UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

Department of Computing

Machine Learning and Data Mining

5DATA001C.2

Module	: 5DATA001C.2 Machine Learning and Data Mining
Module Leader	: Mr. Nipuna Senanayake
Date of submission	: 2024/05/02
Student ID	: 20221846 /w1985618
Student First Name	: Pawan
Student Surname	: De Silva
Tutorial Group	: SE - D

Table of Contents

1. Partitioning Clustering Part

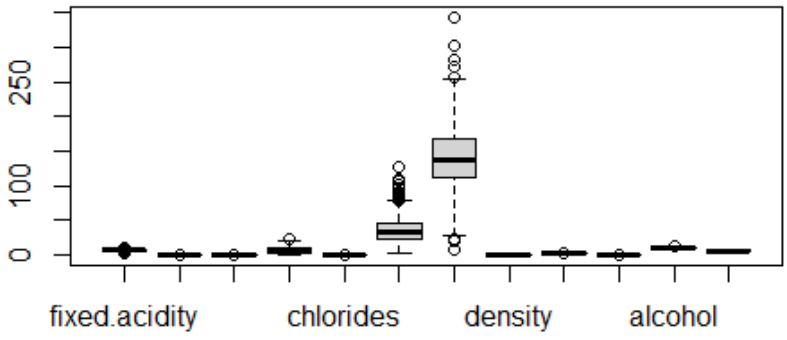
1st Subtask Objectives

1.1 Outlier Detection and Removal and Scaling

During the process of preprocessing data, it's essential to locate and remove outliers. This is because outliers can have a significant impact on the clustering process, causing clusters to become distorted or wrapped. One way to detect outliers is by using a box plot. The data points that fall far outside the whiskers can be visually inspected to identify and eliminate outliers. By removing these outliers, the accuracy and dependability of the clustering can be improved. Here's an example code fragment that illustrates this process:

```
#import whitewine data set
whitewhine_data <- read.csv("Whitewine_v6.csv")

#calculate  total number of missing values
sum(is.na(whitewhine_data))
#Provide a summary of logicsal values which indicate the missing values or not
summary(is.na(whitewhine_data))

#store attributes from column 1 to 11
numarical_variables <- whitewhine_data[,-12]
#quality type of the
quality_variable <- whitewhine_data$quality

#create a empty vector for store found outliers during the data analysis
Outliers <- c()

#graphical summary of the distribtion
boxplot(numarical_variables)
```

To visualize the distribution of numerical variables in the dataset 'numerical_variables', the boxplot() method can be used. This figure displays the median, quartiles, and outliers of the data. The whiskers extend to the extreme data points within 1.5 times the interquartile range (IQR), and separate points are shown for each outlier.

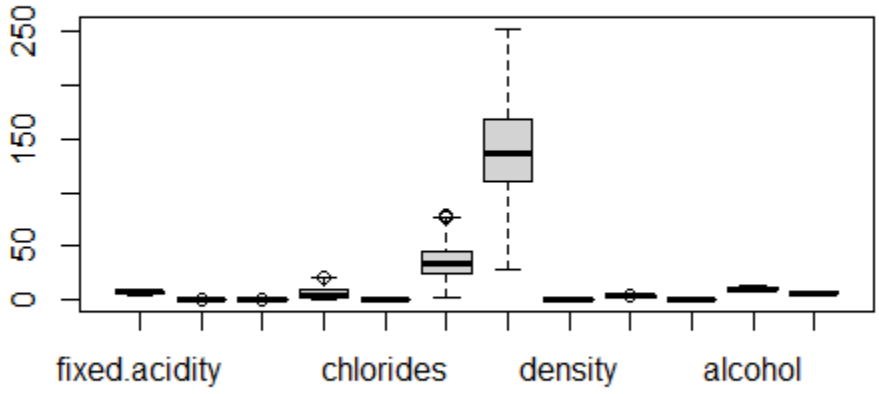


I utilize a for loop in the following code to iterate through the data set, find the outliers, and eliminate them from the original data set.

```
for(i in 1:11){
  column <- numarical_variables[, i]
  logical_vector <- column%in% boxplot.stats(column)$out
  current_column_out <- which(logical_vector == TRUE)
  Outliers <- c(Outliers,current_column_out)
}


eleminate <- unique(Outliers) #containing unique outliers
eleminate <- sort(eleminate) #ascending order
removed_data <- whitewhine_data[-eleminate,] #remove outliers
```

After removing any outliers, we can use the boxplot method again to determine if there are any other unusual data points.

```
#after remove outliers from original data set
boxplot(removed_data[,-12])
```

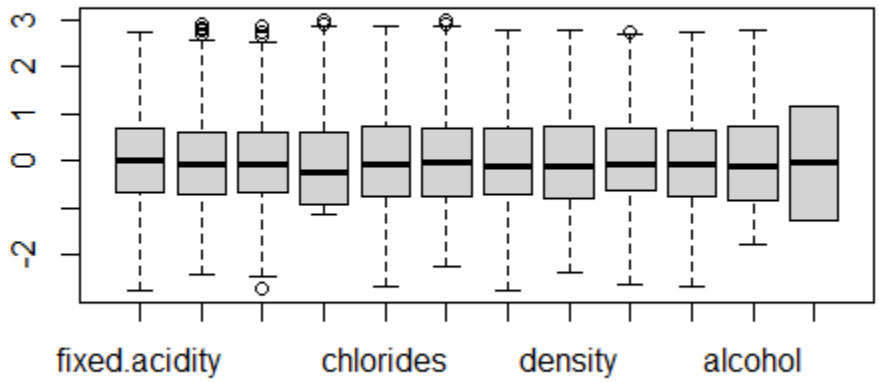


After removing the outliers, the dataset increased with 457 new data points, lowering the average from 2700 to 2243.

The data was separated into numerical and categorical classes for preprocessing. The next step is to scale the data.

Scaling is a process that standardizes feature ranges and helps to avoid biased clustering results that may arise from using various units or scales. Moreover, it assists k-means in converging more quickly by preventing traits of greater magnitude from dominating the clustering process, which ensures a more precise and efficient clustering procedure.

```
#Scaling process for standardize the rage
scaled_removed_data <- data.frame(scale(removed_data[,-12]))

#after scaling
boxplot(scaled_removed_data)
```



1.2 Get the number of clusters using automated tools.

1.2.1 NbClust Method

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

Based on the "scaled_removed_data" dataset, the "NbClust()" function from the R NbClust package was used to obtain the results.

The “euclidean” distance metric is the one specified by the distance parameter, which is used in this instance. “kmeans” is the clustering algorithm that is specified by the method parameter.

The range of clusters to be taken into consideration is specified by the min.nc and max.nc parameters, which are set to 2 and 10, respectively. The index parameter, which is set to "all" to calculate all possible indices, lastly indicates which cluster validity indices to compute.

A list of the best number of clusters, the chosen index value, and the clustering outcomes are all provided by NbClust() function. According to its suggestion, two clusters are ideal for applying the majority rule.

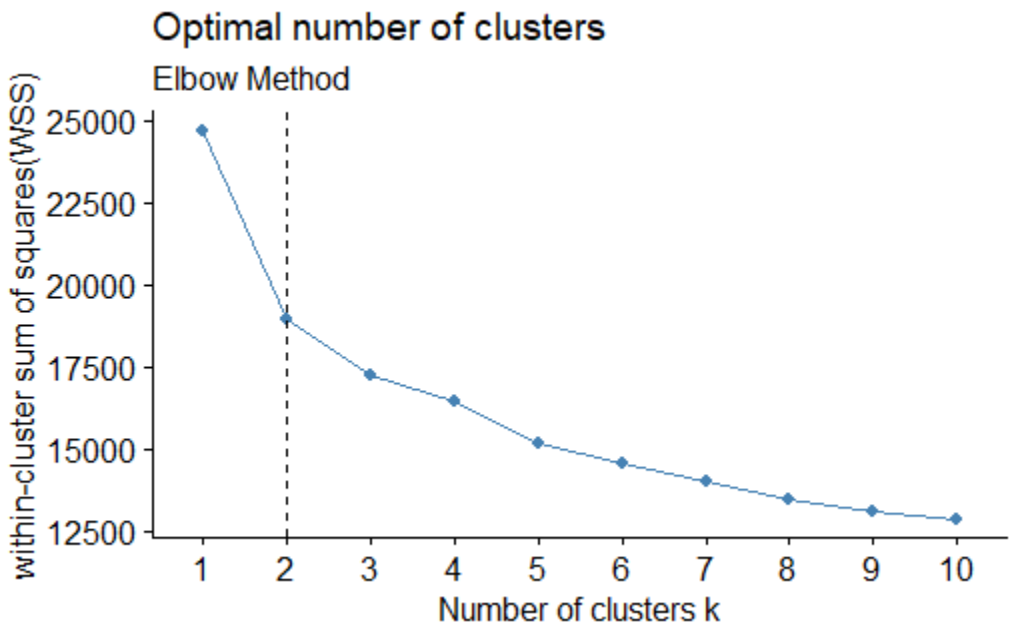
```
*** : The Hubert index is a graphical method of determining the number of clusters.
               In the plot of Hubert index, we seek a significant knee that corresponds to a
               significant increase of the value of the measure i.e the significant peak in Hubert
               index 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
* 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


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

1.2.2 Elbow Method

The elbow method technique is a useful strategy to determine the number of clusters in a given data set. This technique uses the Within Cluster Sum of Squares (WSS) approach to compute the qualified number of clusters automatically. To do this, the WSS score is calculated ten times, and the value is saved after each iteration. The WSS score will decrease as the number of clusters increases, and it will increase as the number of clusters decreases. By identifying the formation point of the elbow shape with the appropriate WSS values and number of clusters, we can determine the optimal number of clusters using the elbow method.

The k-means clustering algorithm in the following code uses the scaled_removed_data dataset to produce a plot of the within-cluster sum of squares for varying numbers of clusters. The ideal number of clusters may be determined by looking at the vertical line t x = 2, which represents the elbow point.

```
#Elbow method
set.seed(123)
fviz_nbclust(scaled_removed_data, kmeans, method = "wss")+
  geom_vline(xintercept = 2, linetype = 2)+
  labs(subtitle = "Elbow Method", x = "Number of clusters k",
       y= "within-cluster sum of squares(WSS)")
```



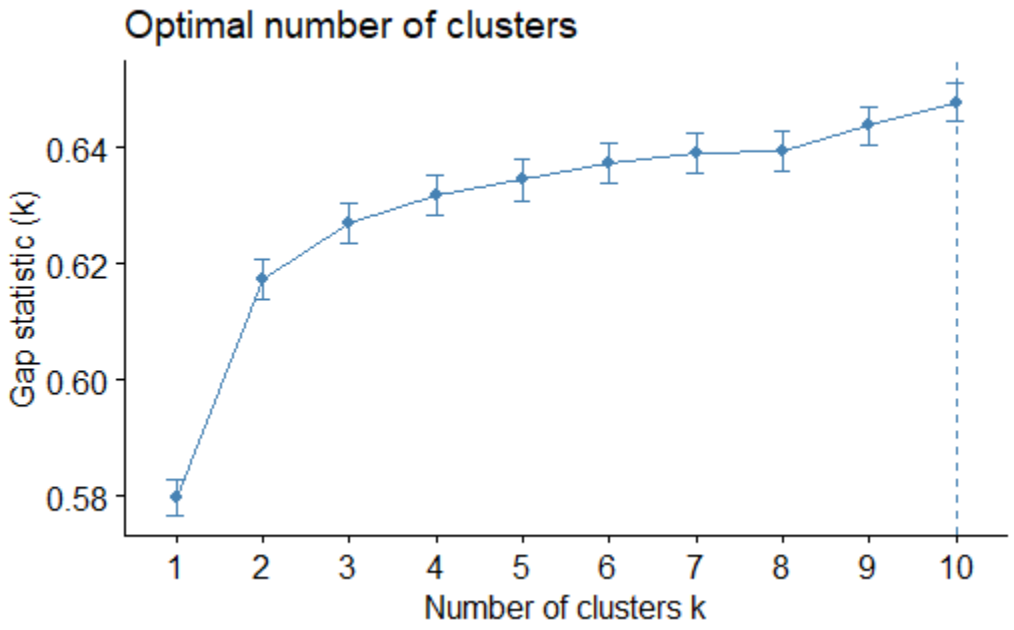
The Elbow method provides the best number of clusters as 2.

1.2.3 Gap Statistics Method

A statistical methodology known as the gap statistic technique may be used to find the ideal number of clusters in a dataset to utilize with a clustering algorithm. The technique compares the expected value under a null reference distribution to the total within-cluster variance, which is determined as the sum of the squared distance between data points and their cluster centre.

Use k-means clustering algorithm with nstart and B respectively 25 random starts and 50 bootstrap replicates to calculate the gap with different numbers of clusters using clusGap(). Visualise the gap statistic with the fviz_gap_stat() function.

```
#Gap-static automated tool
set.seed(123)
gap_stat <- clusGap(scaled_removed_data, kmeans, K.max = 10,
                    nstart = 25, B =50)
fviz_gap_stat(gap_stat)
```

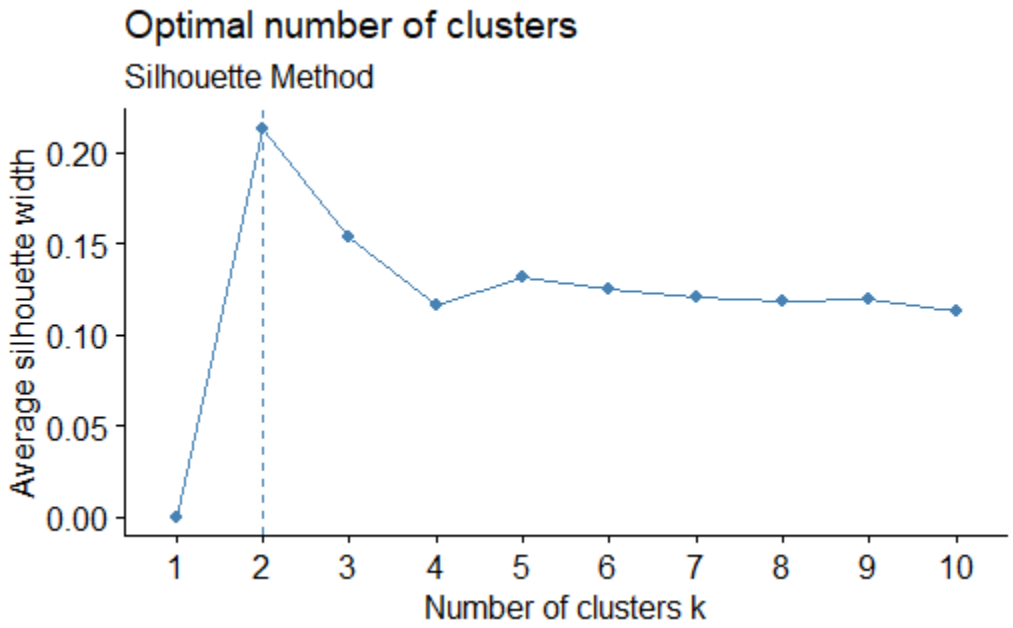


The Gap Statistic method provides the best number of clusters as 10 clusters.

1.2.4 Silhouette Method

The silhouette approach is commonly used to determine the optimal number of clusters in a dataset. This approach involves calculating the silhouette coefficient for different values of k (the number of clusters). The k value with the highest mean silhouette coefficient is chosen as the best number of clusters. This method is often used in combination with other clustering validation techniques, such as the elbow method or gap statistic, to determine the best number of clusters for a given dataset.

