0.28 ...
 $ free sulfur dioxide : num  -0.793 -0.793 -1.113 -1.113 -1.177 ...
 $ total sulfur dioxide: num  -1.169 -1.169 0.157 0.157 -1.579 ...
 $ density             : num  -1.08 -1.08 0.12 0.12 -1.46 ...
 $ pH                  : num  0.898 0.898 1.669 1.669 -0.386 ...
 $ sulphates           : num  1.764 1.764 -0.477 -0.477 0.557 ...
 $ alcohol             : num  1.252 1.252 -0.213 -0.213 0.763 ...
```

Cluster Centres:

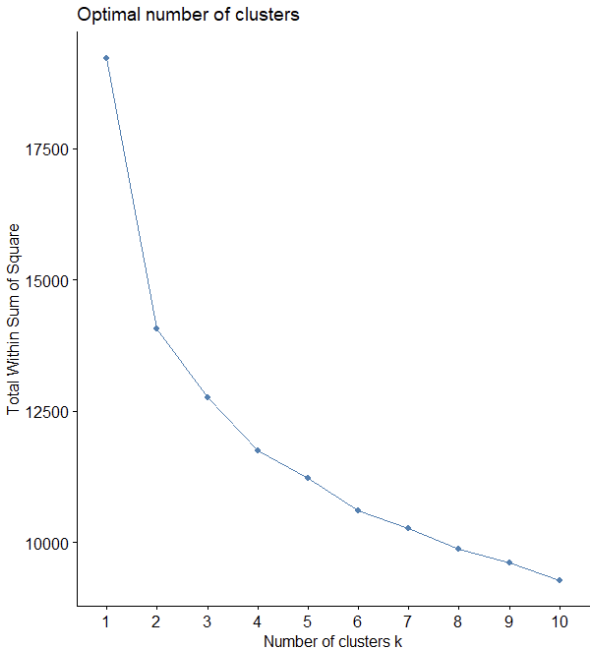
Cluster Centres are points at the centre of a cluster/group of variables.

Automated tools used for clustering and their outputs:

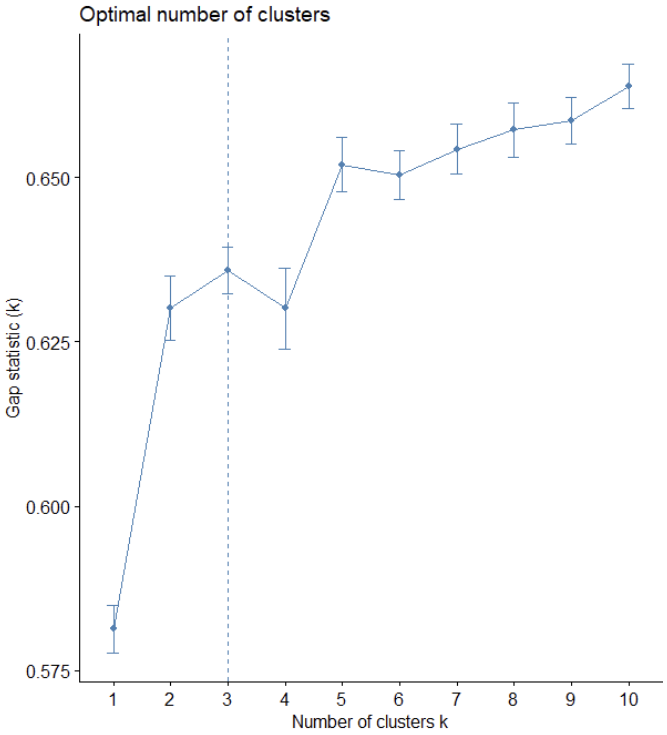
- NBclust:

```
*******************************************************************
* Among all indices:
* 12 proposed 2 as the best number of clusters
* 5 proposed 3 as the best number of clusters
* 3 proposed 5 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 2 proposed 13 as the best number of clusters
* 1 proposed 15 as the best number of clusters

                   ***** Conclusion *****

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


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

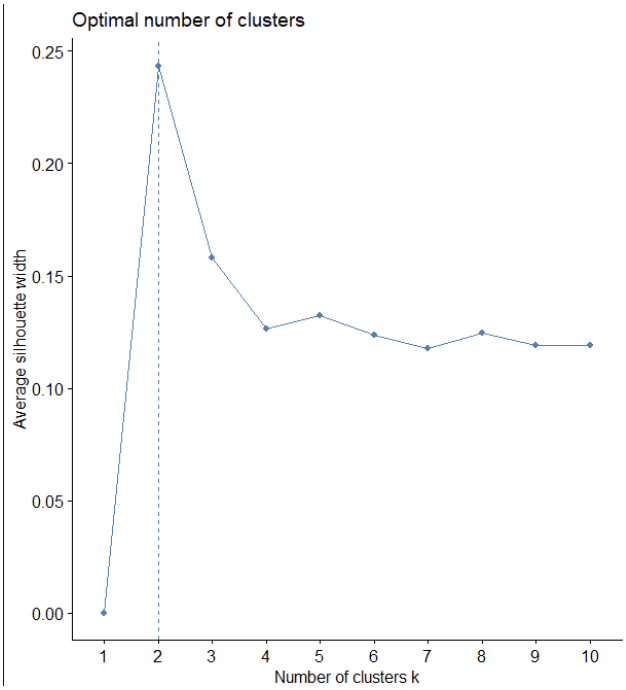
Elbow:



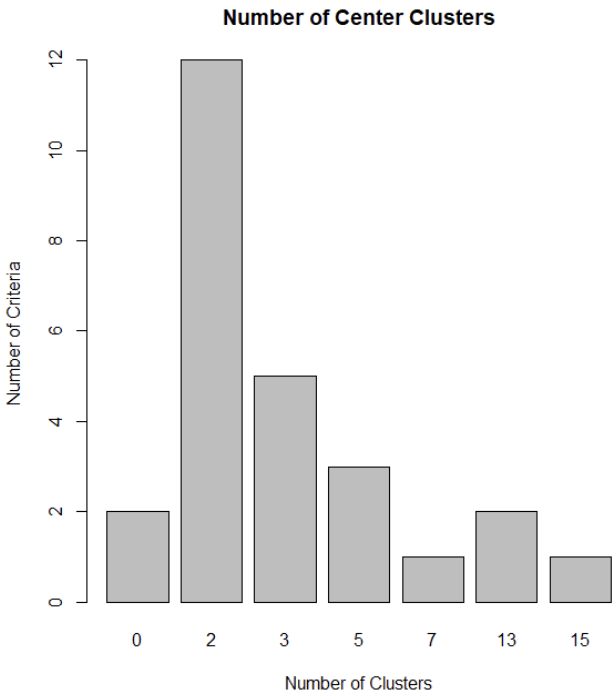
Gap statistics:



Silhouette:



Bar plot using NbClust Data



The automated tools used for determining the number of clusters provide insightful results:

According to NbClust, the optimal number of clusters is suggested to be 2, as indicated by the bar chart. This method considers various clustering criteria and indicates 2 clusters as the best choice.

In contrast, the Elbow Method suggests cluster 3 as a potential optimal solution. The curvature of the plot is notably higher at cluster 3 compared to other cluster values, indicating a possible inflection point.

Similarly, the Gap Statistics method supports the choice of 3 clusters. The graph of Gap Statistic exhibits a clear peak at k = 3, suggesting a significant improvement in clustering quality compared to fewer clusters.

Conversely, the Silhouette Method presents a maximum average silhouette value with 2 clusters, implying well-separated and compact clusters.

Considering the results from these automated tools, there is some inconsistency regarding the optimal number of clusters. While NbClust and Silhouette Method advocate for 2 clusters, Elbow Method and Gap Statistics lean towards 3 clusters. Each method provides valuable insights

Further analysis and consideration of the dataset's characteristics may be necessary to decide the appropriate number of clusters for the k-means clustering analysis

K-means analysis:

K-means Clustering is a form of Unsupervised Machine Learning, where the computer autonomously detects patterns in unsorted data. In this method, data points are grouped into clusters based on their similarity. The process involves continuously assigning each data point to the nearest cluster centroid, which represents the centre of each cluster. This process continues until the clusters stabilize and the centroids no longer change. By organizing data into clusters, K-means helps reveal underlying structures and relationships within the dataset without the need for labelled data.

The analysis involved testing two different numbers of clusters 2 and 3, to determine which yielded better-defined clusters. The ratio of between-cluster sum of squares (BSS) to total sum of squares (TSS) was used to measure the quality of the cluster. For the 3-cluster solution, the BSS/TSS ratio was 33.5%, indicating relatively well-defined clusters while the other was 26%. However, there was an issue where the within-cluster sum of squares (WSS) exceeded the BSS indicating ineffective cluster separation. Despite attempting multiple adjustments, the weight of the values remained same. Nonetheless, the 3-cluster solution was chosen due to its higher BSS/TSS ratio and overall clearer clustering.