```
#silhouette Method
fviz_nbclust(scaled_removed_data, kmeans, method = "silhouette")+
  labs(subtitle = "Silhouette Method")
```



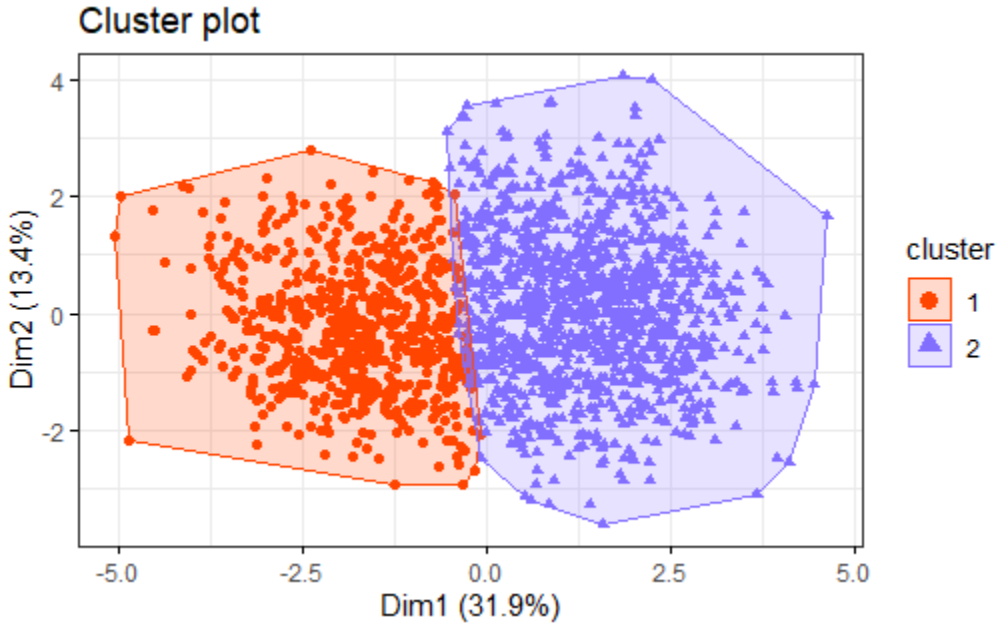
The silhouette method provides the bets number of clusters as 2 clusters.

1.3 K-means Clustering

After determining the optimal number of clusters, the clustering process is performed using those clusters.

The scaled_removed_data dataset has k = 2 clusters, and the k-means clustering algorithm is applied to it using the kmeans() function, which stores the outcomes in the kmean_2 object.

```r
#k-means
#k-means clustering using 2 clusters based on sacaled data
kmean_2 <- kmeans(scaled_removed_data,2)

#visualization of the k-means clustering result
fviz_cluster(kmean_2, data = scaled_removed_data,
             palette= c("#ff4500", "#836fff"),
             geom = "point", ellipse.type = "convex", ggtheme = theme_bw()
             )

kmean_2
```



```
K-means clustering with 2 clusters of sizes 910, 1333

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar  chlorides free.sulfur.dioxide
1     0.1619461       0.08285808  0.03182782      0.8532817  0.6051817           0.5951815
2    -0.1105558      -0.05656478 -0.02172792     -0.5825104 -0.4131398          -0.4063130
  total.sulfur.dioxide    density          pH   sulphates    alcohol
1            0.7959548  0.9729058 -0.12934941  0.12025514 -0.8078073
2           -0.5433750 -0.6641742  0.08830305 -0.08209466  0.5514663

Clustering vector:
   1    5    7    8   13   14   16   17   18   19   21   22   23   24   25   31   32   33   34
   1    1    2    2    2    1    2    2    1    1    1    1    2    2    1    2    2    2    2
  35   36   38   39   41   42   43   44   47   48   49   52   53   55   56   57   58   59   60
   2    1    1    1    2    2    2    1    2    2    2    2    1    2    2    2    2    2    2
  61   68   69   72   80   83   84   85   86   89   91   92   93   94   95   96   97  100  101
   2    1    2    2    2    1    2    2    2    2    2    1    2    2    1    1    2    1    2
 102  103  104  105  106  107  108  109  110  111  115  116  118  119  120  121  122  125  126
   1    1    1    1    1    1    1    1    2    2    2    2    1    2    2    1    1    1    1
 127  128  130  131  132  134  135  136  137  139  140  141  142  143  144  145  146  148  149
   1    1    1    1    2    2    1    1    2    1    2    1    2    2    1    2    1    2    1
 151  153  154  155  156  157  158  159  160  161  162  163  164  166  167  168  169  170  171
   1    2    1    1    1    1    1    2    1    1    1    1    2    1    1    2    1    1    1
 172  173  174  175  176  177  178  179  180  181  182  183  184  186  187  188  190  192  193
   1    1    2    2    2    1    1    1    1    1    2    2    1    1    2    1    1    1    1
 194  195  196  197  198  199  203  204  208  209  210  213  214  215  221  222  223  224  225
   1    1    1    1    1    2    2    1    1    1    2    2    2    2    1    2    1    2    2
 226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  244  245
   2    2    2    2    2    1    1    1    1    1    2    2    1    1    2    1    2    2    2
 246  247  250  251  252  254  255  256  258  259  260  261  262  264  265  266  268  269  270
   2    1    2    2    2    2    1    1    1    1    1    2    1    1    1    2    2    1    1
 271  272  273  274  275  277  279  280  282  283  284  285  286  288  289  290  291  293  294
   1    1    2    1    2    1    1    1    2    2    1    1    2    1    1    1    1    1    1
 295  296  297  298  300  302  303  305  307  308  309  310  311  312  313  314  315  316  319
   1    1    1    2    2    1    1    2    2    1    1    1    2    1    1    2    2    2    2


Within cluster sum of squares by cluster:
[1]  7417.076 11508.146
 (between_SS / total_SS =  23.3 %)


Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
```

```r
centers <- kmean_2$centers #cluster centers
bss <- kmean_2$betweenss #between cluster sum of squares
tss <- kmean_2$totss #total sum of squares
wss <- kmean_2$withinss # within cluster sum of squares
```

Here, the BSS, WSS, and TSS values for clustering attempts of k =2 were calculated.

The bss variable represents the between-cluster sum of squares, which quantifies the dispersion of the data points around the cluster centroids.

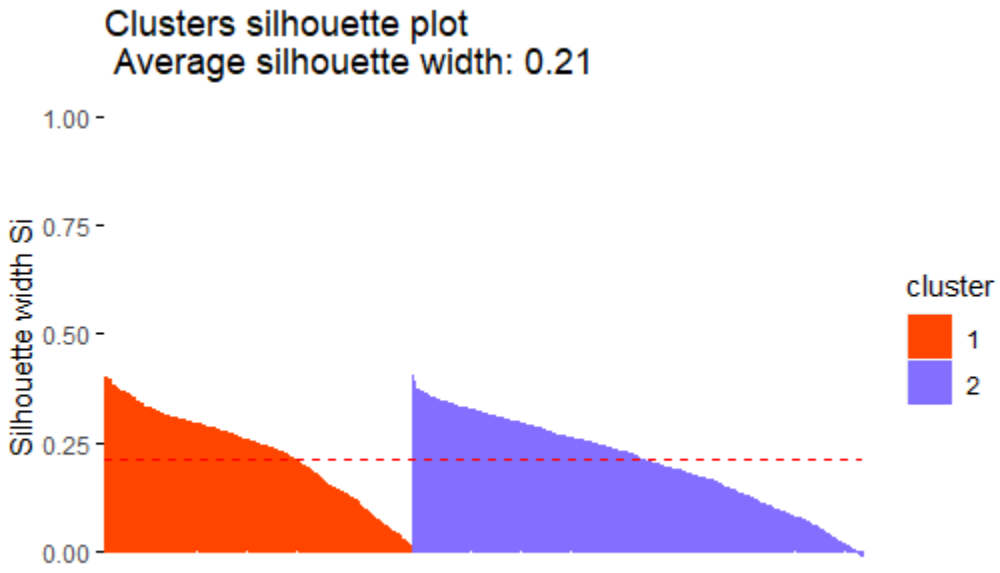
The wss variable represents the within-cluster sum of squares, which quantifies the amount of variance inside each cluster.

The tss variable represents the total sum of squares, which quantifies the total amount of variation in the data.

1.4 Silhouette plot analysis of the kmeans attempt

```
#for clustering effectiveness
ratio_bss_tss <- (bss/tss)*100

silh <- silhouette(kmean_2$cluster, dist(scaled_removed_data))
fviz_silhouette(silh, palette=c("#ff4500", "#836fff"))
```

A silhouette plot is made using the fviz_silhouette function of the factoextra package. It shows the silhouette width of each observation as a vertical bar, with the colour of the observation's cluster and the height indicating the intensity of the clustering. Additionally, the plot shows each cluster's average silhouette width as a horizontal line.



An overview measure that shows the overall effectiveness of the clustering strategy is the average silhouette width score. The range of the average silhouette is -1 to 1, where smaller values correspond to better clustering and more significant values to better clustering. A value less than 0 implies that the observations could be incorrectly categorised, whereas a number around 0 indicates that the observations are near at the decision boundary between clusters.

Each point is allocated to the appropriate cluster, and the higher average silhouette width score indicates that the clusters are clearly defined.

But in this case, the average silhouette width is 0.21. in this case, the clustering is based on the number of clusters provided by the automated tools, so the average silhouette width of 0.21 is not good.

2nd Subtask Objectives (PCA)

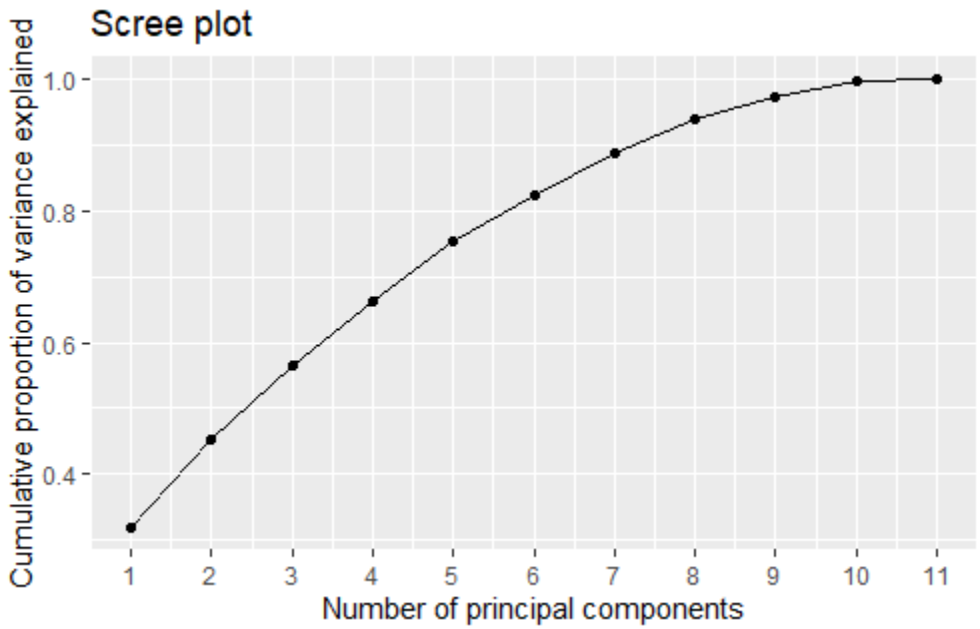
2.1 PCA

The PCA results are contained in the pca object, which also includes additional diagnostic data and the principal component (PC) scores, eigenvalues, and eigenvectors.

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

The first six principal components accurately represent the original data, accounting for approximately 85% of the total variation, as shown in the table.

We can reduce the dimensionality of the data while keeping most of the information by choosing these seven components. Applications such as data visualisation, feature extraction, and data compression can all benefit from this.



Also, the scree plot shows that the graph begins to level out after PC 8 and ends completely after PC 9, suggesting that the significance of those PCs decreases.

The final definition is first 7 PC are more optimal in this case for the transformed data.

```
> -pca$rotation #principal components loading
                         PC1         PC2         PC3          PC4          PC5          PC6
fixed.acidity        0.140280555 -0.61203007  0.08099541  0.050273825  0.239390810  0.202119195
volatile.acidity    -0.001648807  0.05314887 -0.62457123  0.272511245  0.512961409 -0.447784784
citric.acid          0.040124348 -0.32132792  0.54159243  0.330132076 -0.053376798 -0.561054834
residual.sugar       0.410982945 -0.03933982 -0.24898684  0.006918946 -0.089172378  0.155132459
chlorides            0.338328117  0.03473757  0.15695036 -0.349345357  0.152999670 -0.449712143
free.sulfur.dioxide  0.289171695  0.22266397  0.04181912  0.589572059 -0.325841762  0.152859046
total.sulfur.dioxide 0.401451559  0.18757223 -0.00715271  0.364120908 -0.022116850 -0.091230206
density              0.497599117 -0.01203937 -0.04204983 -0.177527935  0.046285535  0.079510269
pH                  -0.102081611  0.61384666  0.22229552 -0.146213582  0.004448838 -0.185707198
sulphates            0.049771260  0.22933911  0.40733541  0.174126630  0.732471586  0.380780719
alcohol             -0.435946613 -0.04771543 -0.06704241  0.358802195  0.003763370  0.007426925
                         PC7         PC8         PC9         PC10         PC11
fixed.acidity        0.14742572  0.66010302 -0.09643739 -0.087425407  0.155174115
volatile.acidity    -0.06672807  0.05095558  0.08539279 -0.234020372  0.008584601
citric.acid         -0.38257151 -0.11512197  0.11260850 -0.052113995  0.013663161
residual.sugar      -0.56809914 -0.12371710 -0.37998724  0.182038560  0.468950306
chlorides            0.44095616 -0.13035265 -0.54740198  0.022568850  0.020235091
free.sulfur.dioxide  0.23068335  0.01809707 -0.22949796 -0.529923070 -0.029792713
total.sulfur.dioxide 0.27765658  0.15996883  0.30966135  0.679316666  0.046492099
density             -0.31493143  0.11221931  0.04470069 -0.049159354 -0.770916214
pH                  -0.26787365  0.63893607 -0.07919648 -0.066970681  0.129298929
sulphates           -0.05768005 -0.24621799 -0.04396560 -0.007604633  0.035135493
alcohol             -0.07549850  0.09350628 -0.60656849  0.389844670 -0.374196636

> pca$sdev^2  #standard deviation of every PC
 [1] 3.51307584 1.47894520 1.23237648 1.07598238 0.97945806 0.77277019 0.70870498 0.59029437
 [9] 0.35782562 0.27753115 0.01303572
```

The following code fragment is used for the transformation the data with first 7 principal components.

```
transformed <- data.frame(pca$x[,1:7], class= removed_data[,-13])
transformed_data <- transformed[, 1:7]
```

2.2 Check the number of clusters using automated tools after PCA

There are a few methods for automated clustering tools that are frequently used. This project includes the NbClust, Elbow methods, Gap Statistics, and silhouette methods.

2.2.1 NbClust Method

This method was previously mentioned and is now being used for clustering, as the steps mentioned earlier are incorporated here.

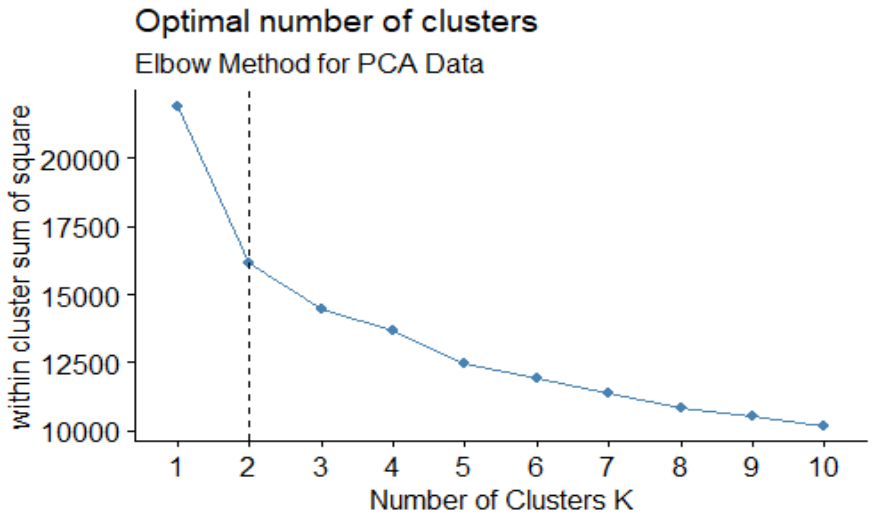
```
nbc_pca <- NbClust(transformed_data, distance="euclidean",
                   method="kmeans", min.nc=2, max.nc=10, index="all")
```

```
*** : 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:
* 12 proposed 2 as the best number of clusters
* 8 proposed 3 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 1 proposed 9 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
```

2.2.2 Elbow Method

Here, I also mentioned earlier the steps of the elbow method.

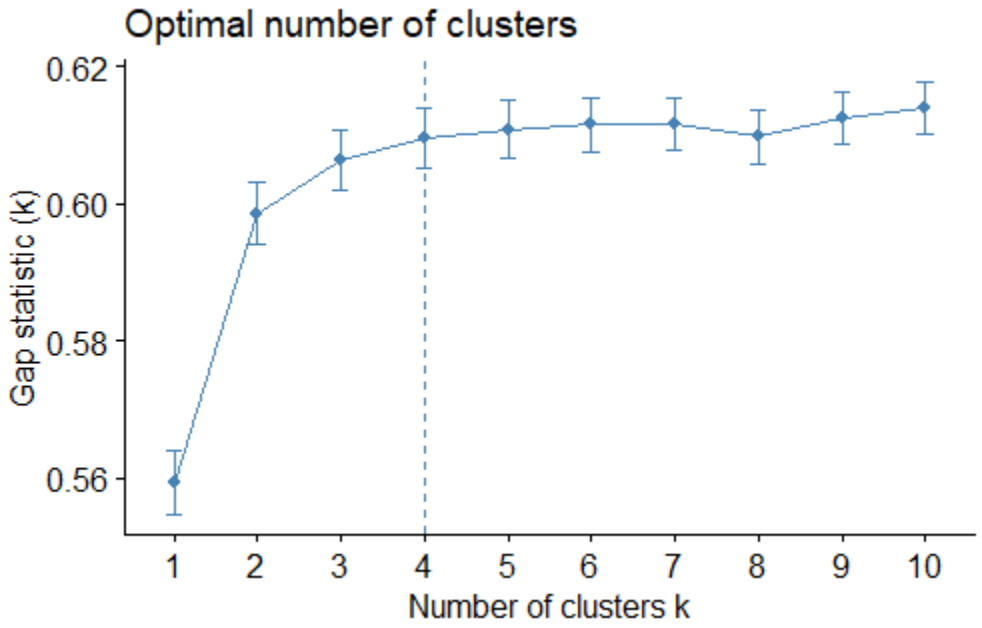
```
#Elbow method with PCA

set.seed(26)
fviz_nbclust(transformed_data, kmeans, method="wss")+
  geom_vline(xintercept = 3, linetype = 2)+
  labs(subtitle = "Elbow Method for PCA Data",
       x='Number of Clusters K', y='within cluster sum of square')
```



After doing PCA, the suggestion has not been changed.

2.2.3 Gap Statistic Method

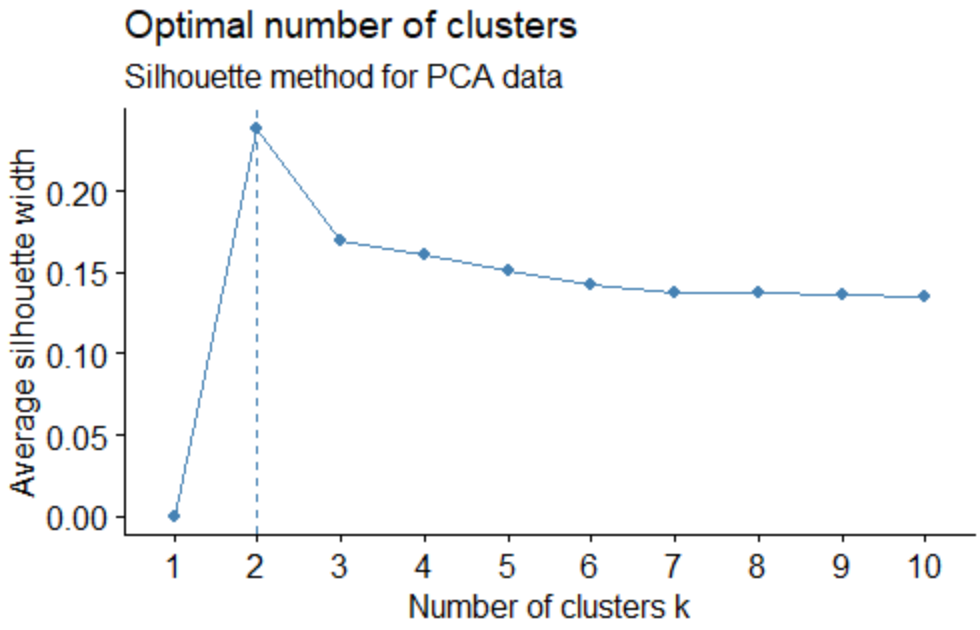
```
#gap statistic method with PCA

set.seed(26)
gap_stat_pca <- clusGap(transformed_data, kmeans, K.max = 10,
                        nstart=25, B=50)
fviz_gap_stat(gap_stat_pca)
```



However, the gap statistic method suggests the optimal number of clusters, in this case, is 4.

2.2.4. Silhouette Method

```
#silhouette method with PCA
fviz_nbclust(transformed_data, kmeans, method = "silhouette")+
  labs(subtitle = "Silhouette method for PCA data")
```



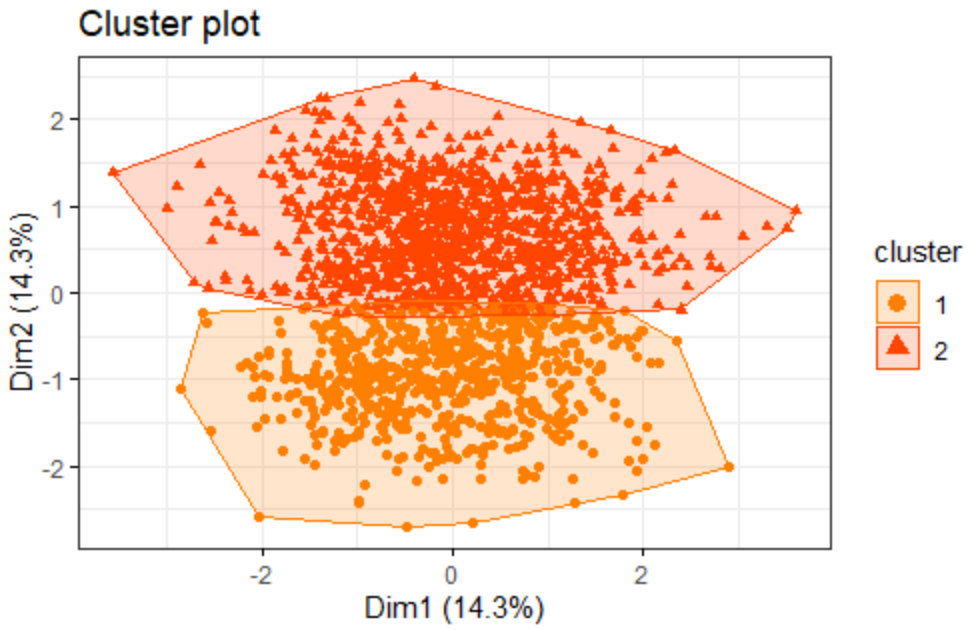
After the PCA suggestion is the same as the previous one.

2.2.5 Conclusion of the automated tools after PCA

Three out of four automated tools recommended using two clusters as the best number for clustering the given data set, while the other tool (gap statistic) suggested using four clusters. In summary, the majority favoured two clusters as the optimal choice.

2.3 K-means clustering after PCA

```
#kmeans with PCA
pca_kmean_2 <- kmeans(transformed_data,2)
fviz_cluster(pca_kmean_2, data = transformed_data,
             palette= c("#ff7f00", "#ff4500"),
             geom = "point", ellipse.type = "convex", ggtheme = theme_bw()
             )
```




```
> pca_kmean_2
K-means clustering with 2 clusters of sizes 896, 1347

Cluster means:
        PC1        PC2         PC3         PC4          PC5          PC6         PC7
1 -1.951519 -0.1336637  0.08854496 -0.06518916 -0.002771397 -0.003621631  0.08163956
2  1.298115  0.0889107 -0.05889850  0.04336265  0.001843483  0.002409044 -0.05430516


Within cluster sum of squares by cluster:
[1]  6083.505 10064.505
 (between_SS / total_SS =  26.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
```

BSS, WSS, TSS and cluster center values are as follows

```
> pca_kmean_2[["betweenss"]]
[1] 5736.854
> pca_kmean_2[["tot.withinss"]]
[1] 16148.01
> pca_kmean_2[["totss"]]
[1] 21884.86
> pca_kmean_2[["centers"]]
        PC1        PC2         PC3         PC4          PC5          PC6         PC7
1 -1.951519 -0.1336637  0.08854496 -0.06518916 -0.002771397 -0.003621631  0.08163956
2  1.298115  0.0889107 -0.05889850  0.04336265  0.001843483  0.002409044 -0.05430516
```

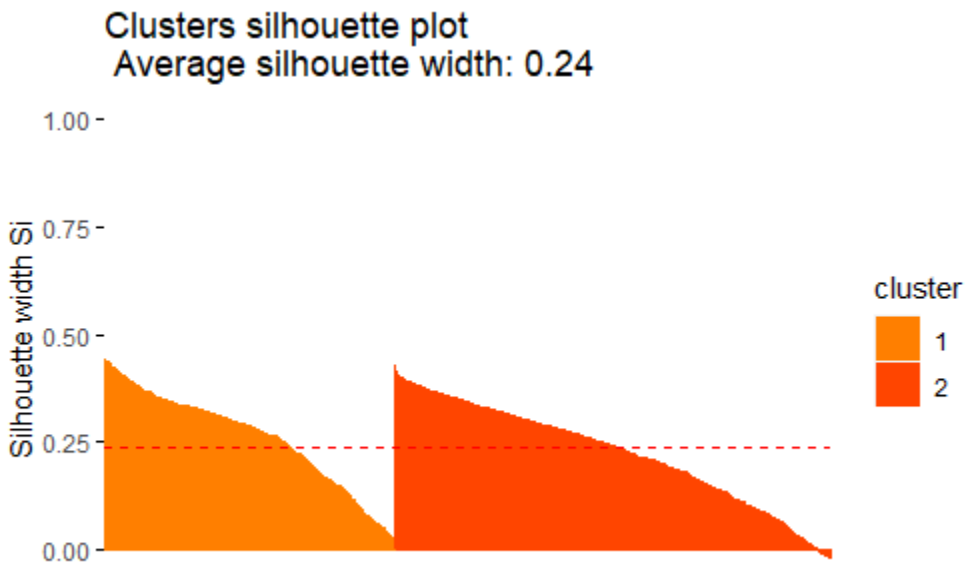
The ratio of BSS over TSS

```
> pca_ratio_bss_tss <- (pca_bss/pca_tss)*100
> pca_ratio_bss_tss
[1] 26.2138
```

2.4 Silhouette plot analysis of kmean after PCA

```
pca_silh <- silhouette(pca_kmean_2$cluster, dist(transformed_data))
fviz_silhouette(pca_silh, palette= c("#ff7f00", "#ff4500"))
```

In this part, the average silhouette width of 0.24 is a good score after PCA.



2.5 Calinski-Harabaz Index

An internal cluster validation metric called the Calinski-Harabasz index calculates the ratio of within-cluster dispersion to between-cluster dispersion to evaluate the quality of a clustering solution. To calculate the index, multiply the sum of squared distances inside each cluster by the sum of squared distances between cluster centroids. Then, divide the result by the total number of observations minus the number of clusters minus one. The Calinski-Harabasz index value increases with the quality of the clustering solution.

```
> ch_index <- calinhara(transformed_data, pca_kmean_2$cluster)
> print(ch_index)
[1] 796.1533
>
```

The clustering solution in this project has a rather high Calinski-Harabasz score of 796.1533, indicating that it is of excellent quality.

2. Financial Forecasting Part

When using Multilayer Perceptron (MLP) models for exchange rate forecasting, it is crucial to define the input vector. This is necessary to identify relevant patterns and relationships in the time series data. The input vector contains variables that the MLP model will use to forecast future exchange rates. In the financial domain, different techniques and approaches are employed to define this input vector, especially in exchange rate forecasting. The following is a typical method used for defining the input vector.

1. Autoregressive (AR) Approach
   The autoregressive (AR) approach is a method that uses previous values of the target variable (financial rate) as input variables. In this method, the input vector consists of a series of historical exchange rates at regular time intervals. The AR model assumes that the future of the target variable depends on its past values.

2. Moving Average (MA) Approach

   The input vector for this technique contains moving average values of the target variables at different time intervals. According to the MA model, the target variable's future value will be determined by averaging its value over a range of time periods.

3. Autoregressive Moving Average (ARMA) Approach

   This technique combines the AR and MA models and uses both time-delayed values and the moving average values of the target variable as input variables. Based on historical data and average values across different time periods, the RMA model predicts that the target variable's
   The future value will be determined by these two factors.

4. Seasonal Autoregressive Integrated Moving Average (SAIMA) Approach

   This method extends the capability of the ARMA model by adding sessional components, such as time-delayed and moving average values at regular and sessional intervals in the input vector of the SARIMA model.

2.1 Creating I/O Matrices.

In this project, we predict the next day's exchange rate using the historical data.

```
#create I/O matrix function for testing and training
create_io_df <- function(data, num_cols){
  dataInput <- matrix(, nrow = 0, ncol = num_cols)
  dataOutput <- c() # initialize vector for store the output

  for(i in 1:length(data)){ #start from the first row
    lastValue <- i + (num_cols - 1)
    if(lastValue+1>length(data)){
      break
    }
    input <- data[i:lastValue]
    output <- data[lastValue+1]

    dataInput <- rbind(dataInput, input)
    dataOutput <- append(dataOutput, output)
  }
  dataIoDf <- cbind(as.data.frame(dataInput), dataOutput) #combine the input matrix with output vect
  return(dataIoDf)
}
```

```
#create io_matrix for training and testing for t-4
create_t4IoDf <- function(data, num_cols=5) {
  dataInput <- matrix( ,nrow = 0, ncol = num_cols) #matrix to extract input values
  dataOutput <- c() #vector to store output value

  for (i in 4:length(data)) { #starting from 4th row, use a lag of 4 days in input vector
    lastValue <- i + (num_cols - 2)
    t4_value <- (lastValue - 3)
    if (lastValue + 1 > length(data)) {
      break
    }
    input <- data[i:lastValue-1]
    t4_input <- data[t4_value]
    input <- append(input, t4_input)
    output <- data[lastValue + 1]

    dataInput <- rbind(dataInput, input) #add the new input vector to input matrix
    dataOutput <- append(dataOutput, output) #add new output value to output vector
  }

  data_4tIoDf <- cbind(as.data.frame(dataInput), dataOutput)
  return(data_4tIoDf)
}

#creating io_matrix for training
training_io_1 <- create_io_df(train_data, 1)
training_io_2 <- create_io_df(train_data, 2)
training_io_3 <- create_io_df(train_data, 3)
training_io_4 <- create_t4IoDf(train_data, 4)

#creating io_matrix for testing
testing_io_1 <- create_io_df(test_data, 1)
testing_io_2 <- create_io_df(test_data, 2)
testing_io_3 <- create_io_df(test_data, 3)
testing_io_4 <- create_t4IoDf(test_data, 4)
```

2.2 Normalizing the Data

When training neural networks, especially Multilayer Perceptron (MLP) topologies, data normalisation is a common preprocessing step. The training process and networking performance can be improved by normalisation which guarantees that all features or variables in the dataset have comparable scales. The network may find it difficult to cover and may require a longer training period when the input attribute size changes significantly.

The process of normalisation involves bringing the input data's mean and standard deviation down to zero. There are several ways to do this, including min-max normalisation and z-score scaling. Min-max normalisation is applied to this instance.

Normalisation facilitates data centring and facilitates MLP learning. This may lead to enhanced generalisation and quicker convergence, suggesting that the MLP can function effectively on fresh, untested data.

Additionally, normalisation can help reduce the impact of high values or outliers in the dataset, leading to an overfit or underfit of the network in the data. By reducing the data to a comparable range, outliers have less impact, and the network can better identify patterns in the data.

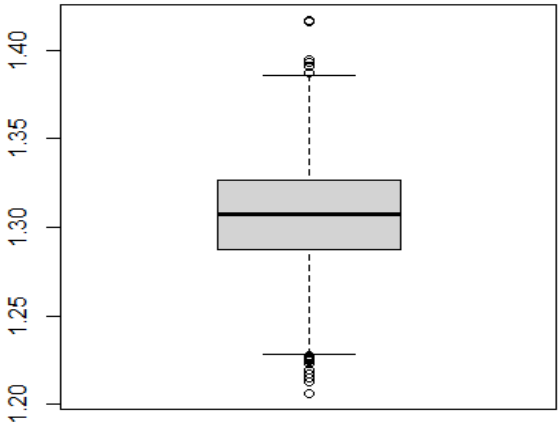
The following has the normalising data function.

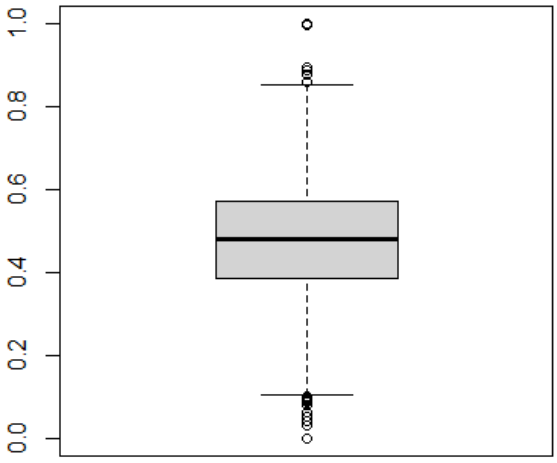
```
#Normalize function
normalize <- function(x){
  return ((x - min(x)) / (max(x) - min(x)))
}
```

Normalise the data using the above function.