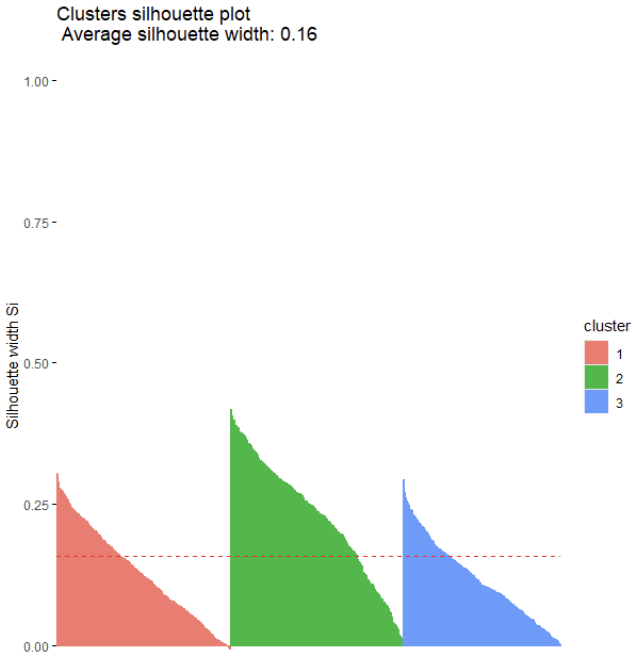
The calculations for WSS, BSS, TSS and BSS/TSS Ratio:

Values	
BSS	6425.61698744207
BSS_TSS_ratio	0.334367097533046
TSS	19217.2526389413
WSS	12791.6356514992

Output of K-means:

kmeans	list [9] (S3: kmeans)	List of length 9
cluster	integer [2243]	3 3 3 3 1 3 ...
centers	double [3 x 11]	0.0628 0.1182 -0.3255 0.0757 0.0111 -0.4029 -0.0421 -0.0263 -0.1950 -0.5318 ...
totss	double [1]	19217.25
withinss	double [3]	4493 4346 3953
tot.withinss	double [1]	12791.64
betweenss	double [1]	6425.617
size	integer [3]	774 769 700
iter	integer [1]	3
ifault	integer [1]	0

Silhouette plot:



The silhouette plot offered valuable insights into our clustering results. We observed that all clusters, with similar sizes, had silhouette widths above the average line. This suggests that our clusters are well-defined and each data point is appropriately assigned to its respective cluster. In summary, the silhouette plot indicates how well each data point fits into its cluster.

	cluster	size	ave.sil.width
1	1	774	0.13
2	2	769	0.23
3	3	700	0.12

2nd Subtask Objectives:

PCA:

Principal Component Analysis (PCA) was utilized to simplify the complex structure of the wine dataset, reducing its dimensionality while preserving as much information as possible. This technique helped to transform the multi-dimensional problem into a more manageable form with lose some information.

By applying PCA to the wine dataset, we computed eigenvalues and eigenvectors, which provided us with essential information about the direction and magnitude of variance in the data. The cumulative score, which represents the cumulative proportion of variance helped in the selection process.

It retained principal components that collectively explain at least 85% of the total variance. After careful analysis, PC7 emerged as the optimal choice, with a cumulative score of 89%. This indicates that PC7 captures a substantial portion of the dataset's variability, allowing us to gain critical information while achieving significant dimensionality reduction.

Using PCA and selecting PC7 helps simplify and clarify the wine dataset, trading its important aspects for reduction in dimensionality. This method makes it easier for us to understand the data and find useful information, improving our ability to make smart choices in future analyses.

```
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10    PC11
Standard deviation     1.8743 1.2161 1.1101 1.03730 0.98968 0.87907 0.84185 0.76831 0.59819 0.52681 0.11417
Proportion of Variance 0.3194 0.1344 0.1120 0.09782 0.08904 0.07025 0.06443 0.05366 0.03253 0.02523 0.00119
Cumulative Proportion  0.3194 0.4538 0.5658 0.66367 0.75271 0.82296 0.88739 0.94106 0.97358 0.99881 1.00000
```

Cluster Centres:

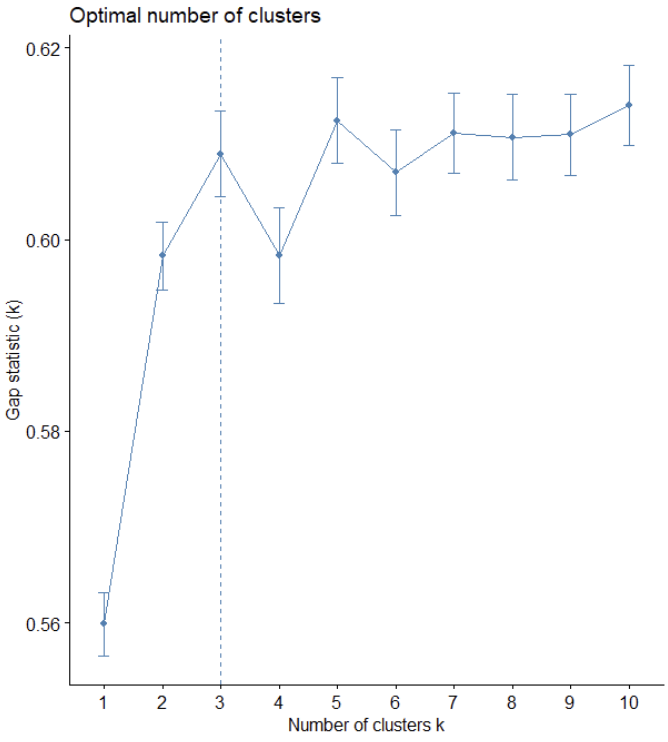
PCA Wine Data was used.

Automated tools used for clustering and their outputs:

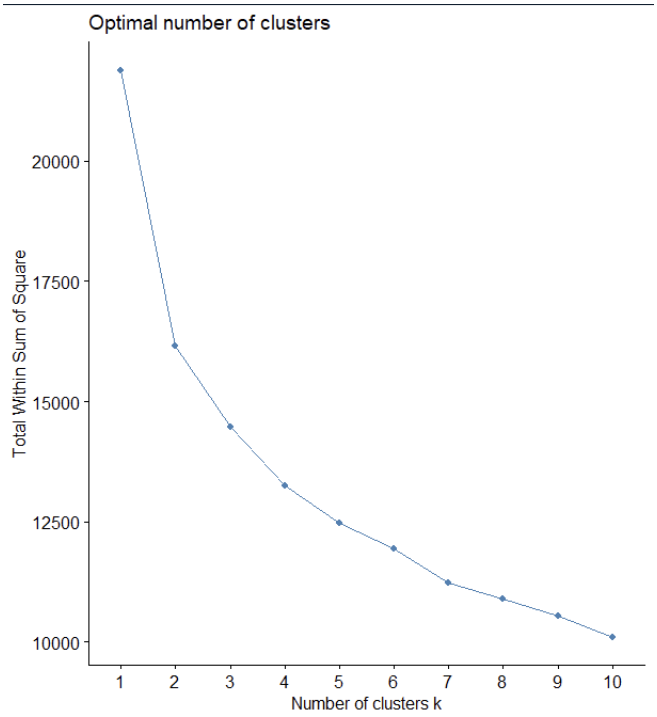
- NBclust:

```
*******************************************************************
* Among all indices:
* 12 proposed 2 as the best number of clusters
* 6 proposed 3 as the best number of clusters
* 3 proposed 4 as the best number of clusters
* 1 proposed 6 as the best number of clusters
* 1 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


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

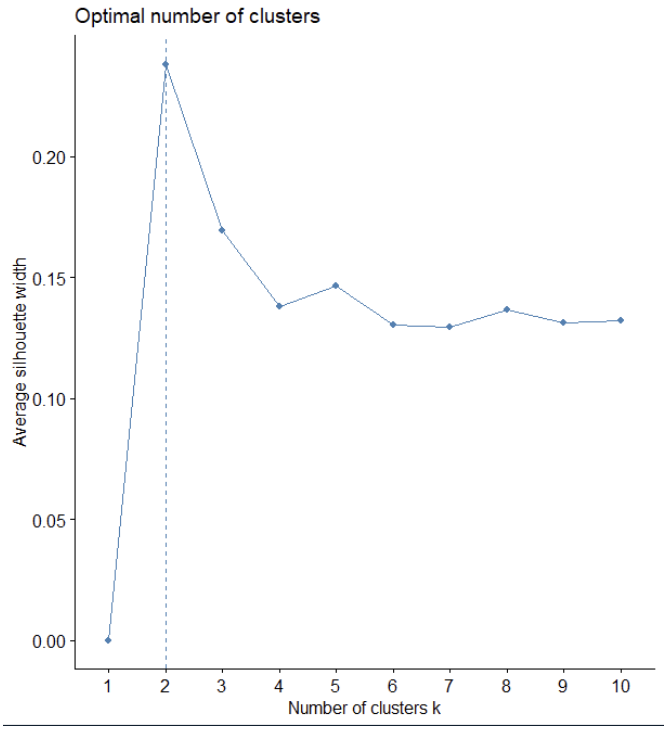
Gap statistics:



Elbow:

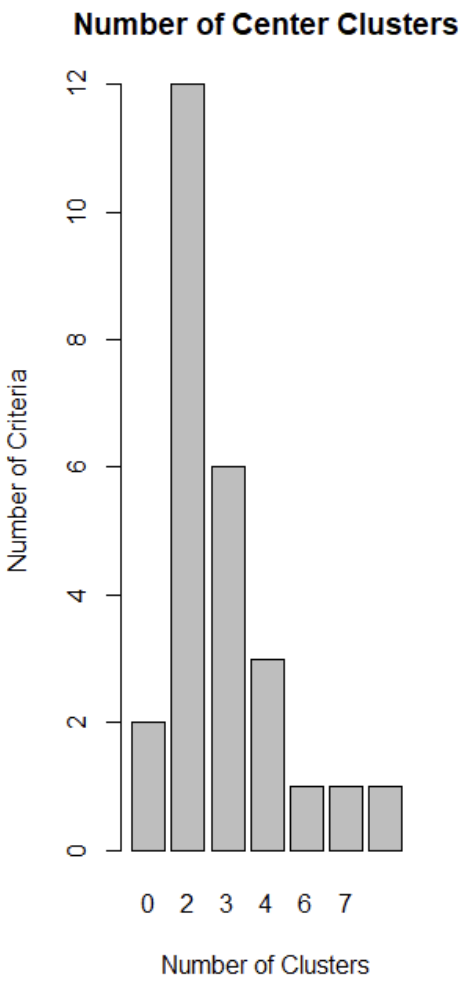


Silhouette:



Bar plot using NbClust Data

According to NbClust, 2 clusters are best. The Elbow Method leans towards 3 clusters due to a noticeable bend in the plot. Similarly, the Gap Statistics method also supports 3 clusters with a clear peak. However, the Silhouette Method suggests 2 clusters with the highest silhouette value, indicating well-separated clusters.



PCA K-means analysis:

Like the normal wine data k-means analysis it involved testing two different numbers of clusters 2 and 3, to determine which yielded better-defined clusters. The ratio of between-cluster sum of squares (BSS) to total sum of squares (TSS) was used to measure the quality of the cluster. For k = 2, the BSS/TSS ratio was 26.2%. while k = 3it was 33.9%, indicates k = 3 is relatively well-defined. As the problem faced above the WSS was higher than BSS.

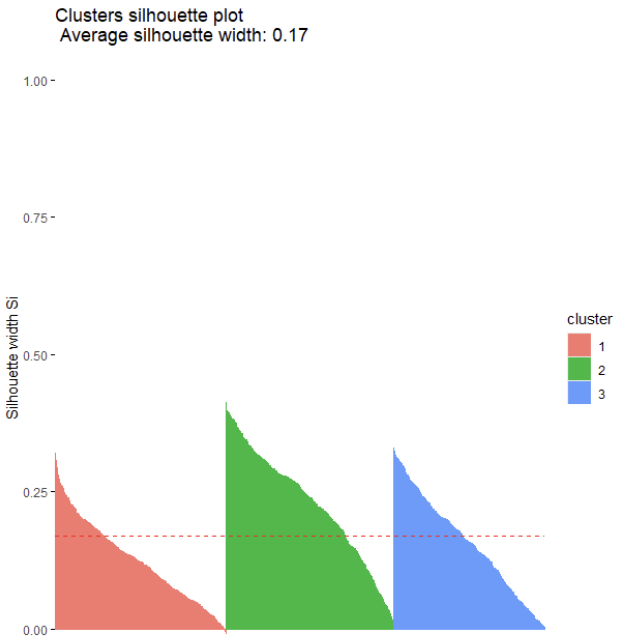
The calculations for WSS, BSS, TSS and BSS/TSS Ratio:

Values	
BSS_pca	7426.74000386269
BSS_TSS_ratio_pca	0.339355089629192
cumulative_score	num [1:11] 0.319 0.454 0.566 0.664 0.753 ...
eigenvalues	num [1:11] 3.513 1.479 1.232 1.076 0.979 ...
k	3
num_components	7L
TSS_pca	21884.8640578156
WSS_pca	14458.1240539529

PCA K-means output

kmeans_pca	list [9] (S3: kmeans)	List of length 9
cluster	integer [2243]	3 3 3 3 3 3 ...
centers	double [3 x 7]	1.8728 -2.1568 0.4619 0.3073 -0.0518 -0.2216 -0.5435 -0.2102 0.6892 -0.4373 ...
totss	double [1]	21884.86
withinss	double [3]	4491 4885 5082
tot.withinss	double [1]	14458.12
betweenss	double [1]	7426.74
size	integer [3]	693 769 781
iter	integer [1]	3
ifault	integer [1]	0

Silhouette plot:



As seen all the clusters are above the average silhouette width which indicates that they are well defined.

	cluster	size	ave.sil.width
1	1	781	0.12
2	2	769	0.23
3	3	693	0.16

Calinski-Harabasz:

It tells how well defined a cluster is. Higher Calinski-Harabasz Index score indicates better-defined and more separated clusters. The score for the PCA Wine data k = 2 is 796.15 and k=3 is 575.3. which indicates that k= 2 is a better-defined cluster according to Calinski-Harabasz.

Financial Forecasting:

In exchange rates forecasting, defining the input vector involves selecting relevant features or variables that can provide valuable information for predicting future exchange rate movements. Here's a brief discussion of various methods used for defining the input vector in exchange rates forecasting:

Fundamental Analysis:

Fundamental analysis involves analysing economic, political, and social factors that may influence exchange rate movements.

Economic indicators such as GDP growth, inflation rates, and interest rates are key inputs.

Geopolitical events, central bank policies, and trade balances also play a significant role.

Sentiment Analysis:

Sentiment analysis involves gauging market sentiment or investor sentiment using textual data from news articles, social media, and other sources.

Natural language processing (NLP) techniques are used to analyse sentiment and extract relevant features.

Sentiment indicators can provide insights into market expectations and sentiment-driven movements in exchange rates.

Machine Learning and Artificial Intelligence:

Machine learning and artificial intelligence techniques are increasingly used in exchange rate forecasting.

These methods can automatically learn patterns from historical data and adapt to changing market conditions.

Methods like Support Vector Machines (SVM) and Neural Networks, especially Recurrent Neural Networks (RNNs), can automatically select and combine a wide range of features, including complex patterns in the data, for forecasting

Neural networks, support vector machines (SVM), and random forests are common ML algorithms used in this domain.

Time-Series Analysis:

Time-series analysis involves analysing data points collected at successive time intervals. In exchange rate forecasting, historical exchange rate data are used to predict future movements. Autoregressive (AR) and Moving Average (MA) models, as well as their combinations like ARIMA, use time series data to forecast future values based on the assumption that future values are a linear combination of past values and stochastic terms.

Technical Indicators:

Technical indicators are mathematical calculations based on historical price, volume, or open interest data.

These indicators help traders and analysts analyse market trends, volatility, and momentum.

Popular indicators include moving averages, Relative Strength Index (RSI), and Bollinger Bands.

These methods provide a diverse set of tools for analysts and traders to forecast exchange rates. By combining multiple approaches and leveraging the strengths of each, more accurate predictions can often be achieved.

However, we will be utilizing the Autoregressive (AR) approach AR, i.e. time-delayed values of the exchange rates. We will experiment with various input vectors up to the (t-4) level.

Creating input/output matrices (I/O)

The Daily exchange rate data file has 500 records of exchange rate. This dataset will be divided into 2 sets, first 400 records will be utilized for training and the remaining 100 records will be used for testing.

This involves creating input/output matrices (I/O) for MLP training/testing using up to 4 time-delayed exchange rate values from each dataset. We will iterate through different lag orders and construct the I/O matrices accordingly.

An input matrix is created where each row represents a sample, and each column represents a lagged value of the exchange rate.

The number of columns in the input matrix equals the lag order. Each row of the input matrix contains lagged values of the exchange rate for a specific time step as shown before for the various sets of input vectors.

```
$X
             [,1]
  [1,] 0.04959631
  [2,] 0.15686275
  [3,] 0.10495963
  [4,] 0.19031142
  [5,] 0.16839677
  [6,] 0.18685121
  [7,] 0.11072664
  [8,] 0.19492503
  [9,] 0.30911188
```

```
$X
             [,1]       [,2]
  [1,] 0.15686275 0.04959631
  [2,] 0.10495963 0.15686275
  [3,] 0.19031142 0.10495963
  [4,] 0.16839677 0.19031142
  [5,] 0.18685121 0.16839677
  [6,] 0.11072664 0.18685121
  [7,] 0.19492503 0.11072664
  [8,] 0.30911188 0.19492503
  [9,] 0.24567474 0.30911188
```

```
$x
            [,1]       [,2]       [,3]
 [1,] 0.10495963 0.15686275 0.04959631
 [2,] 0.19031142 0.10495963 0.15686275
 [3,] 0.16839677 0.19031142 0.10495963
 [4,] 0.18685121 0.16839677 0.19031142
 [5,] 0.11072664 0.18685121 0.16839677
 [6,] 0.19492503 0.11072664 0.18685121
 [7,] 0.30911188 0.19492503 0.11072664
 [8,] 0.24567474 0.30911188 0.19492503
 [9,] 0.37254902 0.24567474 0.30911188
```

```
$x
            [,1]       [,2]       [,3]       [,4]
 [1,] 0.19031142 0.10495963 0.15686275 0.04959631
 [2,] 0.16839677 0.19031142 0.10495963 0.15686275
 [3,] 0.18685121 0.16839677 0.19031142 0.10495963
 [4,] 0.11072664 0.18685121 0.16839677 0.19031142
 [5,] 0.19492503 0.11072664 0.18685121 0.16839677
 [6,] 0.30911188 0.19492503 0.11072664 0.18685121
 [7,] 0.24567474 0.30911188 0.19492503 0.11072664
 [8,] 0.37254902 0.24567474 0.30911188 0.19492503
```

An output vector containing the exchange rate values for the next time step corresponding to each sample in the input matrix. The length of the output vector is equal to the total number of samples minus the lag order, as we cannot predict the next time step for the last lagged sample.

```
$y
 [1] 0.15686275 0.10495963 0.19031142 0.16839677 0.18685121 0.11072664 0.19492503 0.30911188
 [9] 0.24567474 0.37254902 0.34602076 0.36216840 0.54440600 0.53979239 0.53056517 0.58131488
[17] 0.66897347 0.62283737 0.64013841 0.65282584 0.73010381 0.72549020 0.50749712 0.39100346
[25] 0.36447520 0.35294118 0.26643599 0.27912341 0.27220300 0.27912341 0.26643599 0.16493656
[33] 0.06574394 0.10265283 0.00000000 0.08304498 0.31372549 0.31487889 0.30565167 0.42560554
[41] 0.36908881 0.34832757 0.42329873 0.48212226 0.50403691 0.52364475 0.53056517 0.57208766
[49] 0.56286044 0.56747405 0.59861592 0.50980392 0.57093426 0.59976932 0.63783160 0.71164937
[57] 0.65167243 0.62168397 0.54555940 0.55478662 0.56401384 0.63091119 0.66205306 0.74971165
[65] 0.68512111 0.67704729 0.71164937 0.68512111 0.70703576 0.63898501 0.52133795 0.48673587
[73] 0.44867359 0.50519031 0.39792388 0.45213379 0.55940023 0.56170704 0.60668973 0.62283737
[81] 0.57785467 0.66320646 0.67128028 0.66435986 0.86735871 0.86389850 0.82814302 0.82468281
[89] 0.87773933 0.81891580 0.88004614 0.87658593 0.87658593 0.94348328 1.00000000 0.93310265
[97] 0.91695502 0.93079585 0.85121107
```