```
#normalizing the data
exchangeUSD_norm <- as.data.frame(lapply(exchangeUSD[,3], normalize))
```

Boxplots for before and after normalisation part.





2.3 Training MLP

The code fragment below demonstrates how the function in this project trains multiple MLP models using different input vectors and internal network architectures. This includes variations in the number of nodes, hidden layers, and activation functions. The code also shows that a total of 13 MLP models have been trained.

```
#create neural network func
create_neural_network <- function(data, input_cols, hidden_layers, act_fct) {
  formula <- as.formula(paste("dataOutput ~", paste(input_cols, collapse = " + ")))
  set.seed(12345)
  neural_network_model <- neuralnet(formula, data = data, hidden = hidden_layers,
                       act.fct = act_fct, linear.output = TRUE) #create neural network model
  return(neural_network_model)
}

# Training Models
input1_nn_1 <- create_neural_network(training_io_1, c("V1"), c(5), "logistic") # with logistic sigmoid function
input1_nn_2 <- create_neural_network(training_io_1, c("V1"), c(7), "logistic")
input1_nn_3 <- create_neural_network(training_io_1, c("V1"), c(5,7), "tanh") # with hyperbolic tangent function
input2_nn_1 <- create_neural_network(training_io_2, c("V1", "V2"), c(5), "tanh")
input2_nn_2 <- create_neural_network(training_io_2, c("V1", "V2"), c(7), "tanh")
input2_nn_3 <- create_neural_network(training_io_2, c("V1", "V2"), c(5,7), "logistic")
input3_nn_1 <- create_neural_network(training_io_3, c("V1", "V2", "V3"), c(5), "tanh")
input3_nn_2 <- create_neural_network(training_io_3, c("V1", "V2", "V3"), c(10), "logistic")
input3_nn_3 <- create_neural_network(training_io_3, c("V1", "V2", "V3"), c(5,10), "logistic")
input4_nn_1 <- create_neural_network(training_io_4, c("V1", "V2", "V3", "V4"), c(5), "logistic")
input4_nn_2 <- create_neural_network(training_io_4, c("V1", "V2", "V3", "V4"), c(10), "tanh")
input4_nn_3 <- create_neural_network(training_io_4, c("V1", "V2", "V3", "V4"), c(5,7), "tanh")
input4_nn_4 <- create_neural_network(training_io_4, c("V1", "V2", "V3", "V4"), c(7,10), "logistic")
```

After training the MLP, the following code evaluates the models.

```r
#testing and evaluating function for NN model
evaluating_neural_network <- function(ordern_neural_network, ordern_test_io, cons_data = exchangeUSD) {
  #test the neural networks with test data
  number_col_i <- ncol(ordern_test_io)#number of columns in test I/O data
  testing_data <- data.frame(ordern_test_io[,1:(number_col_i - 1)])
  set.seed(12345)
  ordern_nn_results <- compute(ordern_neural_network, testing_data) # uses the trained NN to compute predictions
  # results of NN
  results <- data.frame(actual = ordern_test_io$dataOutput,
                        prediction = ordern_nn_results$net.result) #actual and predicted data frame

  resultsMin <- min(cons_data$"USD/EUR") #store the minimum rate from the dataset
  resultsMax <- max(cons_data$"USD/EUR") #store the maximum rate from the dataset

  #unnormalized the predicted and actual outputs
  comparison <- data.frame(
    predicted = unnormalize(results$prediction, resultsMin, resultsMax),
    actual = unnormalize(results$actual, resultsMin, resultsMax)
  )

  #1st layer weights
  weights_layer1 <- ordern_neural_network$weights[[1]]
  #calculate the number of hidden layers in NN
  hidden_layer_no <- length(weights_layer1) - 1
  if(hidden_layer_no == 2) {
    hidden_layer_config_1 <- ncol(ordern_neural_network[["weights"]][[1]][[1]])# number of neuron in the 1st hidden layer
    hidden_layer_config_2 <- ncol(ordern_neural_network[["weights"]][[1]][[2]])# number of neurons in the second layer
    hidden_layer_config <- c(hidden_layer_config_1, hidden_layer_config_2)
  }else {
    hidden_layer_config_1 <- ncol(ordern_neural_network[["weights"]][[1]][[1]])
    hidden_layer_config <- c(hidden_layer_config_1)#combine the number of neurons in the hidden layer
  }

  #statistic indices
  RMSE <- rmse(comparison$actual, comparison$predicted) #Root Mean Square Error
  MAE <- mae(comparison$actual, comparison$predicted) # Mean Absolute Error
  MAPE <- mape(comparison$actual, comparison$predicted) # Mean Absolute Percentage Error
  sMAPE <- smape(comparison$actual, comparison$predicted) # Symmetric Mean Absolute Percentage Error

  #crate a data frame to store the evaluation matrics with information about the NNA
  evaluation_metrix <- data.frame(No_of_inputs = (number_col_i-1), No_of_hidden_layers = hidden_layer_no,
                          Hidden_layer_cofig = paste(hidden_layer_config, collapse = " + "),
                          RMSE = RMSE, MAE = MAE, MAPE = MAPE, sMAPE = sMAPE)
  plot(ordern_neural_network) #Generate the plots

  return(list(evaluation_metrix = evaluation_metrix, comparison = comparison))
}
```

The output values of the network may be normalised to a particular range during the training phase when using a multilayer perceptron (MLP) for prediction. Improving the convergence of the network and making it easier to work with may benefit from this. Unnormalising the project values to their original scale is crucial for assessing the network's performance.

Therefore, in this case, the data is unnormalized before the common statistical indices computations are started.

It is important to note that statistical indices like mean square error (MSE) and mean absolute error (MAE) are typically calculated on the original scale of the data. However, this can cause issues with interpretation and usefulness if the expected values have not been unnormalized prior to these calculations.

The MSE computed on the normalized predictions will be different from the MSE computed on the original scale if the MLP output values are normalized to the interval [0, 1]. Interpreting the assessment findings or comparing the performance of several models may become difficult as a result.

When we normalize data during training, we need a way to reverse this process to obtain meaningful predictions. This can be achieved by applying the inverse normalization function that was used during training. By doing so, the predictions are returned to their original scale, allowing for a relevant and understandable comparison with actual values.

Overall, unnormalized predicted values are essential for assessing an MLP's performance and calculating standard indices because they ensure that the findings are understandable and relevant to the data's original scale.

2.4 Description of statistical indices

1. Root Mean Square Error (RMSE)

The Root Mean Square Error (RMSE) is a measure of the average magnitude of the errors in a set of predicted values as compared to their actual values. It is calculated as the square root of the average of the squared differences between the predicted and actual values. RMSE is commonly used to assess the accuracy of regression models.

2. Mean Absolute Error (MAE)