```
$y
 [1] 0.10495963 0.19031142 0.16839677 0.18685121 0.11072664 0.19492503 0.30911188 0.24567474
 [9] 0.37254902 0.34602076 0.36216840 0.54440600 0.53979239 0.53056517 0.58131488 0.66897347
[17] 0.62283737 0.64013841 0.65282584 0.73010381 0.72549020 0.50749712 0.39100346 0.36447520
[25] 0.35294118 0.26643599 0.27912341 0.27220300 0.27912341 0.26643599 0.16493656 0.06574394
[33] 0.10265283 0.00000000 0.08304498 0.31372549 0.31487889 0.30565167 0.42560554 0.36908881
[41] 0.34832757 0.42329873 0.48212226 0.50403691 0.52364475 0.53056517 0.57208766 0.56286044
[49] 0.56747405 0.59861592 0.50980392 0.57093426 0.59976932 0.63783160 0.71164937 0.65167243
[57] 0.62168397 0.54555940 0.55478662 0.56401384 0.63091119 0.66205306 0.74971165 0.68512111
[65] 0.67704729 0.71164937 0.68512111 0.70703576 0.63898501 0.52133795 0.48673587 0.44867359
[73] 0.50519031 0.39792388 0.45213379 0.55940023 0.56170704 0.60668973 0.62283737 0.57785467
[81] 0.66320646 0.67128028 0.66435986 0.86735871 0.86389850 0.82814302 0.82468281 0.87773933
[89] 0.81891580 0.88004614 0.87658593 0.87658593 0.94348328 1.00000000 0.93310265 0.91695502
[97] 0.93079585 0.85121107
```

```
$y
 [1] 0.19031142 0.16839677 0.18685121 0.11072664 0.19492503 0.30911188 0.24567474 0.37254902
 [9] 0.34602076 0.36216840 0.54440600 0.53979239 0.53056517 0.58131488 0.66897347 0.62283737
[17] 0.64013841 0.65282584 0.73010381 0.72549020 0.50749712 0.39100346 0.36447520 0.35294118
[25] 0.26643599 0.27912341 0.27220300 0.27912341 0.26643599 0.16493656 0.06574394 0.10265283
[33] 0.00000000 0.08304498 0.31372549 0.31487889 0.30565167 0.42560554 0.36908881 0.34832757
[41] 0.42329873 0.48212226 0.50403691 0.52364475 0.53056517 0.57208766 0.56286044 0.56747405
[49] 0.59861592 0.50980392 0.57093426 0.59976932 0.63783160 0.71164937 0.65167243 0.62168397
[57] 0.54555940 0.55478662 0.56401384 0.63091119 0.66205306 0.74971165 0.68512111 0.67704729
[65] 0.71164937 0.68512111 0.70703576 0.63898501 0.52133795 0.48673587 0.44867359 0.50519031
[73] 0.39792388 0.45213379 0.55940023 0.56170704 0.60668973 0.62283737 0.57785467 0.66320646
[81] 0.67128028 0.66435986 0.86735871 0.86389850 0.82814302 0.82468281 0.87773933 0.81891580
[89] 0.88004614 0.87658593 0.87658593 0.94348328 1.00000000 0.93310265 0.91695502 0.93079585
[97] 0.85121107
```

```
$y
 [1] 0.16839677 0.18685121 0.11072664 0.19492503 0.30911188 0.24567474 0.37254902 0.34602076
 [9] 0.36216840 0.54440600 0.53979239 0.53056517 0.58131488 0.66897347 0.62283737 0.64013841
[17] 0.65282584 0.73010381 0.72549020 0.50749712 0.39100346 0.36447520 0.35294118 0.26643599
[25] 0.27912341 0.27220300 0.27912341 0.26643599 0.16493656 0.06574394 0.10265283 0.00000000
[33] 0.08304498 0.31372549 0.31487889 0.30565167 0.42560554 0.36908881 0.34832757 0.42329873
[41] 0.48212226 0.50403691 0.52364475 0.53056517 0.57208766 0.56286044 0.56747405 0.59861592
[49] 0.50980392 0.57093426 0.59976932 0.63783160 0.71164937 0.65167243 0.62168397 0.54555940
[57] 0.55478662 0.56401384 0.63091119 0.66205306 0.74971165 0.68512111 0.67704729 0.71164937
[65] 0.68512111 0.70703576 0.63898501 0.52133795 0.48673587 0.44867359 0.50519031 0.39792388
[73] 0.45213379 0.55940023 0.56170704 0.60668973 0.62283737 0.57785467 0.66320646 0.67128028
[81] 0.66435986 0.86735871 0.86389850 0.82814302 0.82468281 0.87773933 0.81891580 0.88004614
[89] 0.87658593 0.87658593 0.94348328 1.00000000 0.93310265 0.91695502 0.93079585 0.85121107
```

Normalization of data:

Before using them to generate input/output (I/O) matrices, both the training and testing datasets undergo normalizaStion.

Normalization of data is essential before using it in an MLP structure. This standard procedure ensures that all input variables are on the same scale, preventing certain variables from dominating others during training. Normalization improves the convergence of the training algorithm and helps in achieving better performance of the neural network.

```
17  # Each one of these I/O matrices needs to be normalised,
18  # Function to normalize data
19  # The next step is to normalize the data so that all the features are on the same scale.
20  #As seen from the data exploration step, the values of all the attributes are more or less i
21  normalize <- function(x) {
22    (x - min(x)) / (max(x) - min(x))
23  }
24
25  # Normalize training and testing data
26  norm_training_data <- normalize(training_data)
27  norm_testing_data <- normalize(testing_data)
28
29  head(training_data)
30  head (norm_training_data)
31  head(testing_data)
32  head (norm_testing_data)
33
```

20:38 (Top Level) R Script

Console Terminal Background Jobs

R 4.4.0 · ~/

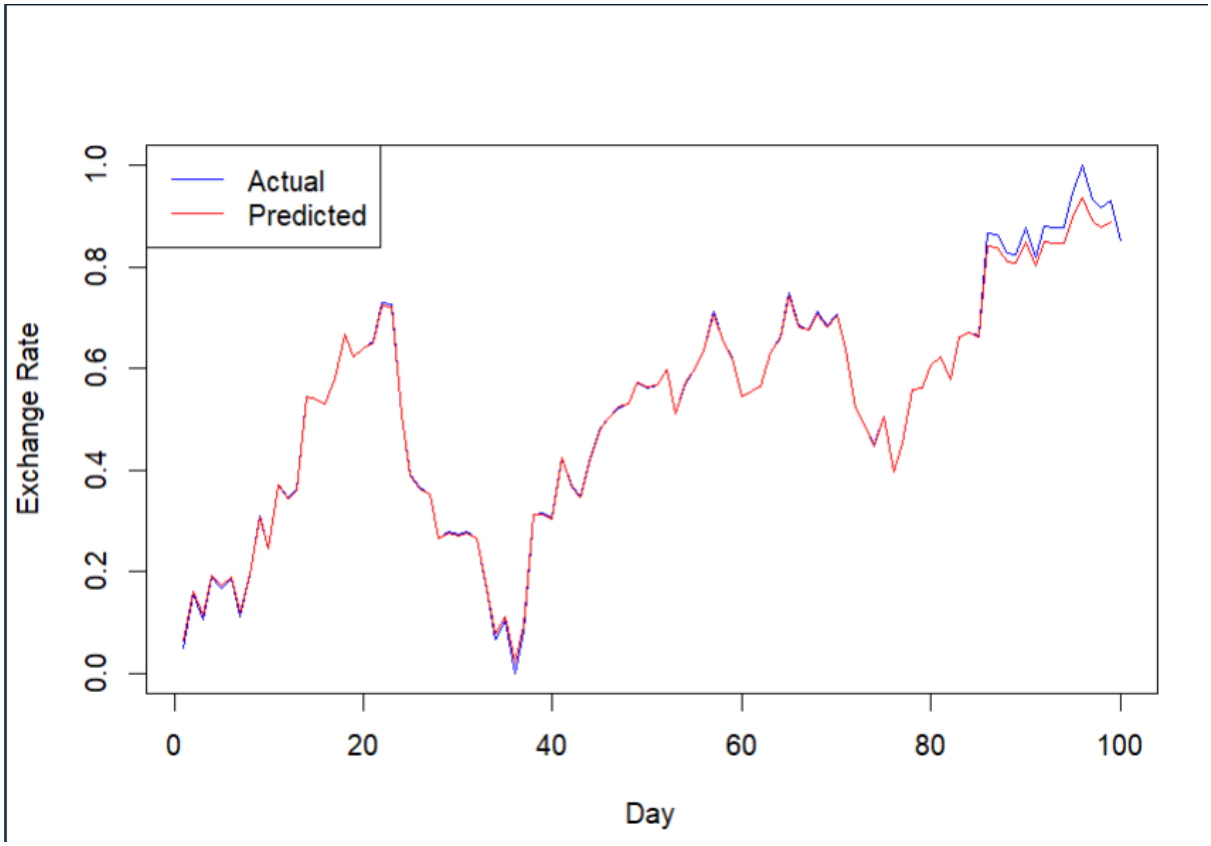
```
[89] 0.87658593 0.87658593 0.94348328 1.00000000 0.93310265 0.91695502 0.93079585 0.85121107