The average size of a mistake in a collection of projected values compared to actual values is calculated using the metric. MAE is calculated as the average of the absolute differences between the values that were predicted and those that were observed. It is often used to evaluate how accurate forecasts and predictions are.

3. Means Absolute Percentage Error (MAPE)

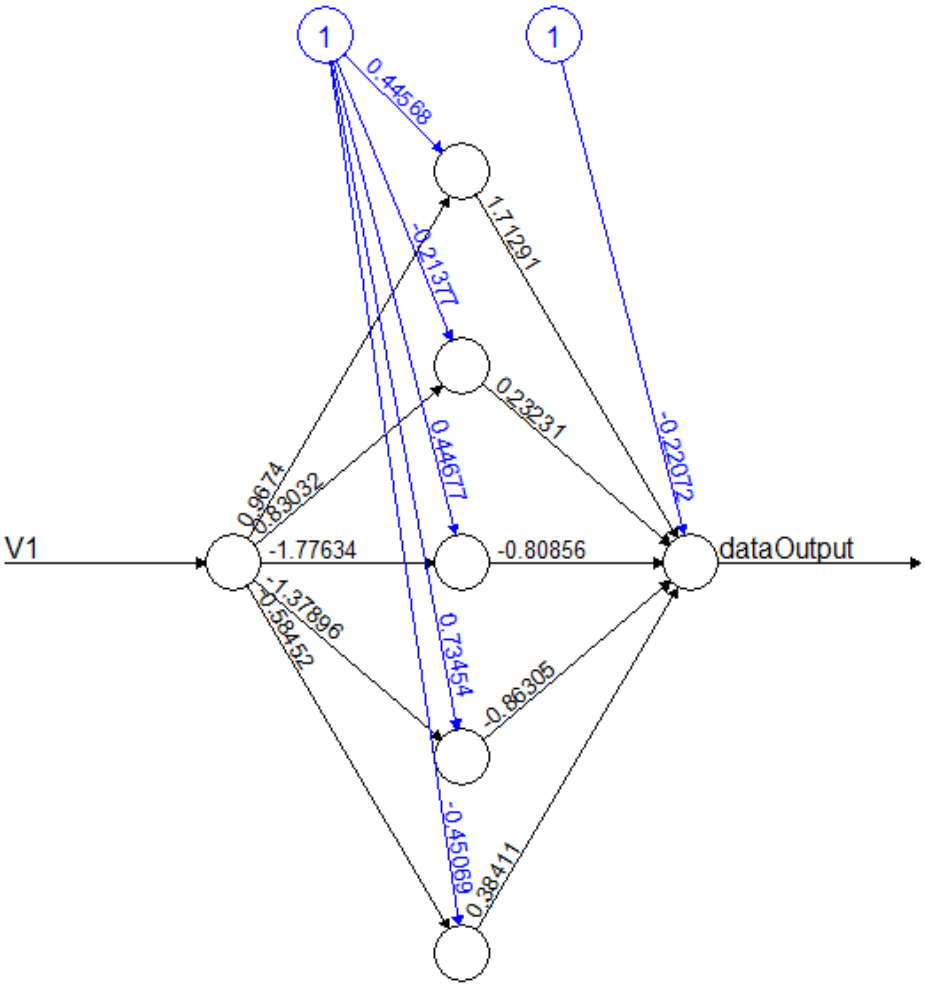
Measuring the percentage difference between predicted and actual data sets. The formula calculates MAPE as the average of the absolute percentage difference between the expected and actual values. In situations where the error size is substantial, it is commonly employed to evaluate the accuracy of forecasts or predictions.

4. Symmetric Mean Absolute Percentage Error (SMAPE)

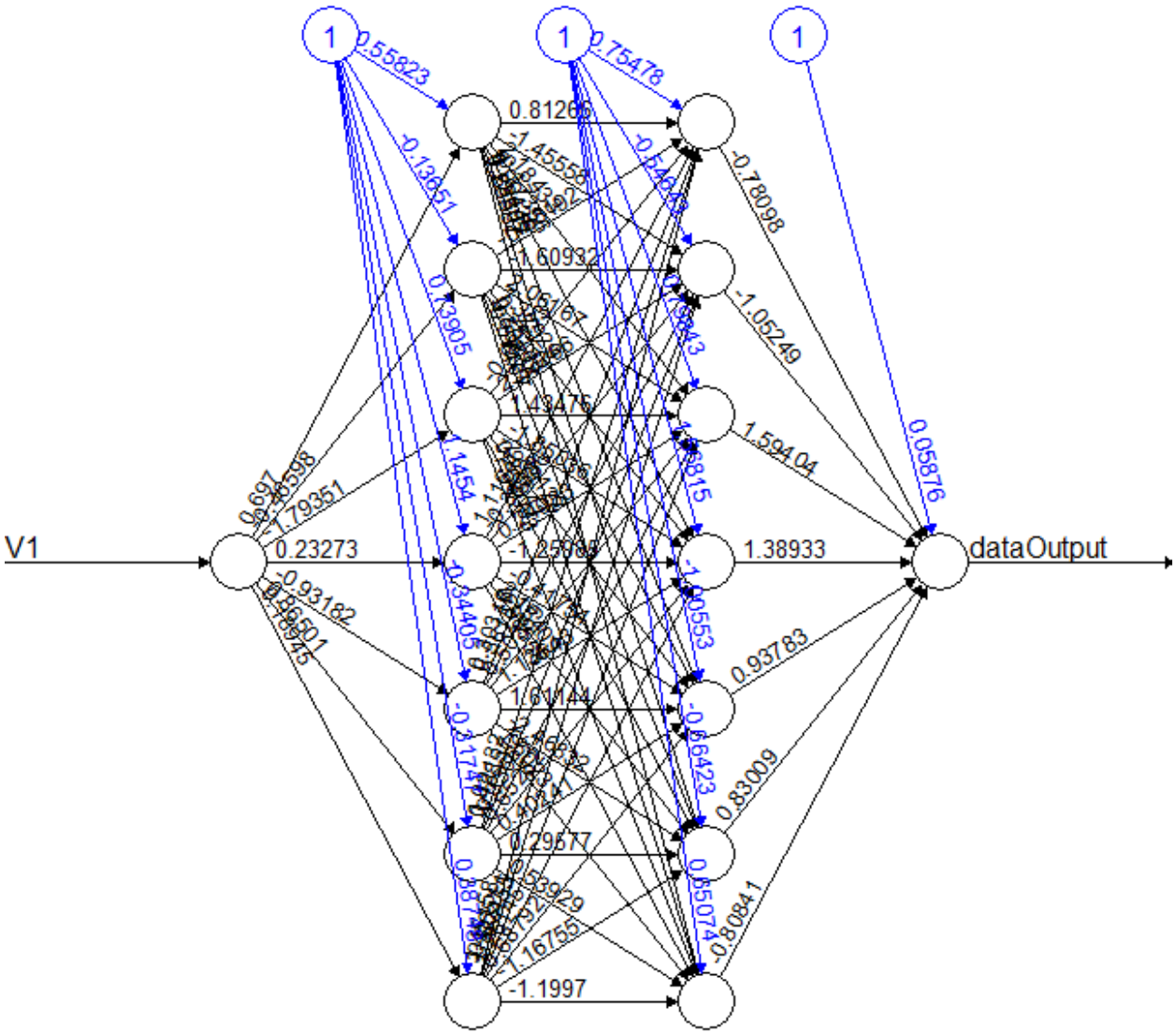
While taking the data scale into consideration, it computes the average percentage difference between the projected and actual values. By dividing the total absolute values of the predicted and actual values by the average of the absolute percentage differences between the expected and actual values, the SMAPE is calculated. When evaluating the precision of forecasting or predictions, time series analysis frequently uses it.

2.5 AR Models (MLP)

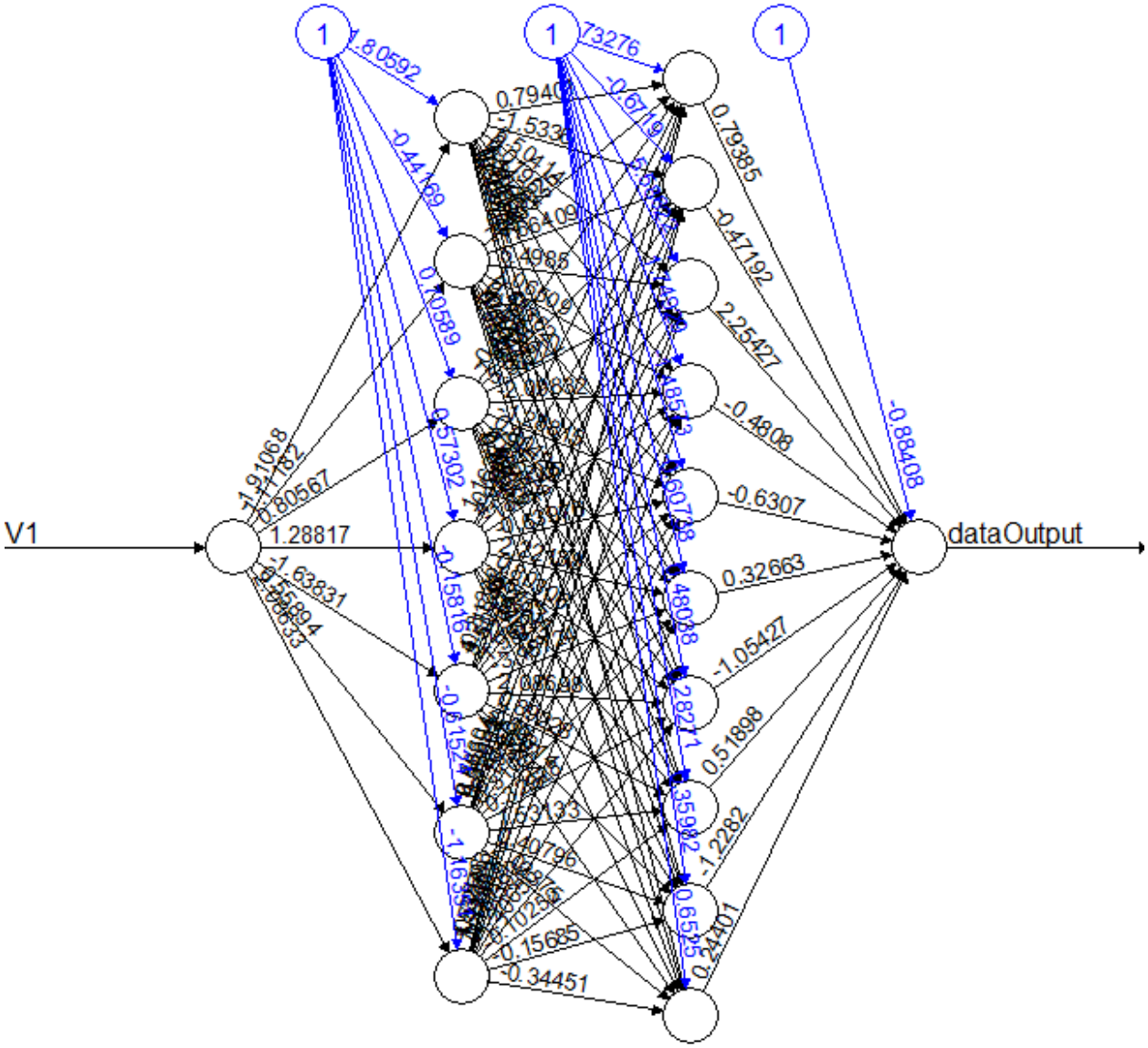
1 Input Models



Error: 0.252259 Steps: 193

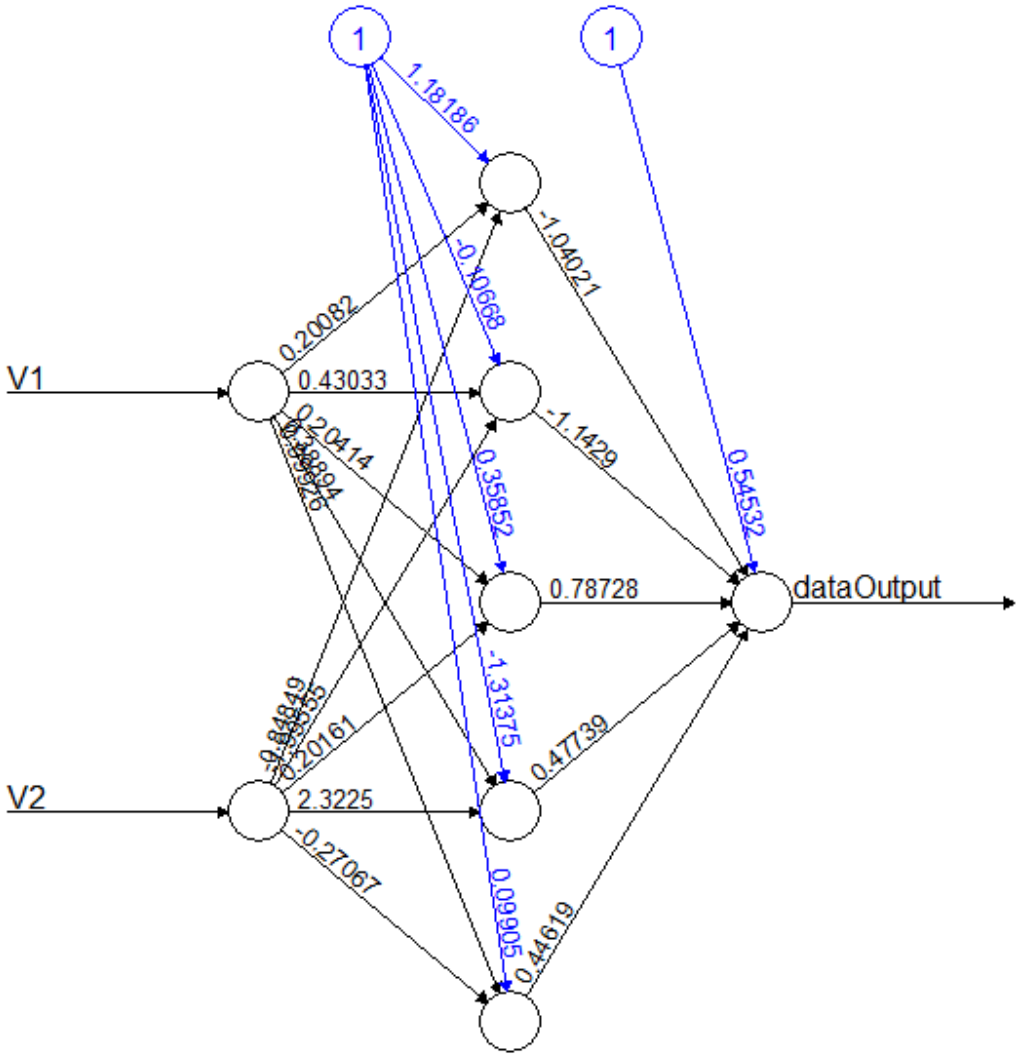


Error: 0.24636 Steps: 501

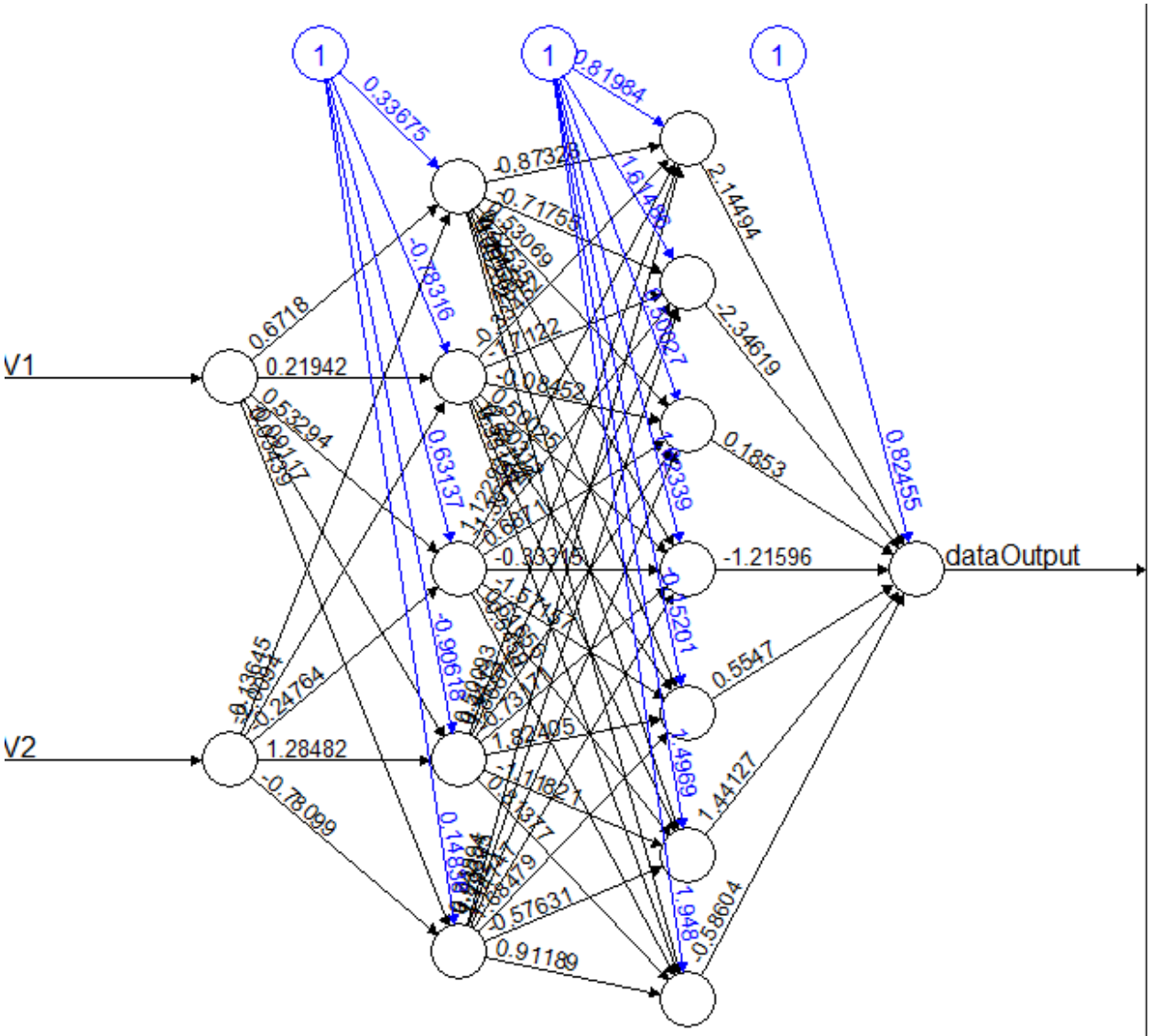


Error: 0.249857 Steps: 316

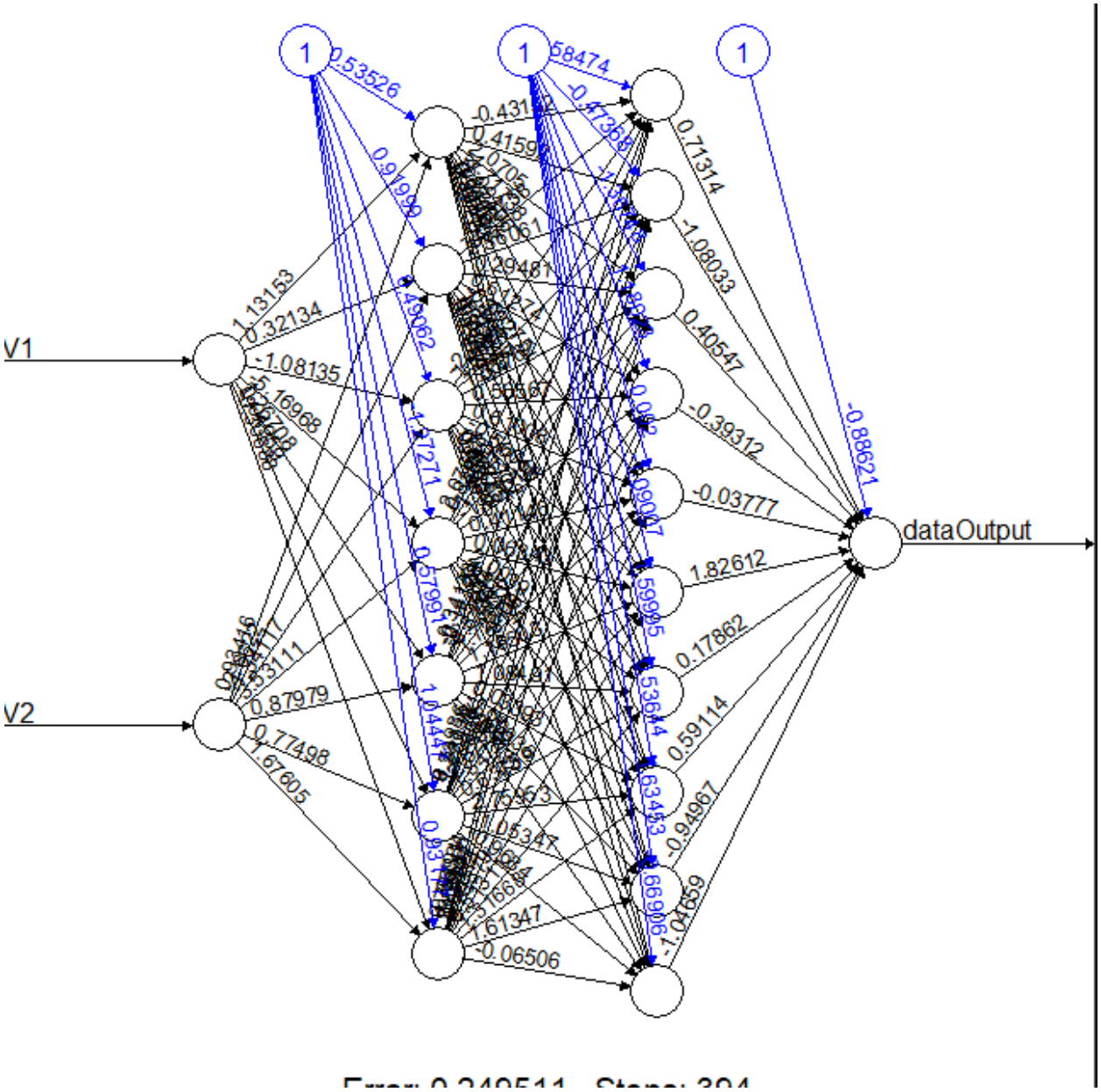
2 input models



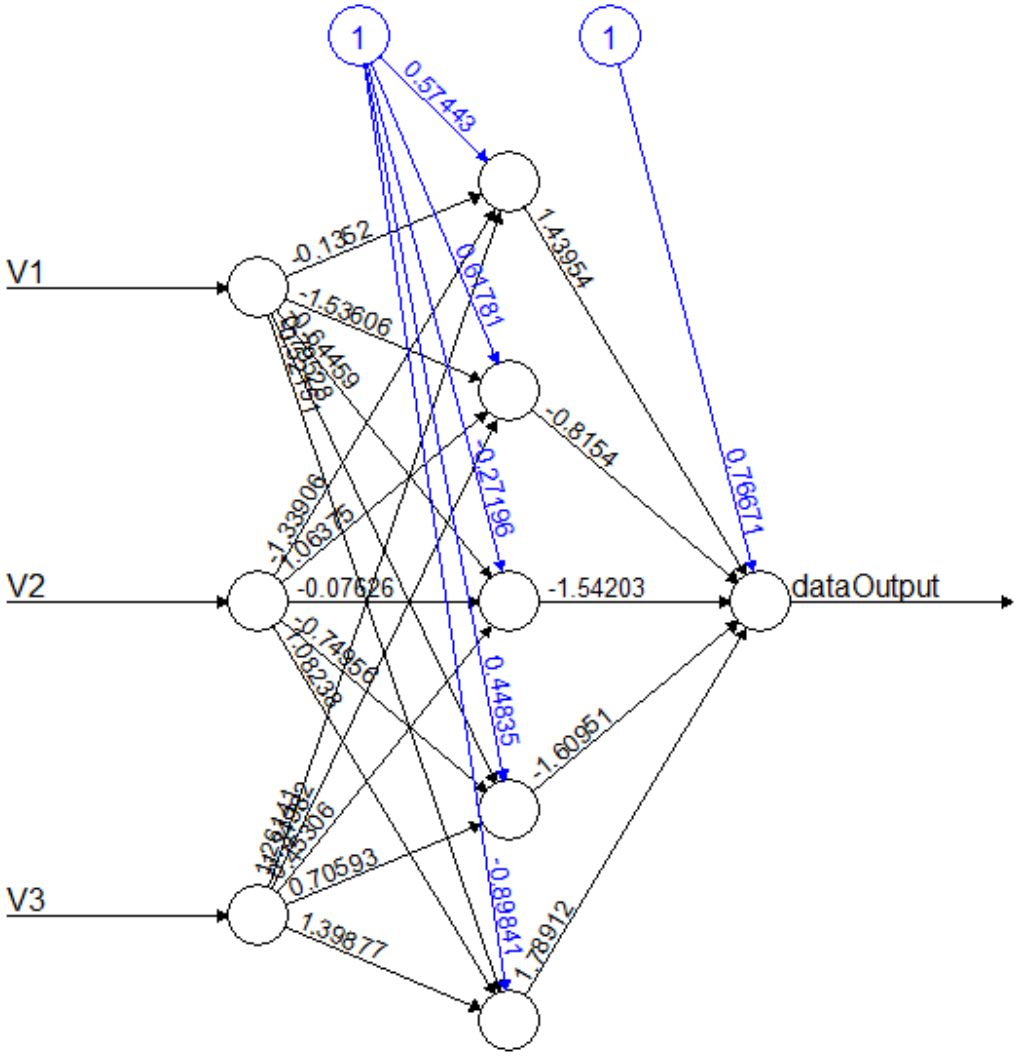
Error: 0.247815 Steps: 203



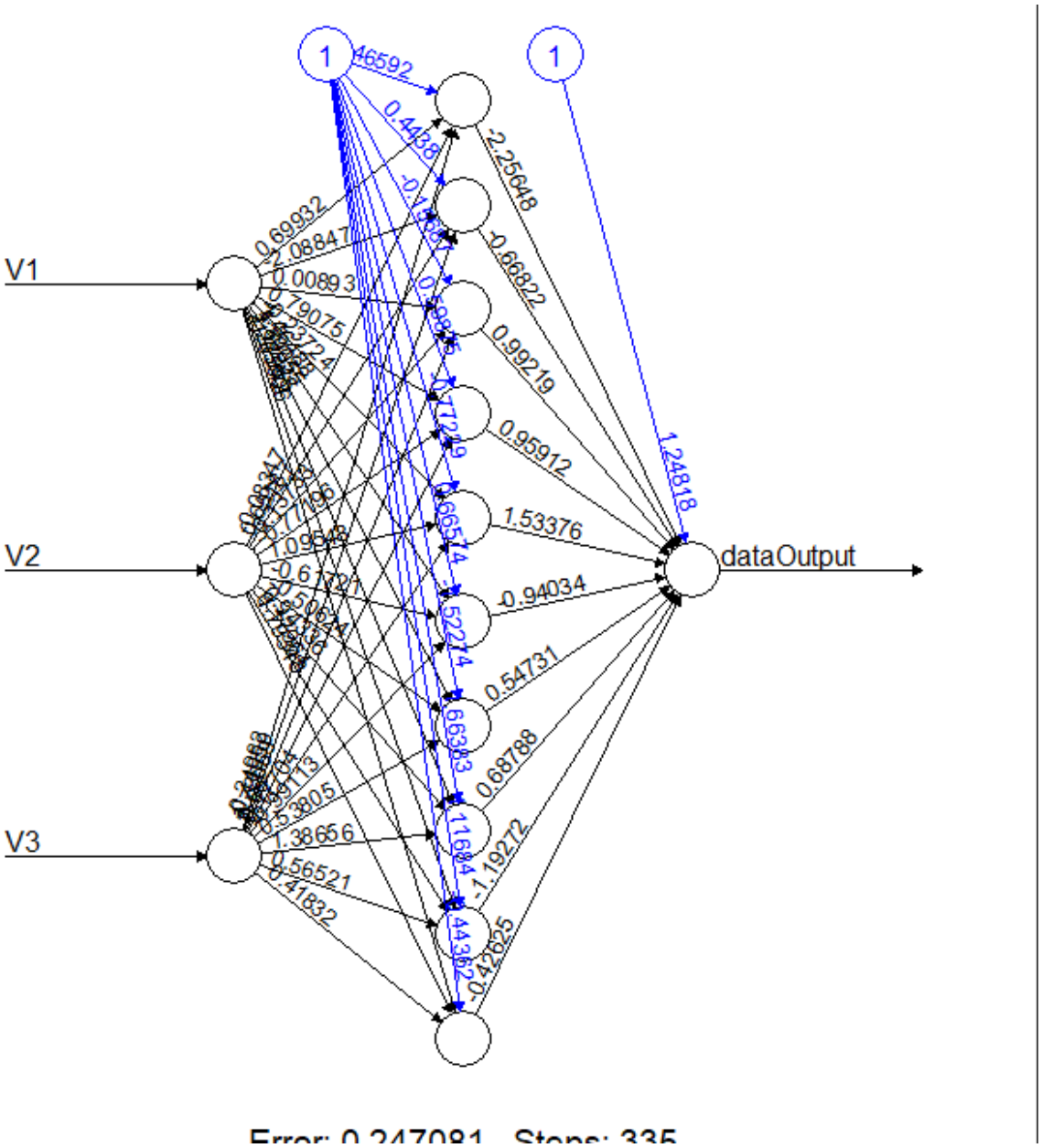
Error: 0.246472 Steps: 795

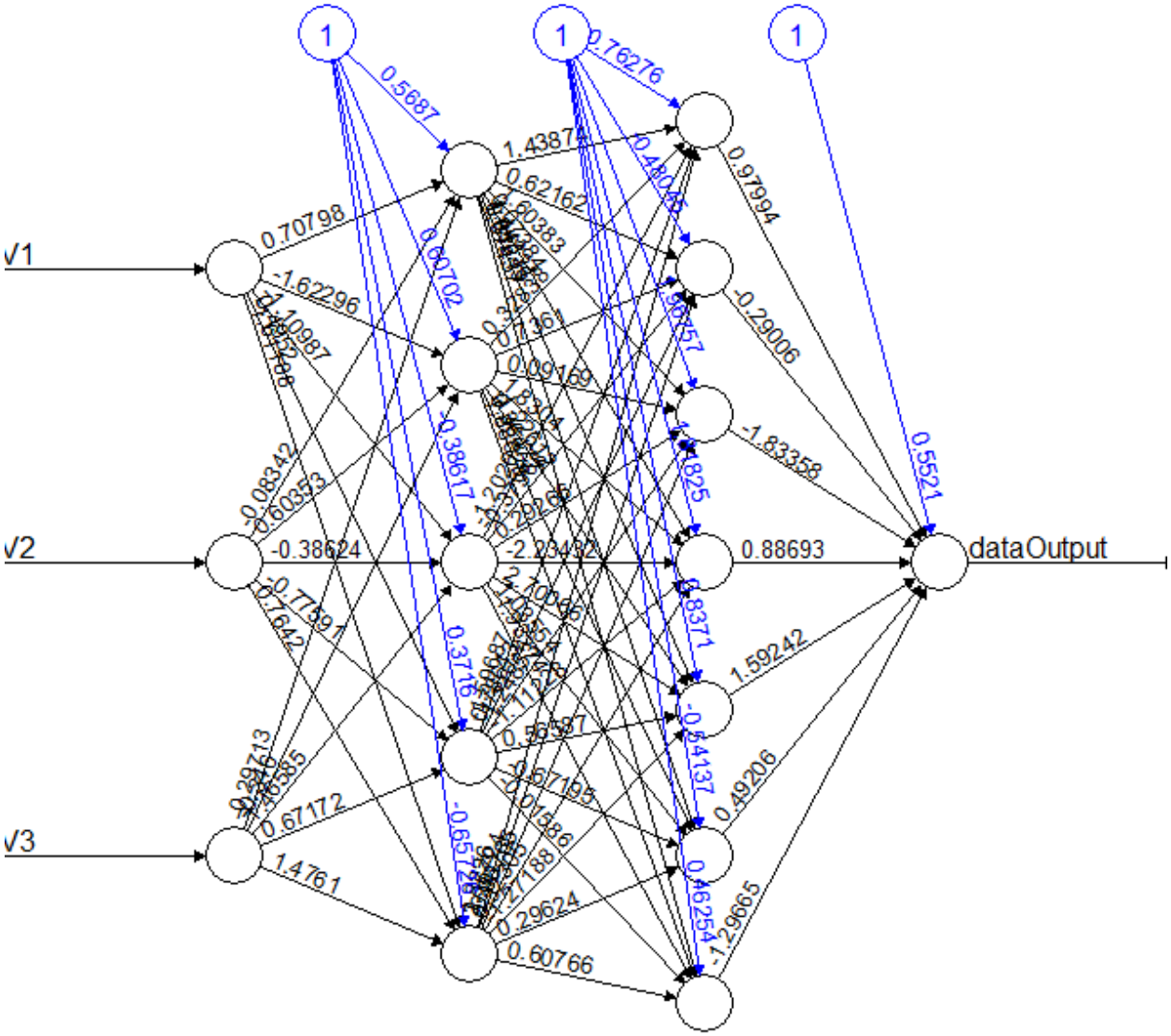

1
1
1
0.53526
58474
-0.47368
0.71314
-1.08033
0.40547
-0.39312
-0.03777
1.82612
0.17862
0.59114
-0.94967
-1.04659
-0.88621
V1
V2
dataOutput
1.13153
0.32134
-1.08135
0.87979
0.77498
1.67605
0.91999
1.61347
-0.06506

3 input models

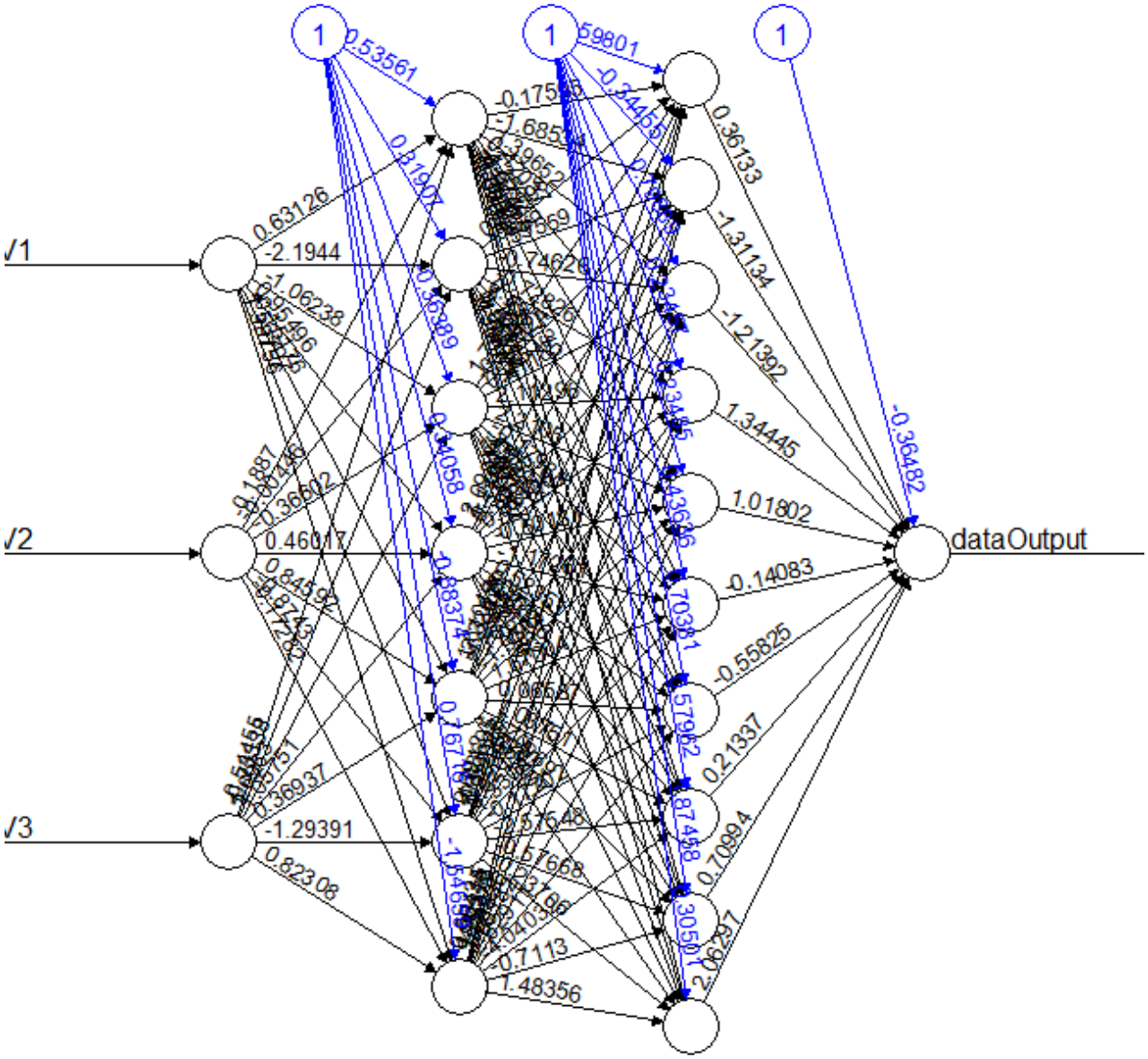


Error: 0.246935 Steps: 1037


1
1
V1
V2
V3
dataOutput
Error: 0.247081 Steps: 335

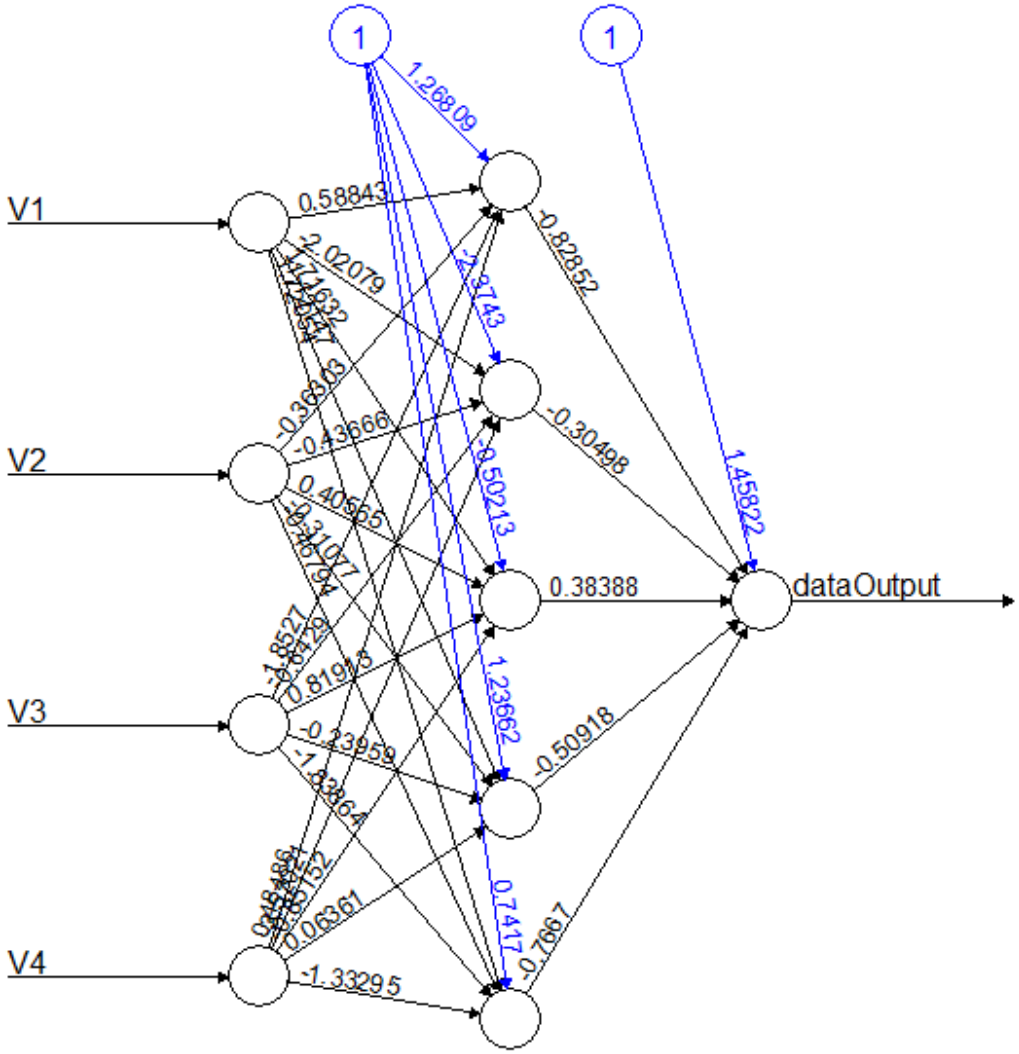


Error: 0.247147 Steps: 1428

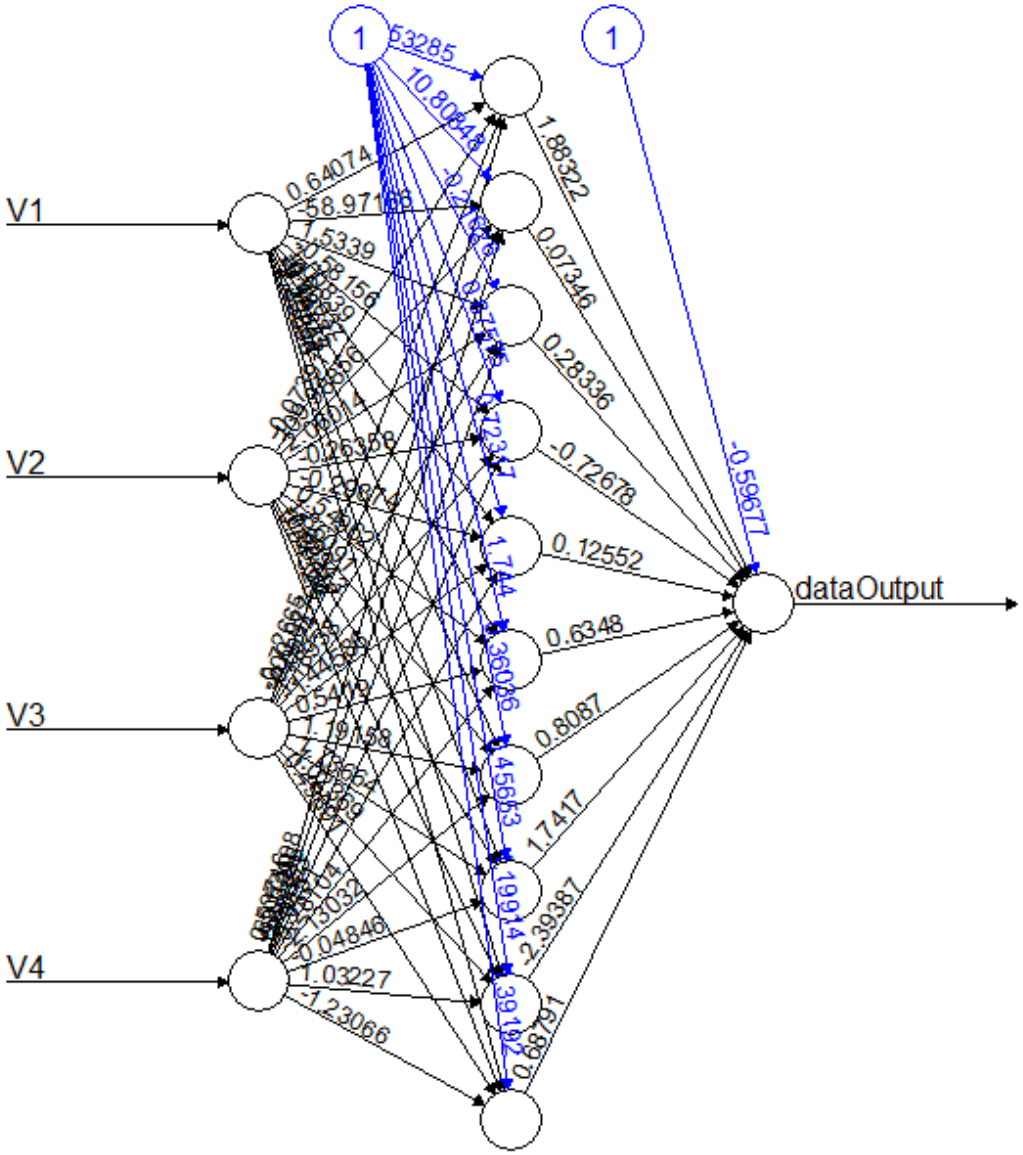


Error: 0.22222 Steps: 2257

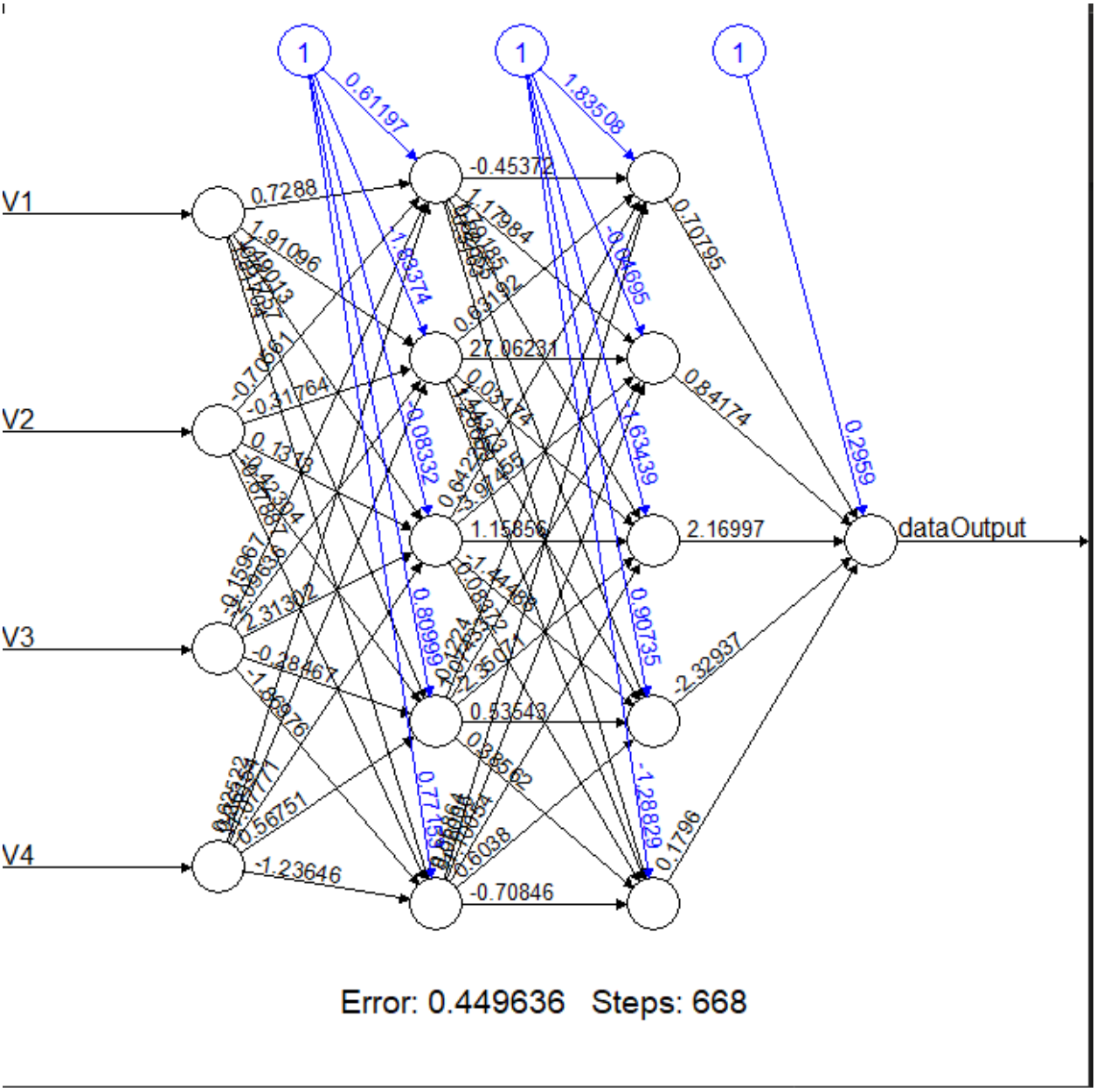