> head(training_data)
[1] 1.3730 1.3860 1.3768 1.3718 1.3774 1.3672
> head (norm_training_data)
[1] 0.7909953 0.8526066 0.8090047 0.7853081 0.8118483 0.7635071
> head(testing_data)
[1] 1.2817 1.2910 1.2865 1.2939 1.2920 1.2936
> head (norm_testing_data)
[1] 0.04959631 0.15686275 0.10495963 0.19031142 0.16839677 0.18685121
```

the normalize () function here takes a numeric vector as input and scales its values using min-max normalization, ensuring that all values fall within the range [0, 1] as shown in the above screenshot.

Also, the point to be noted is the normalized data gives better results than the denormalized data. To demonstrate, the predicted values are denormalized and compared with the original test data.

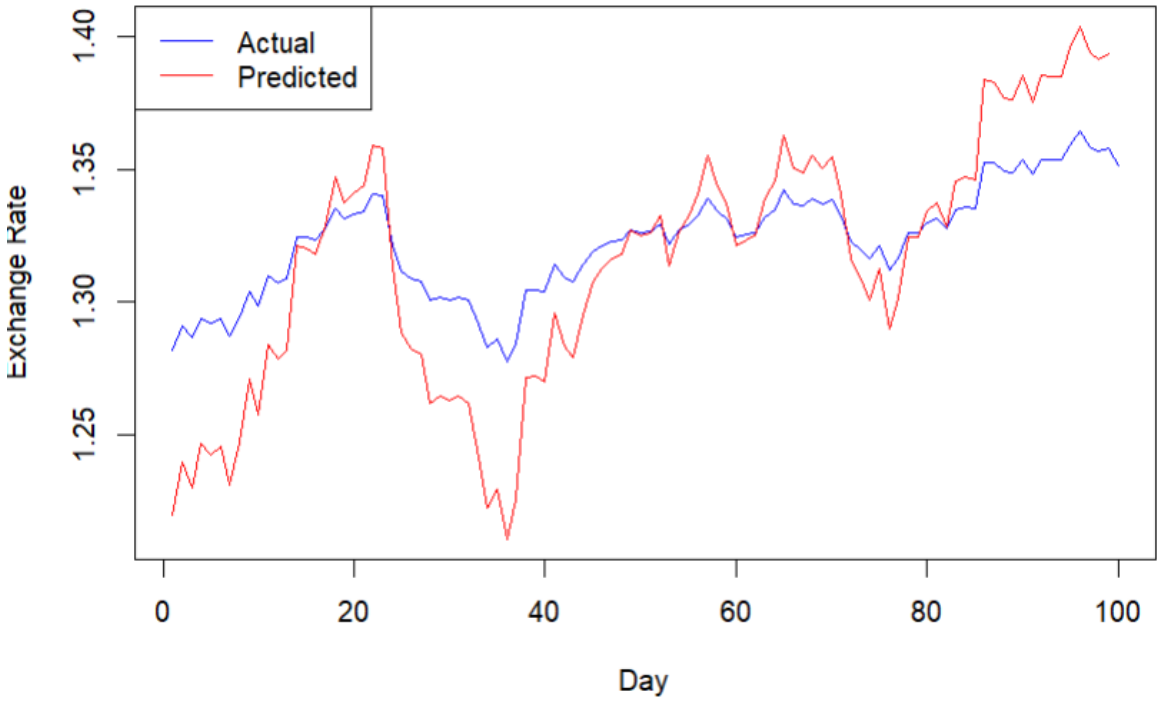
Normalized data output:

```
> head(predicted_values_best)
          [,1]
[1,] 0.0645970
[2,] 0.1607575
[3,] 0.1134536
[4,] 0.1919795
[5,] 0.1714602
[6,] 0.1887238
> head(norm_testing_data)
[1] 0.04959631 0.15686275 0.10495963 0.19031142 0.16839677 0.18685121
```



Denormalized data output:

```
> head(strength_pred)  # this is NN's output denormalized to original ranges
         [,1]
[1,] 1.219730
[2,] 1.240020
[3,] 1.230039
[4,] 1.246608
[5,] 1.242278
[6,] 1.245921
> head(testing_data)
[1] 1.2817 1.2910 1.2865 1.2939 1.2920 1.2936
```



MLP Models:

For training MLP Models, we will experiment with various MLP models, utilizing different input vectors and internal network structures. This includes varying the number of hidden layers, nodes, activation functions, and output layer configurations. For each case, we will evaluate the testing performance using standard statistical indices such as RMSE, MAE, MAPE, and sMAPE.

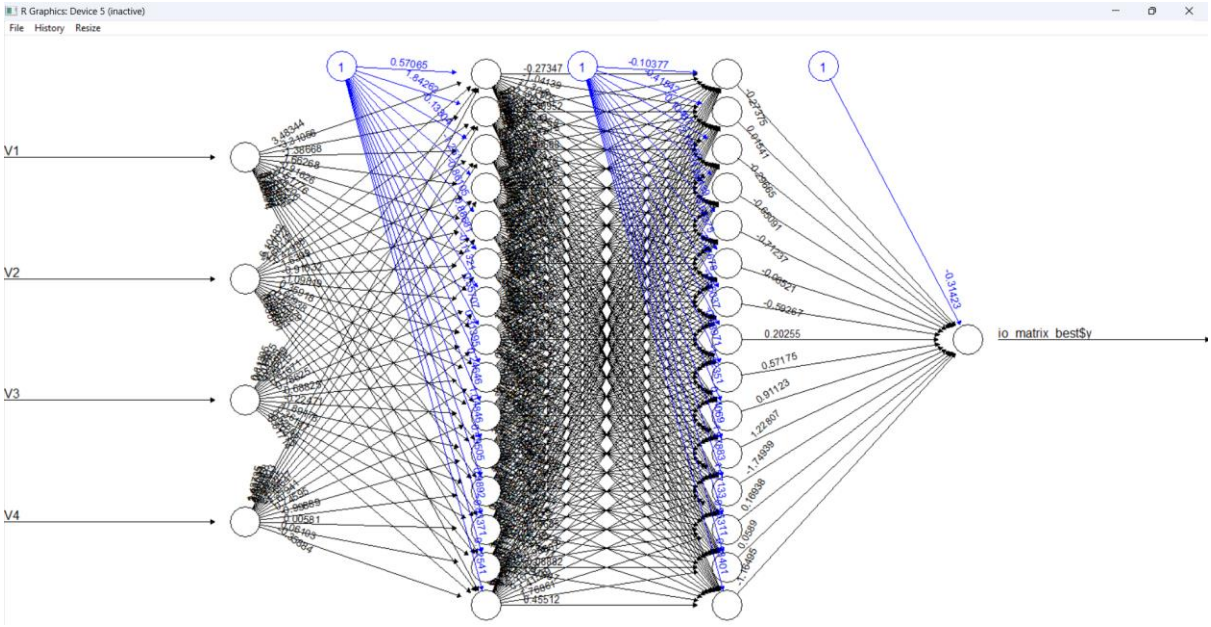
The number of hidden layers will be 1 or 2 with two sets of neural nodes and we have 4 inputs and 1 output. The maximum number of MLP models generated will be 4*2*2 = 16.

The number of neural nodes which will give us best MLP Model can be determined by trying various combinations and generation the models and the comparing the actual and predicted values.

The MLP Models compared for various node values:

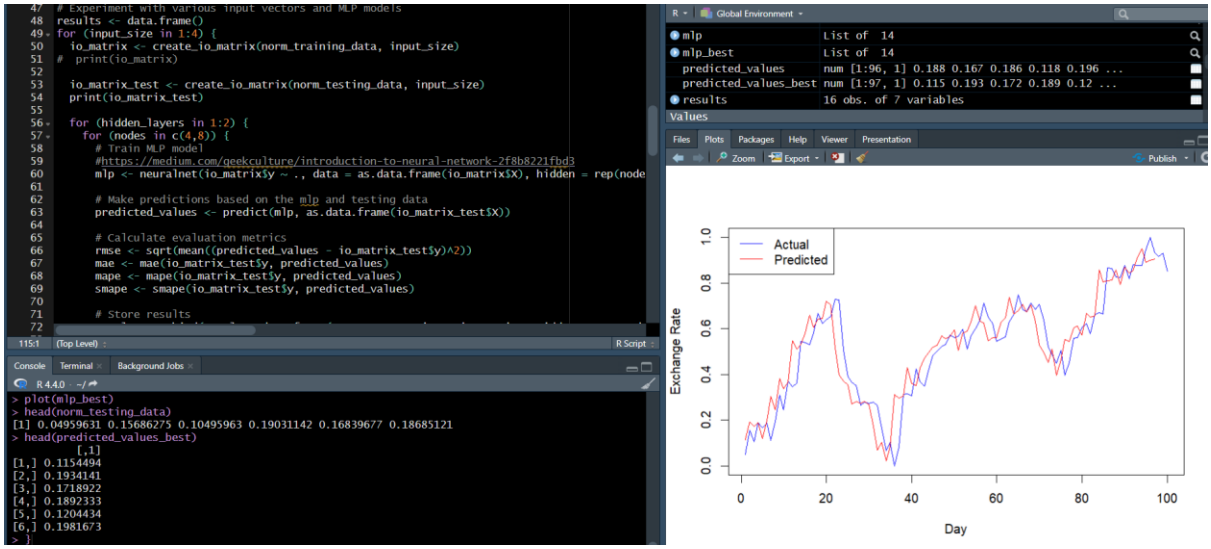
Nodes (10,15):

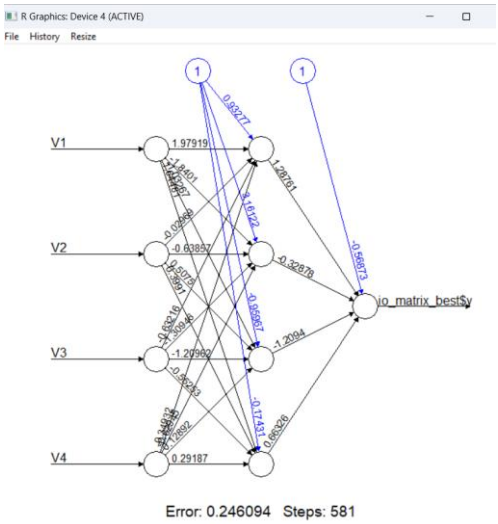
When the number of Neural nodes is 10 or 15, the best MLP model generated has the variance between the Actual and Predicted values as seen the plot below.



Nodes(4,8):

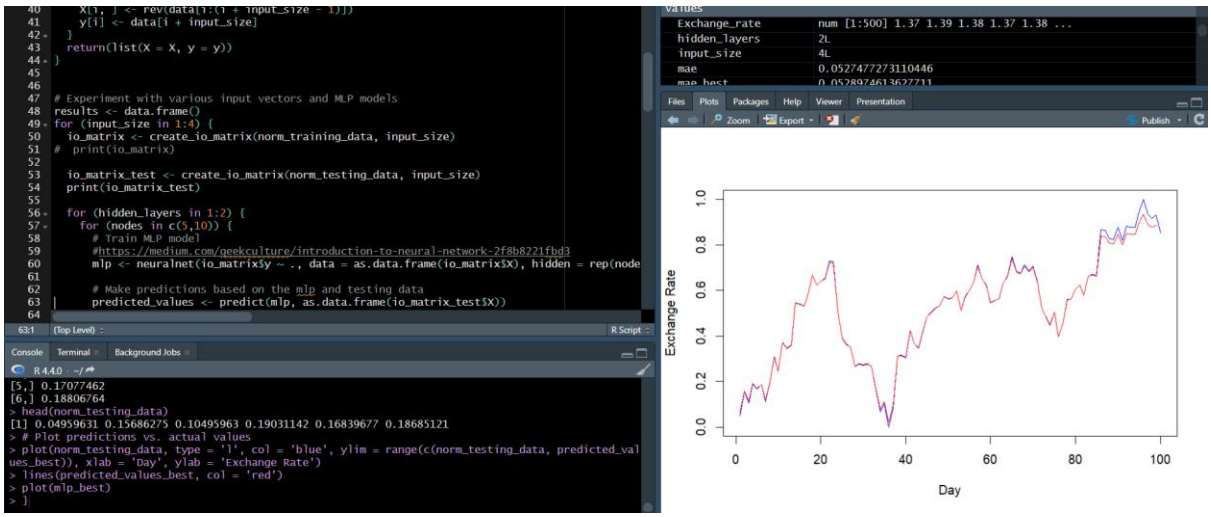
With this 4,8 nodes combination, the Actual and Predicted values are quite compatible.

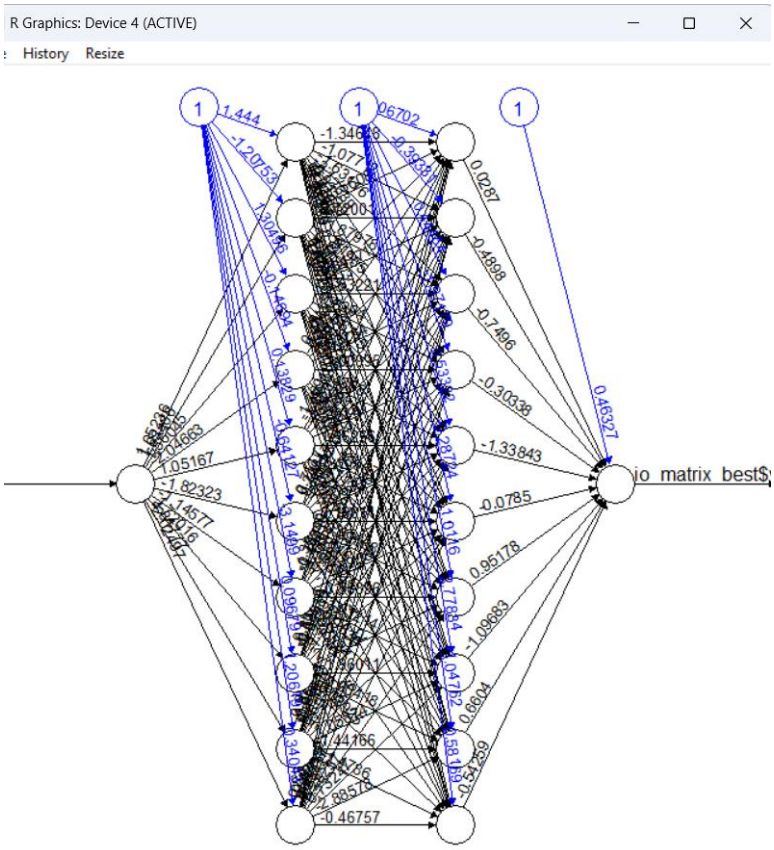




Nodes(5,10)

The nodes (5,10) gives the best result out of all the sets tested. The Actuals and Predicted values are almost identical giving the best result.





Using the four input vectors, 1-2 hidden layers and Nodes (5,10), multiples MLP models are generated. The MLP models are evaluated using the standard statistical indices (RMSE, MAE, MAPE and sMAPE – symmetric MAPE) which are calculated for each MLP model and a comparison table is created for same.

The four statistical indices - RMSE, MAE, MAPE, and sMAPE are:

Root Mean Square Error (RMSE):

RMSE measures the average magnitude of the errors between predicted values and actual values.

It calculates the square root of the average of the squared differences between predicted and actual values.

RMSE is sensitive to outliers, as larger errors have a greater impact on the overall value.

Lower RMSE values indicate better model performance, with a value of 0 representing perfect predictions.

RMSE is expressed in the same units as the variable being forecasted.

Mean Absolute Error (MAE):

MAE measures the average absolute difference between predicted and actual values.

It calculates the average of the absolute differences between predicted and actual values.

MAE is less sensitive to outliers compared to RMSE, as it does not square the differences.

Lower MAE values indicate better model performance, with a value of 0 representing perfect predictions.

MAE is expressed in the same units as the variable being forecasted.

Mean Absolute Percentage Error (MAPE):

MAPE measures the average percentage difference between predicted and actual values, relative to the actual values.

It calculates the average of the absolute percentage differences between predicted and actual values.

MAPE provides a percentage-based measure of forecasting accuracy, making it easier to interpret across different datasets.

However, MAPE can be problematic when actual values are close to zero, as it may result in undefined or extremely large values.

Lower MAPE values indicate better model performance, with a value of 0 representing perfect predictions.

Symmetric Mean Absolute Percentage Error (sMAPE):

sMAPE is like MAPE but calculates the percentage difference symmetrically around zero.

It calculates the average of the absolute percentage differences between predicted and actual values, adjusted for the average of the predicted and actual values.

sMAPE addresses the issue of asymmetry in MAPE, especially when actual values are close to zero.

Like MAPE, lower sMAPE values indicate better model performance, with a value of 0 representing perfect predictions.

Using the four input vectors, 1-2 hidden layers and Nodes (5,10), multiples MLP models are generated. The Comparison table is created for the various combination of input vectors, hidden layer, and the neural nodes. For all these combinations the RMSE, MAE, MAPE and sMAPE values are computed and compared.

The efficiency of two best NN structures involves balancing computational resources, training time, model complexity, and performance considerations to determine the most suitable architecture for the specific problem and constraints**.**

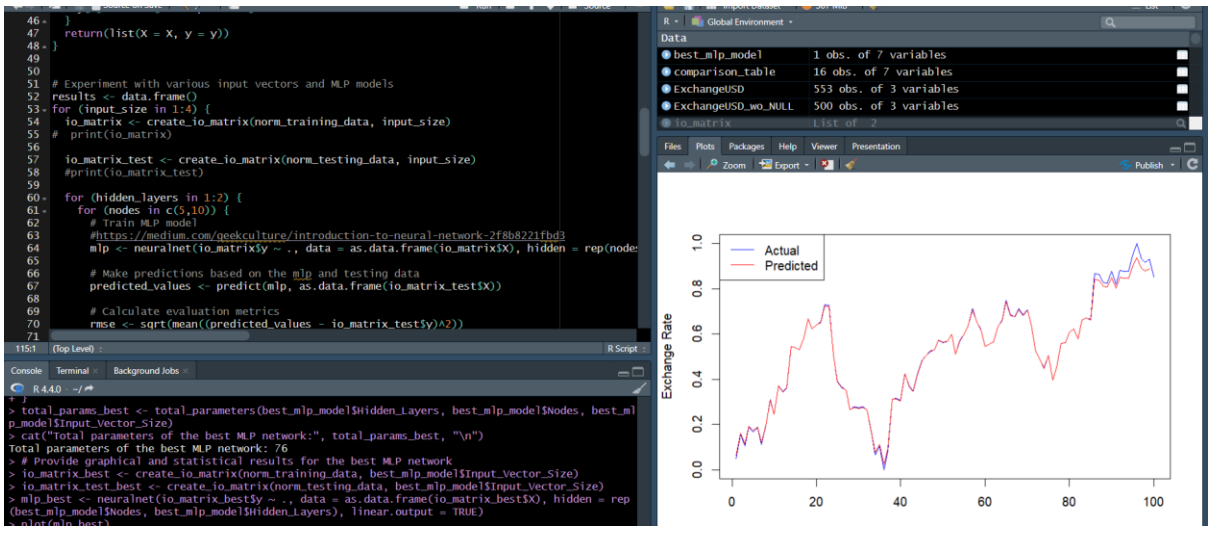
As mentioned in the explanation of the Statistical indices, the lower the value of the index, the performance of the MLP model is better.

Input_Vector_Size	Hidden_Layers	Nodes	RMSE	MAE	MAPE	sMAPE
1	1	5	0.07080683	0.05336310	Inf	0.1796351
2	1	5	0.07045889	0.05243864	Inf	0.1632779
3	1	5	0.07111208	0.05290006	Inf	0.1626064
4	1	5	0.07162981	0.05383865	Inf	0.1713482
1	2	5	0.07018800	0.05253021	Inf	0.1666863
2	2	5	0.07089337	0.05306015	Inf	0.1600567
3	2	5	0.07094644	0.05292745	Inf	0.1617191
4	2	5	0.07106885	0.05375146	Inf	0.1721010
1	1	10	0.07052828	0.05330591	Inf	0.1789515
2	1	10	0.07126329	0.05447273	Inf	0.1811266
3	1	10	0.07135411	0.05350502	Inf	0.1686799
4	1	10	0.07038550	0.05306375	Inf	0.1589436
1	2	10	0.07064612	0.05322759	Inf	0.1824399
2	2	10	0.07108358	0.05337277	Inf	0.1655858
3	2	10	0.07102084	0.05358939	Inf	0.1662018
4	2	10	0.07105925	0.05281364	Inf	0.1571504

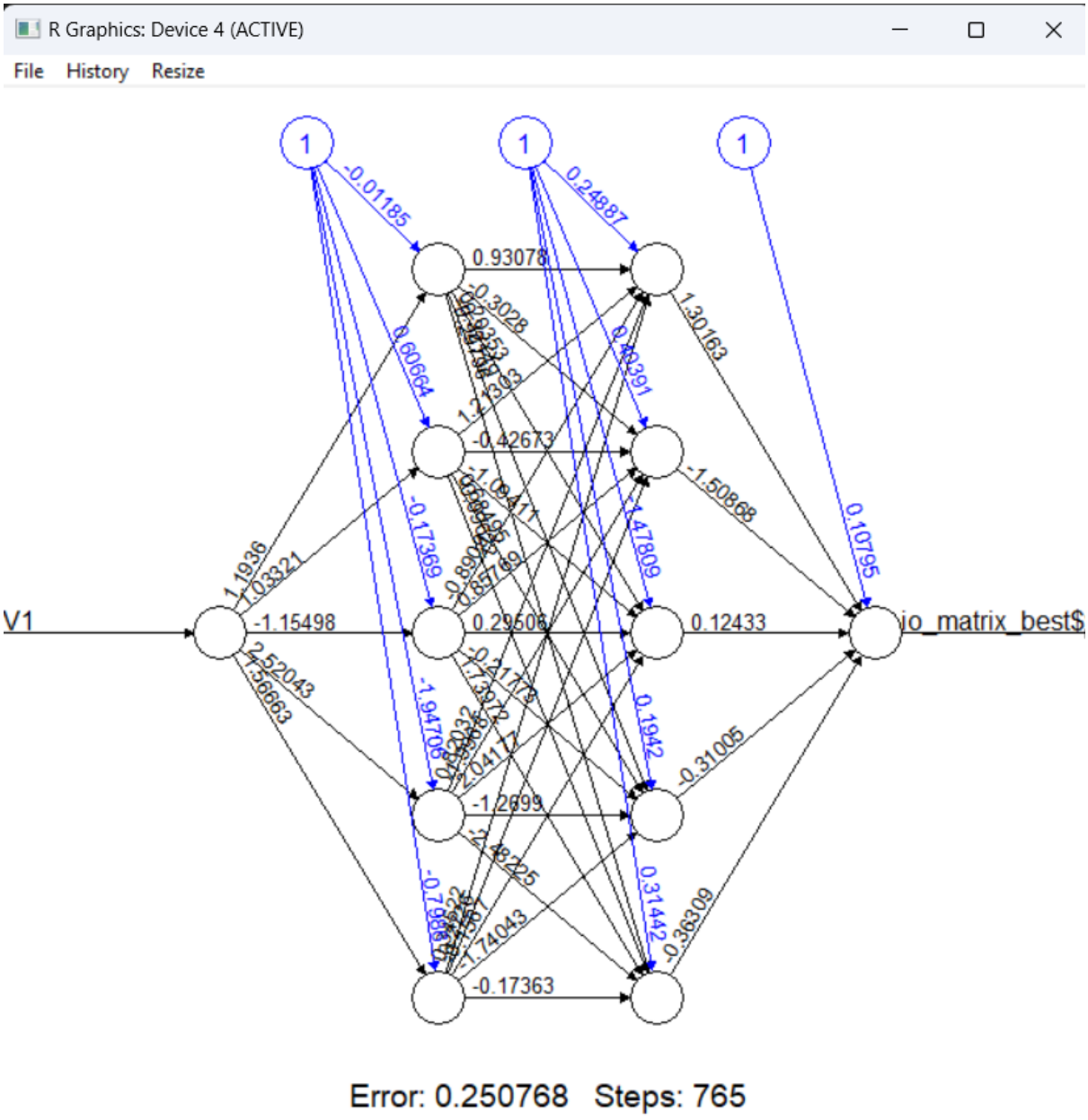
From above comparison table, the combination having least of RMSE is selected. This give the best MLP Model.

```
  Input_Vector_Size Hidden_Layers Nodes     RMSE        MAE MAPE     sMAPE
5                 1             2     5 0.070188 0.05253021  Inf 0.1666863
```

```
Total parameters of the best MLP network: 76
```



Best Mlp Model based on the Statistical Indices.



Input_Vector_	Hidden_Layer	Nodes	RMSE	MAE	MAPE	sMAPE
1	2	5	0.070188	0.05253021	Inf	0.1666863
4	1	10	0.0703855	0.05306375	Inf	0.1589436

The 2nd best MLP model based on the RMSE index is with 4 Input Vectors, 1 Hidden Layers and 10 nodes.

Comparing the two NN Structures, the 2nd best will need more computational resources compared to the 1st NN Structure which is a simple MLP Model.

Looking at the data set, the complexity is low as we are only considering the time-lagged inputs and no other input variable affects the outcome, the best will be 1st NN Structure.

Bibliography:

Ryzhkov, E. (2020). *5 Stages of Data Preprocessing for K-means clustering*. [online] Medium. Available at: https://medium.com/@evgen.ryzhkov/5-stages-of-data-preprocessing-for-k-means-clustering-b755426f9932.

Franklin, S.J. (2019). *Effect of outliers on K-Means algorithm using Python*. [online] Medium. Available at: https://medium.com/analytics-vidhya/effect-of-outliers-on-k-means-algorithm-using-python-7ba85821ea23.

Frost, J. (2019). *5 Ways to Find Outliers in Your Data*. [online] Statistics By Jim. Available at: https://statisticsbyjim.com/basics/outliers/.

search.r-project.org. (n.d.). *R: Calinski-Harabasz index*. [online] Available at: https://search.r-project.org/CRAN/refmans/fpc/html/calinhara.html [Accessed 30 Apr. 2024].

GeeksforGeeks. (2020). *Interquartile Range to Detect Outliers in Data*. [online] Available at: https://www.geeksforgeeks.org/interquartile-range-to-detect-outliers-in-data/.

GeeksforGeeks. (2017). *K means Clustering - Introduction*. [online] Available at: https://www.geeksforgeeks.org/k-means-clustering-introduction/?ref=header_search.

Stack Overflow. (n.d.). *Calculate cumulative sum (cumsum) by group*. [online] Available at: https://stackoverflow.com/questions/16850207/calculate-cumulative-sum-cumsum-by-group [Accessed 30 Apr. 2024].

Dautel, A.J., Härdle, W.K., Lessmann, S. and Seow, H.-V. (2020). Forex exchange rate forecasting using deep recurrent neural networks. *Digital Finance*. doi:https://doi.org/10.1007/s42521-020-00019-x.

Patil, N., Masih, S., Jeneya Rumao and Veena Gaurea (2021). Predict Foreign Currency Exchange Rates Using Machine Learning. *Advances in intelligent systems and computing*, pp.223–232. doi:https://doi.org/10.1007/978-981-16-4641-6_19.

www.computer.org. (n.d.). *CSDL | IEEE Computer Society*. [online] Available at: https://www.computer.org/csdl/proceedings-article/iisa/2023/10345946/1STKnvnSJQk. [Accessed 30 Apr. 2024].

Ayitey Junior, M., Appiahene, P., Appiah, O. and Bombie, C.N. (2023). Forex market forecasting using machine learning: Systematic Literature Review and meta-analysis. *Journal of Big Data*, 10(1). doi:https://doi.org/10.1186/s40537-022-00676-2.

Khoa, B.T. and Huynh, T.T. (2022). Predicting Exchange Rate under UIRP Framework with Support Vector Regression. *Emerging Science Journal*, 6(3), pp.619–630. doi:https://doi.org/10.28991/esj-2022-06-03-014.

Vandeput, N. (2021). *Forecast KPI: RMSE, MAE, MAPE & Bias*. [online] Medium. Available at: https://towardsdatascience.com/forecast-kpi-rmse-mae-mape-bias-cdc5703d242d.

www.ibm.com. (2023). *Forecasting statistical details*. [online] Available at: https://www.ibm.com/docs/en/cognos-analytics/11.1.0?topic=forecasting-statistical-details.

BabyPips.com. (2015). *Fundamental Analysis*. [online] Available at: https://www.babypips.com/forexpedia/fundamental-analysis#:~:text=Fundamental%20analysis%20involves%20studying%20economic%20trends%20and%20geopolitical [Accessed 30 Apr. 2024].

Appendix:

Partitioning Clustering:

```
install.packages("readxl")

install.packages("factoextra")

install.packages("fpc")

library(cluster)

library(NbClust)

library(factoextra)

library(fpc)

library(readxl)

winedataset <- read_xlsx("C:/Users/omsad/OneDrive/Desktop/University of Westminster/Year 2/Machine Learning and Data Mining/CW/Part 1/Whitewine_v6.xlsx")

chosenset <- winedataset[, -ncol(winedataset)]

 str(chosenset)

scaledata <- scale(chosenset)

 summary(scaledata)

 print(scaledata)

Scaledata_df <- as.data.frame(scaledata)

 str(Scaledata_df)


remove_outliers_iqr <- function(data) {

   Q1 <- apply(data, 2, quantile, probs = 0.25)

   Q3 <- apply(data, 2, quantile, probs = 0.75)

   IQR <- Q3 - Q1

   lower_bound <- Q1 - 1.5 * IQR

   upper_bound <- Q3 + 1.5 * IQR

   outliers <- apply(data, 1, function(x) any(x < lower_bound | x > upper_bound))

   return(data[!outliers, , drop = FALSE])

}


cleaned_data <- remove_outliers_iqr(Scaledata_df)

str(cleaned_data)
```

```
# NbClust Method:
nbClusters <- NbClust(cleaned_data, distance = "euclidean", min.nc = 2, max.nc = 15, method = "kmeans", index = "all")
table(nbClusters$Best.n[1,])
barplot(table(nbClusters$Best.n[1,]),
     xlab="Number of Clusters",
     ylab="Number of Criteria",
     main="Number of Center Clusters")