4 input models



Error: 0.461539 Steps: 357



Error: 0.416596 Steps: 5912


1
1
1
0.61197
1.83508
V1
V2
V3
V4
0.7288
1.91096
-0.70661
-0.31764
0.1343
-0.42304
-0.15967
-2.31302
-0.18461
-1.86976
0.56751
-1.23646
-0.45372
1.17984
0.93192
27.06231
0.03474
0.64222
3.97455
1.15656
-1.44488
0.80999
-2.35071
0.53643
0.38562
0.6038
-0.70846
1.83374
-0.08332
0.70795
-0.84695
0.84174
-1.63439
2.16997
0.90735
-2.32937
-1.28829
0.1796
0.2959
dataOutput
Error: 0.449636 Steps: 668

2.6 Comparison Table for Testing Performance

All MPL models have been tested.

Partitioning Clusttering Part.R | Financial Forecasting Part.R | eval_metrix_comparison_df | best_model

Filter

	No_of_inputs	No_of_hidden_layers	Hidden_layer_cofig	RMSE	MAE	MAPE	sMAPE
1	1	1	5	0.006123789	0.004650527	0.003521219	0.003524225
2	1	2	7 + 7	0.006379788	0.004994344	0.003779405	0.003784054
3	1	2	7 + 10	0.006140372	0.004604184	0.003487339	0.003490350
4	2	1	5	0.006107994	0.004598010	0.003479830	0.003482711
5	2	2	5 + 7	0.006141957	0.004579070	0.003465450	0.003468802
6	2	2	7 + 10	0.006141554	0.004557075	0.003450192	0.003452795
7	3	1	5	0.006183297	0.004597522	0.003478976	0.003482121
8	3	1	10	0.006212638	0.004706768	0.003560437	0.003564358
9	3	2	5 + 7	0.006262211	0.004625362	0.003499945	0.003503259
10	3	2	7 + 10	0.006307630	0.004778625	0.003611802	0.003616251
11	4	1	5	0.008989306	0.006994236	0.005285005	0.005293169
12	4	1	10	0.009036972	0.007104642	0.005366648	0.005375821
13	4	2	5 + 5	0.008888109	0.006736263	0.005093241	0.005099582
14	4	2	7 + 10	0.008920953	0.006871189	0.005193077	0.005200143

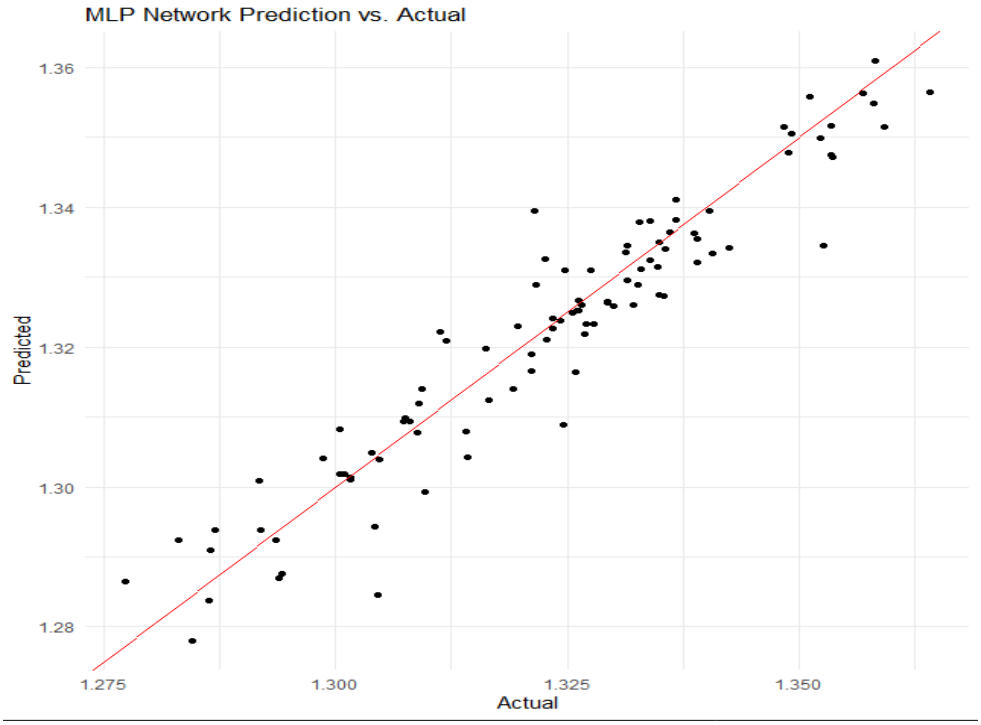
2.7 Best one-layer and best two-layer MLP models

Experiment 4 with 5 neurones in the hidden layer appears to be the best one-layer MLP from the comparison table. Compared to the other tests, it generally has the lowest statistical indices values.

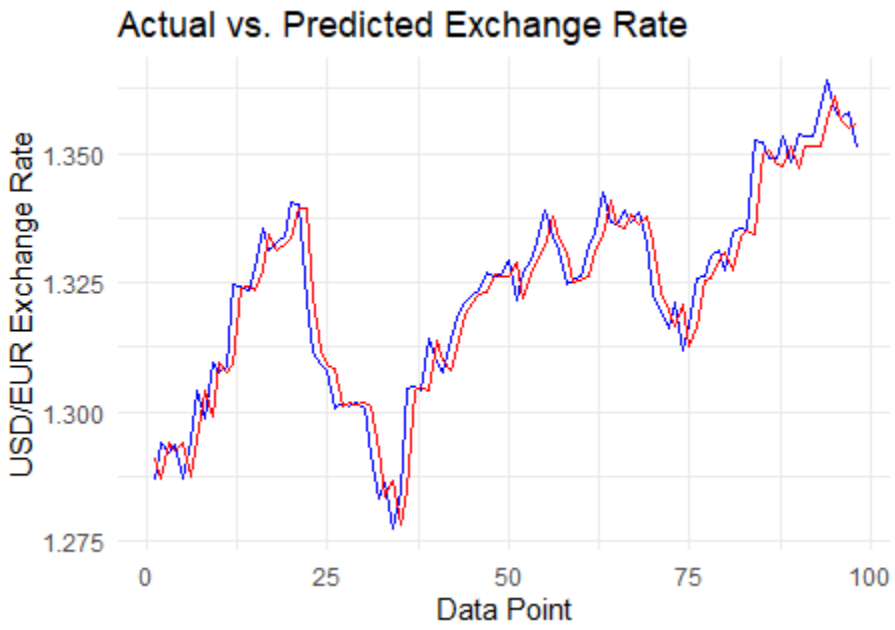
Experiment 3 appears to have the best two-layer MLP from the comparison table, with 7 and 10 neurons in its two hidden layers, respectively. When compared to the other 2-layer MLP studies, it generally has the lowest statistical indices values.

2.8 Best MLP Network Plot

Scatter Plot



Line Chart



Appendix

Partition Clustering Part Code