# Elbow Method:
fviz_nbclust(cleaned_data, kmeans, method = 'wss')

# Gap Statistics Method:
fviz_nbclust(cleaned_data, kmeans, method = 'gap_stat')

# Silhouette Method:
fviz_nbclust(cleaned_data, kmeans, method = 'silhouette')

# kmeans clustering
kmeans <- kmeans(cleaned_data,3)
kmeans
fviz_cluster(kmeans, data = cleaned_data )
# cluster mean:
kmeans$centers

# Cluster Vector:
kmeans$cluster

# Calculation of WSS and BSS:
```

```
WSS <- kmeans$tot.withinss
BSS <- kmeans$betweenss

TSS <- sum(WSS) + sum(BSS)
BSS_TSS_ratio <- BSS / TSS

# silhouette
sil <- silhouette(kmeans$cluster, dist(cleaned_data))
fviz_silhouette(sil)

#REFRESH WORKSPACE FOR PCA---

# PCA

pca_wine <- prcomp(cleaned_data,center= TRUE, scale. = TRUE)
summary(pca_wine)

eigenvalues <- pca_wine$sdev^2
eigenvectors <- pca_wine$rotation



# cumulative score by using function 'cumsum'
cumulative_score <- cumsum(pca_wine$sdev^2 / sum(pca_wine$sdev^2))
summary(cumulative_score)

num_components <- which.max(cumulative_score > 0.85)
summary(num_components)

#Transformed Data
wine_transform <- as.data.frame(-pca_wine$x[,1:num_components])
head(wine_transform)
```

```
# pca NbClust Method(Takes time to load):

pca_nbClusters <- NbClust(wine_transform, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans", index = "all")

table(nbClusters$Best.n[1,])

barplot(table(nbClusters$Best.n[1,]),

     xlab="Number of Clusters",

     ylab="Number of Criteria",

     main="Number of Center Clusters")


## pca Alternative Method:

fviz_nbclust(wine_transform, kmeans, method = 'wss')


# pca Gap Statistics Method:

fviz_nbclust(wine_transform, kmeans, method = 'gap_stat')


# pca Silhouette Method:

fviz_nbclust(wine_transform, kmeans, method = 'silhouette')


# pca k-means

k= 2

kmeans_pca <- kmeans(wine_transform, centers = k, nstart = 10)

summary(kmeans_pca)


# cluster mean:

kmeans_pca$centers


# Cluster Vector:

kmeans_pca$cluster
```

```
# Calculation of WSS and BSS for pca:
WSS_pca <- kmeans_pca$tot.withinss
BSS_pca <- kmeans_pca$betweenss
# BSS/TSS Ratio for pca:
TSS_pca <- sum(WSS_pca) + sum(BSS_pca)
BSS_TSS_ratio_pca <- BSS_pca / TSS_pca



# silhouette fro pca:
sil <- silhouette(kmeans$cluster, dist(wine_transform))
fviz_silhouette(sil)

# Calinski-Harabasz Index for pca:
CalH_index <- calinhara(wine_transform, kmeans_pca$cluster)
```

Financial Forecasting:

```
library(zoo)
library(lubridate)
library(mgcv)
library(rugarch)
library(ggplot2)
library(quantmod)
library (PerformanceAnalytics)
library(rugarch)
library (FinTS)
library(forecast)
library(strucchange)
library (TSA)
library(tseries)
library(timeSeries)
library(xts)
library(pastecs)
library(tidyr)
library(dplyr)
library(splines)
library(tidyverse)
rm(list=ls())
library (FinTS)
library(rugarch)
library(tseries)
library(dynlm)
library(vars)
library(nlWaldTest)
library(broom)
```

```
library(readxl)
# For Multilayer Perceptrons (MLP)
library(nnfor)
library(neuralnet)
library(MLmetrics)
library(Metrics)
library(neuralnet)

# Load dataset
ExchangeUSD <-
read.csv("C:/Users/ramsa/OneDrive/Desktop/Om/MachineLearning/ExchangeUSD.csv")

# Cleaning Data - Remove NULL value records
ExchangeUSD_wo_NULL <- na.omit(ExchangeUSD)

# Use only the 3rd column from the file
Exchange_rate <- ExchangeUSD_wo_NULL$USD.EUR

# Split data - the first 400 of them have to be used as training data,
# while the remaining ones as testing set.
training_data <- Exchange_rate[1:400]
testing_data <- Exchange_rate[401:500]

#         Each one of these I/O matrices needs to be normalised,
# Function to normalize data
normalize <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

# Normalize training and testing data
```

```
norm_training_data <- normalize(training_data)
norm_testing_data <- normalize(testing_data)

head (norm_training_data)
head (norm_testing_data)

# construct an input/output matrix (I/O) for the MLP training/testing
# Function to create input/output matrix for MLP training/testing
create_io_matrix <- function(data, input_size) {
  X <- matrix(nrow = length(data) - input_size, ncol = input_size)
  y <- vector(length = length(data) - input_size)
  for (i in 1:(length(data) - input_size)) {
    X[i, ] <- rev(data[i:(i + input_size - 1)])
    y[i] <- data[i + input_size]
  }
  return(list(X = X, y = y))
}


# Experiment with various input vectors and MLP models
results <- data.frame()
for (input_size in 1:4) {
  io_matrix <- create_io_matrix(norm_training_data, input_size)
  io_matrix_test <- create_io_matrix(norm_testing_data, input_size)


  for (hidden_layers in 1:2) {
    for (nodes in c(5,10)) {
      # Train MLP model
```

```
    mlp <- neuralnet(io_matrix$y ~ ., data = as.data.frame(io_matrix$X), hidden = rep(nodes, hidden_layers), linear.output = TRUE)

    # Make predictions based on the mlp and testing data
    predicted_values <- predict(mlp, as.data.frame(io_matrix_test$X))

    # Calculate evaluation metrics
    rmse <- sqrt(mean((predicted_values - io_matrix_test$y)^2))
    mae <- mae(io_matrix_test$y, predicted_values)
    mape <- mape(io_matrix_test$y, predicted_values)
    smape <- smape(io_matrix_test$y, predicted_values)

    # Store results
    results <- rbind(results, data.frame(Input_Vector_Size = input_size, Hidden_Layers = hidden_layers, Nodes = nodes, RMSE = rmse, MAE = mae, MAPE = mape, sMAPE = smape))
   }
  }
}

# Display results
print(results)

# Get the results in comparison table to further narrow down best result
comparison_table <- aggregate(cbind(RMSE, MAE, MAPE, sMAPE) ~ Input_Vector_Size + Hidden_Layers + Nodes, data = results, FUN = mean)
print(comparison_table)

# The mlp model with least error is the best model
best_mlp_model <- comparison_table[which.min(comparison_table$RMSE), ]
print(best_mlp_model)
```

```
# Discuss the efficiency of the best MLP network

total_parameters <- function(hidden_layers, nodes, input_size) {

  total_params <- (input_size + 1) * nodes # Number of parameters in the input layer

  total_params <- total_params + (nodes + 1) * nodes * hidden_layers # Number of
parameters in hidden layers

  total_params <- total_params + (nodes + 1) # Number of parameters in the output layer

  return(total_params)

}


total_params_best <- total_parameters(best_mlp_model$Hidden_Layers,
best_mlp_model$Nodes, best_mlp_model$Input_Vector_Size)


cat("Total parameters of the best MLP network:", total_params_best, "\n")


# Provide graphical and statistical results for the best MLP network

io_matrix_best <- create_io_matrix(norm_training_data,
best_mlp_model$Input_Vector_Size)

io_matrix_test_best <- create_io_matrix(norm_testing_data,
best_mlp_model$Input_Vector_Size)


mlp_best <- neuralnet(io_matrix_best$y ~ ., data = as.data.frame(io_matrix_best$X), hidden
= rep(best_mlp_model$Nodes, best_mlp_model$Hidden_Layers), linear.output = TRUE)

plot(mlp_best)

predicted_values_best <- predict(mlp_best, as.data.frame(io_matrix_test_best$X))


# Plot predictions vs. actual values

plot(norm_testing_data, type = 'l', col = 'blue', ylim = range(c(norm_testing_data,
predicted_values_best)), xlab = 'Day', ylab = 'Exchange Rate')

lines(predicted_values_best, col = 'red')

legend('topleft', legend = c('Actual', 'Predicted'), col = c('blue', 'red'), lty = 1)

plot(mlp_best)
```

```
# Calculate evaluation metrics for the best MLP network
rmse_best <- sqrt(mean((predicted_values_best - io_matrix_test_best$y)^2))
mae_best <- mae(io_matrix_test_best$y, predicted_values_best)
mape_best <- mape(io_matrix_test_best$y, predicted_values_best)
smape_best <- smape(io_matrix_test_best$y, predicted_values_best)

cat("Evaluation metrics for the best MLP network:\n")
cat("RMSE:", rmse_best, "\n")
cat("MAE:", mae_best, "\n")
cat("MAPE:", mape_best, "\n")
cat("sMAPE:", smape_best, "\n")

strength_min <- min(training_data)
strength_max <- max(training_data)



unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}

strength_pred <- unnormalize(predicted_values_best, strength_min, strength_max)
head(strength_pred)  # this is NN’s output denormalized to original ranges
head(testing_data)

plot(testing_data, type = 'l', col = 'blue', ylim = range(c(testing_data, strength_pred)), xlab =
'Day', ylab = 'Exchange Rate')
lines(strength_pred, col = 'red')
legend('topleft', legend = c('Actual', 'Predicted'), col = c('blue', 'red'), lty = 1)
```

```
#compared to original data, normalized data give better comparison.
head(predicted_values_best)  # this is NN's output denormalized to original ranges
head(norm_testing_data)
```