```
library(NbClust)

library(ggpubr)

library(factoextra)

library(ggplot2)

library(caret)

library(cluster)

library(fpc)


#import whitewine data set

whitewhine_data <- read.csv("Whitewine_v6.csv")


#calculate  total number of missing values

sum(is.na(whitewhine_data))

#Provide a summary of logicsal values which indicate the missing values or not

summary(is.na(whitewhine_data))


#store attributes from column 1 to 11

numarical_variables <- whitewhine_data[,-12]

#quality type of the

quality_variable <- whitewhine_data$quality


#create a empty vector for store found outliers during the data analysis

Outliers <- c()
```

```
#graphical summary of the distribution
boxplot(numarical_variables)

for(i in 1:11){
  column <- numarical_variables[, i]
  logical_vector <- column%in% boxplot.stats(column)$out
  current_column_out <- which(logical_vector == TRUE)
  Outliers <- c(Outliers,current_column_out)
}

eleminate <- unique(Outliers) #containing unique outliers
eleminate <- sort(eleminate) #ascending order
removed_data <- whitewhine_data[-eleminate,] #remove outliers

#after remove outliers from original data set
boxplot(removed_data[,-12])

#Scaling process for standardize the rage
scaled_removed_data <- data.frame(scale(removed_data[,-12]))

#after scaling
boxplot(scaled_removed_data)
```

```
#Automated Tools

#nbc automated tool

nbc <- NbClust(scaled_removed_data, distance = "euclidean",

            method = "kmeans", min.nc = 2, max.nc = 10, index = "all")


#Elbow method

set.seed(123)

fviz_nbclust(scaled_removed_data, kmeans, method = "wss")+

  geom_vline(xintercept = 2, linetype = 2)+

  labs(subtitle = "Elbow Method", x = "Number of clusters k",

      y= "within-cluster sum of squares(WSS)")


#Gap-static automated tool

set.seed(123)

gap_stat <- clusGap(scaled_removed_data, kmeans, K.max = 10,

                nstart = 25, B =50)

fviz_gap_stat(gap_stat)


#silhouette Method

fviz_nbclust(scaled_removed_data, kmeans, method = "silhouette")+

  labs(subtitle = "Silhouette Method")



#k-means

#k-means clustering using 2 clusters based on sacaled data
```

```
kmean_2 <- kmeans(scaled_removed_data,2)


#visualization of the k-means clustering result
fviz_cluster(kmean_2, data = scaled_removed_data,
             palette= c("#ff4500", "#836fff"),
             geom = "point", ellipse.type = "convex", ggtheme = theme_bw()
             )


kmean_2


centers <- kmean_2$centers #cluster centers
bss <- kmean_2$betweenss # amount of variation between cluster centers
tss <- kmean_2$totss # total amount variation  in the data
wss <- kmean_2$withinss # amount of variation within each cluster
kmean_2[["tot.withinss"]]
kmean_2[["betweenss"]]
kmean_2[["totss"]]


#for clustering effectiveness
ratio_bss_tss <- (bss/tss)*100


#average silhouette width
silh <- silhouette(kmean_2$cluster, dist(scaled_removed_data))
fviz_silhouette(silh, palette=c("#ff4500", "#836fff"))
```

```
#-------------------------#Principal Component Analysis#----------------------


#calculate the principal components of scaled removed data

pca <- prcomp(scaled_removed_data)

summary(pca)


#eigenvalues and eigenvectors

-pca$rotation #principal components loading

pca$sdev^2  #standard deviation of every PC


scree <- ggplot(data.frame(prop_var = pca$sdev^2/sum(pca$sdev^2),   #calculate the proportion of each
PC

                           cum_var = cumsum(pca$sdev^2/sum(pca$sdev^2)),

                           PC = 1:length(pca$sdev)), #represent the PC 1 to total

                aes(x = PC, y = cum_var)) +

 geom_line() +

 geom_point() +

 scale_x_continuous(breaks = seq(1,length(pca$sdev))) +

 xlab("Number of principal components") +

 ylab("Cumulative proportion of variance explained") +

 ggtitle("Scree plot")


scree


#get the first 7 PCs
```

```
transformed <- data.frame(pca$x[,1:7], class= removed_data[,-13])

transformed_data <- transformed[, 1:7]


#automated tools

#NbClust method

nbc_pca <- NbClust(transformed_data, distance="euclidean",

                method="kmeans", min.nc=2, max.nc=10, index="all")



#Elbow method with PCA


set.seed(26)

fviz_nbclust(transformed_data, kmeans, method="wss")+

  geom_vline(xintercept = 2, linetype = 2)+

  labs(subtitle = "Elbow Method for PCA Data",

       x='Number of Clusters K', y='within cluster sum of square')



#gap statistic method with PCA

set.seed(26)

gap_stat_pca <- clusGap(transformed_data, kmeans, K.max = 10,

                    nstart=25, B=50)

fviz_gap_stat(gap_stat_pca)


#silhouette method with PCA
```

```
fviz_nbclust(transformed_data, kmeans, method = "silhouette")+
  labs(subtitle = "Silhouette method for PCA data")


#kmeans with PCA
pca_kmean_2 <- kmeans(transformed_data,2)


fviz_cluster(pca_kmean_2, data = transformed_data,
             palette= c("#ff7f00", "#ff4500"),
             geom = "point", ellipse.type = "convex", ggtheme = theme_bw()
             )
pca_kmean_2


pca_centers <- pca_kmean_2$centers
pca_bss <- pca_kmean_2$betweenss #amount variance between cluster centroids
pca_tss <- pca_kmean_2$totss   #total amount of variation in data
pca_wss <- pca_kmean_2$withinss   # amount of variance within the cluster
pca_kmean_2[["betweenss"]]
pca_kmean_2[["tot.withinss"]]
pca_kmean_2[["totss"]]
pca_kmean_2[["centers"]]


#clustering effectiveness after PCA
pca_ratio_bss_tss <- (pca_bss/pca_tss)*100
pca_ratio_bss_tss
```

```
#average silhouette score after PCA
pca_silh <- silhouette(pca_kmean_2$cluster, dist(transformed_data))
fviz_silhouette(pca_silh, palette= c("#ff7f00", "#ff4500"))

#quality of a clustering solution is better when the index value is higher
ch_index <- calinhara(transformed_data, pca_kmean_2$cluster)
print(ch_index)
```

Financial Forecasting Part Code

```
library(ggpubr)
library(neuralnet)
library(readxl)
library(Metrics)
library(keras)
library(ggplot2)


exchangeUSD <- read_excel("ExchangeUSD.xlsx")
colnames(exchangeUSD) <- c('YYYY/MM/DD', 'Wdy', 'USD/EUR')


summary(exchangeUSD)
boxplot(exchangeUSD[,3])



#Normalize function
normalize <- function(x){
  return ((x - min(x)) / (max(x) - min(x)))
}
#Unnormalize function
unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}
#normalizing the data
exchangeUSD_norm <- as.data.frame(lapply(exchangeUSD[,3], normalize))
```

```
summary(exchangeUSD_norm)
boxplot(exchangeUSD_norm)

train_data <- exchangeUSD_norm[1:400 ,]
test_data <- exchangeUSD_norm[401:500 ,]

#create I/O matrix function for testing and training
create_io_df <- function(data, num_cols){
  dataInput <- matrix(, nrow = 0, ncol = num_cols)
  dataOutput <- c() # initialize vector for store the output

  for(i in 1:length(data)){ #start from the first row
    lastValue <- i + (num_cols - 1)
    if(lastValue+1>length(data)){
      break
    }
    input <- data[i:lastValue]
    output <- data[lastValue+1]

    dataInput <- rbind(dataInput, input)
    dataOutput <- append(dataOutput, output)
  }
  dataIoDf <- cbind(as.data.frame(dataInput), dataOutput) #combine the input matrix with output vector
  return(dataIoDf)
```

```
}

#create io_matrix for training and testing for t-4
create_t4IoDf <- function(data, num_cols=5) {
  dataInput <- matrix( ,nrow = 0, ncol = num_cols) #matrix to extract input values
  dataOutput <- c() #vector to store output value

  for (i in 4:length(data)) { #starting from 4th row, use a lag of 4 days in input vector
    lastValue <- i + (num_cols - 2)
    t4_value <- (lastValue - 3)
    if (lastValue + 1 > length(data)) {
      break
    }
    input <- data[i:lastValue-1]
    t4_input <- data[t4_value]
    input <- append(input, t4_input)
    output <- data[lastValue + 1]

    dataInput <- rbind(dataInput, input) #add the new input vector to input matrix
    dataOutput <- append(dataOutput, output) #add new output value to output vector
  }

  data_4tIoDf <- cbind(as.data.frame(dataInput), dataOutput)
  return(data_4tIoDf)
}
```

```
#creating io_matrix for training
training_io_1 <- create_io_df(train_data, 1)
training_io_2 <- create_io_df(train_data, 2)
training_io_3 <- create_io_df(train_data, 3)
training_io_4 <- create_t4IoDf(train_data, 4)

#creating io_matrix for testing
testing_io_1 <- create_io_df(test_data, 1)
testing_io_2 <- create_io_df(test_data, 2)
testing_io_3 <- create_io_df(test_data, 3)
testing_io_4 <- create_t4IoDf(test_data, 4)



#create neural network func
create_neural_network <- function(data, input_cols, hidden_layers, act_fct) {
  formula <- as.formula(paste("dataOutput ~", paste(input_cols, collapse = " + ")))
  set.seed(12345)
  neural_network_model <- neuralnet(formula, data = data, hidden = hidden_layers,
                    act.fct = act_fct, linear.output = TRUE) #create neural network model
  return(neural_network_model)
}


# Training Models
input1_nn_1 <- create_neural_network(training_io_1, c("V1"), c(5), "logistic") # with logistic sigmoid function
```

```
input1_nn_2 <- create_neural_network(training_io_1, c("V1"), c(7,7), "tanh")   # with hyperbolic tangent function

input1_nn_3 <- create_neural_network(training_io_1, c("V1"), c(7,10), "logistic")

input2_nn_1 <- create_neural_network(training_io_2, c("V1", "V2"), c(5), "logistic")

input2_nn_2 <- create_neural_network(training_io_2, c("V1", "V2"), c(5,7), "tanh")

input2_nn_3 <- create_neural_network(training_io_2, c("V1", "V2"), c(7,10), "logistic")

input3_nn_1 <- create_neural_network(training_io_3, c("V1", "V2", "V3"), c(5), "logistic")

input3_nn_2 <- create_neural_network(training_io_3, c("V1", "V2", "V3"), c(10), "logistic")

input3_nn_3 <- create_neural_network(training_io_3, c("V1", "V2", "V3"), c(5,7), "tanh")

input3_nn_4 <- create_neural_network(training_io_3, c("V1", "V2", "V3"), c(7,10), "tanh")

input4_nn_1 <- create_neural_network(training_io_4, c("V1", "V2", "V3", "V4"), c(5), "logistic")

input4_nn_2 <- create_neural_network(training_io_4, c("V1", "V2", "V3", "V4"), c(10), "tanh")

input4_nn_3 <- create_neural_network(training_io_4, c("V1", "V2", "V3", "V4"), c(5,5), "logistic")

input4_nn_4 <- create_neural_network(training_io_4, c("V1", "V2", "V3", "V4"), c(7,10), "logistic")


#testing and evaluating function for NN model

evaluating_neural_network <- function(ordern_neural_network, ordern_test_io, cons_data = exchangeUSD) {

 #test the neural networks with test data

 number_col_i <- ncol(ordern_test_io)#number of columns in test I/O data

 testing_data <- data.frame(ordern_test_io[,1:(number_col_i - 1)])

 set.seed(12345)

 ordern_nn_results <- compute(ordern_neural_network, testing_data) # uses the trained NN to compute predictions

 # results of NN

 results <- data.frame(actual = ordern_test_io$dataOutput,

                       prediction = ordern_nn_results$net.result) #actual and predicted data frame
```

```
resultsMin <- min(cons_data$"USD/EUR") #store the minimum rate from the dataset

resultsMax <- max(cons_data$"USD/EUR") #store the maximum rate from the dataset


#unnormalized the predicted and actual outputs

comparison <- data.frame(

  predicted = unnormalize(results$prediction, resultsMin, resultsMax),

  actual = unnormalize(results$actual, resultsMin, resultsMax)

)


#1st layer weights

weights_layer1 <- ordern_neural_network$weights[[1]]

#calculate the number of hidden layers in NN

hidden_layer_no <- length(weights_layer1) - 1

if(hidden_layer_no == 2) {

  hidden_layer_config_1 <- ncol(ordern_neural_network[["weights"]][[1]][[1]])# number of neuron in the 1st hidden layer

  hidden_layer_config_2 <- ncol(ordern_neural_network[["weights"]][[1]][[2]])# number of neurons in the second layer

  hidden_layer_config <- c(hidden_layer_config_1, hidden_layer_config_2)

}else {

  hidden_layer_config_1 <- ncol(ordern_neural_network[["weights"]][[1]][[1]])

  hidden_layer_config <- c(hidden_layer_config_1)#combine the number of neurons in the hidden layer

}


#statistic indices
```

```
  RMSE <- rmse(comparison$actual, comparison$predicted) #Root Mean Square Error

  MAE <- mae(comparison$actual, comparison$predicted) # Mean Absolute Error

  MAPE <- mape(comparison$actual, comparison$predicted) # Mean Absolute Percentage Error

  sMAPE <- smape(comparison$actual, comparison$predicted) # Symmetric Mean Absolute Percentage Error


  #crate a data frame to store the evaluation matrics with information about the NNA

  evaluation_metrix <- data.frame(No_of_inputs = (number_col_i-1), No_of_hidden_layers = hidden_layer_no,

                  Hidden_layer_cofig = paste(hidden_layer_config, collapse = " + "),

                  RMSE = RMSE, MAE = MAE, MAPE = MAPE, sMAPE = sMAPE)

  plot(ordern_neural_network) #Generate the plots


  return(list(evaluation_metrix = evaluation_metrix, comparison = comparison))

}




training_models <- list(input1_nn_1, input1_nn_2, input1_nn_3, input2_nn_1, input2_nn_2, input2_nn_3,

          input3_nn_1, input3_nn_2, input3_nn_3,input3_nn_4, input4_nn_1, input4_nn_2, input4_nn_3,

          input4_nn_4)


#list of testing I/O matrices for each NN models

testing_data <- list(testing_io_1, testing_io_1, testing_io_1,

                testing_io_2, testing_io_2, testing_io_2,

                testing_io_3, testing_io_3, testing_io_3,
```

```
                testing_io_3, testing_io_4, testing_io_4,

                testing_io_4,testing_io_4)


#testing and evaluating NN models with testing_io matrix

evaluating_results <- list()

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

  evaluating_results[[i]] <- evaluating_neural_network(training_models[[i]], testing_data[[i]])

}


#create the comparison dataframe with indices

eval_metrix_comparison_df <- do.call(rbind, lapply(evaluating_results, function(x) x$evaluation_metrix))



# Identify the best MLP network based on evaluation results

best_model_index <- which.min(eval_metrix_comparison_df$RMSE)  # Assuming RMSE is the metric for
selection

best_model <- training_models[[best_model_index]]

best_model_eval <- evaluating_results[[best_model_index]]


# Prepare data for visualization

results <- best_model_eval$comparison



ggplot(results, aes(x = actual, y = predicted),col) +

  geom_point() +

  geom_abline(slope = 1, intercept = 0, color = "red") +  # Add a line of perfect prediction
```

```
  labs(x = "Actual", y = "Predicted", title = "MLP Network Prediction vs. Actual") +
  theme_minimal()

ggplot(results, aes(x = 1:nrow(results), y = actual),) +
  geom_line(color = "blue") +
  geom_line(aes(y = predicted), color = "red") +
  labs(x = "Data Point", y = "USD/EUR Exchange Rate", title = "Actual vs. Predicted Exchange Rate") +
  theme_minimal()
# Display Statistical Indices
print(best_model_eval$evaluation_metrix)
```

References

Brownlee, J. (2018). *How to Use Statistics to Identify Outliers in Data*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/. [Accessed 27 April 2024]

MSc, F.E.O. (2023). *Unsupervised Learning in R: Determination of Cluster Number*. [online] Medium. Available at: https://medium.com/@ozturkfemre/unsupervised-learning-determination-of-cluster-number-be8842cdb11. [Accessed 27 April 2024]

Stöttner, T. (2019). *Why Data should be Normalized before Training a Neural Network*. [online] Medium. Available at: https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d. [Accessed 29 April 2024]

www.sciencedirect.com. (n.d.). *Multilayer Perceptron - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/veterinary-science-and-veterinary-medicine/multilayer-perceptron. [Accessed 30 April 2024]