2021

5DATA002W - ML & DM – Coursework

TOMASZ WASOWSKI – W1684891

PARTITIONING CLUSTERING

DISCUSSION OF METHODOLOGIES USED FOR DIMENSIONALITY REDUCTION

To explore methods used in input dimensionality reduction, we must first understand why it is beneficial to the outcome of the clustering we want to perform.

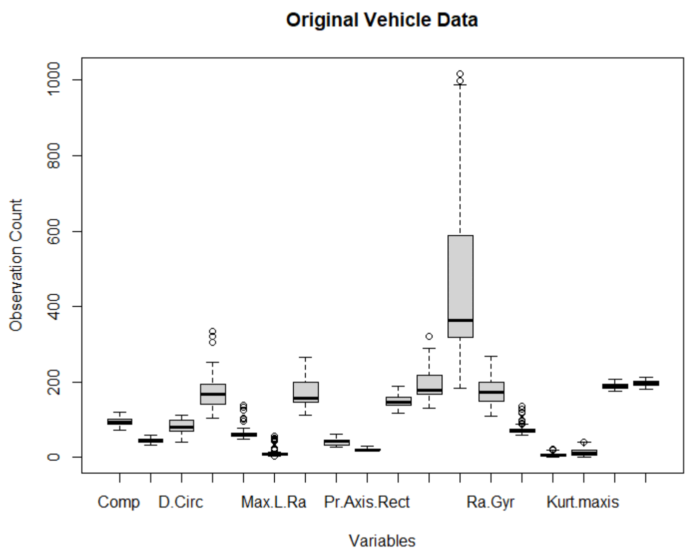
Clustering is dependent on a measure of distance, in our case the Euclidean distance. Distance measures tend to produce non-intuitive results with highly dimensional data (Domingos, 2012). As such, in order to strengthen the effectiveness of our distance metric, we would ideally reduce the dimensionality of the input data prior to performing any Clustering.

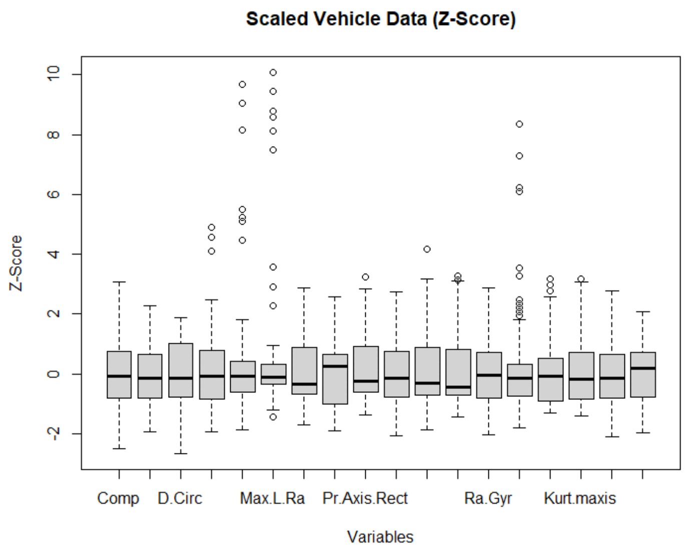
One common method of dimensionality reduction is Principal Component Analysis (PCA). This technique aims to increase interpretability while minimising information loss (Jolliffe and Cadima, 2016). It does so by converting a set of variables that could be correlated into a set of uncorrelated variables called Principal Components. These components will be ordered by variance, with the first component accounting for the most data variability. This ultimately allows us to use a smaller subset of variables for our clustering while retaining as much accuracy as we can.

Another method of dimensionality reduction is Feature Selection. There are different types of feature selection methods, but the two main ones consist of wrapper methods and filter methods. Wrapper methods wrap around a machine learning model and evaluate it with different subsets of input variables, selecting the one which provides the best performance. Filter methods score features by correlation and variability and aim to choose a feature subset that is the most predictive (Brownlee, 2020).

PRE-PROCESSING TASKS

To begin this process, I decided to plot the data to see what I am working with. Below is a boxplot of the original, untouched data (left) as well as the same data normalised with z-score (right):



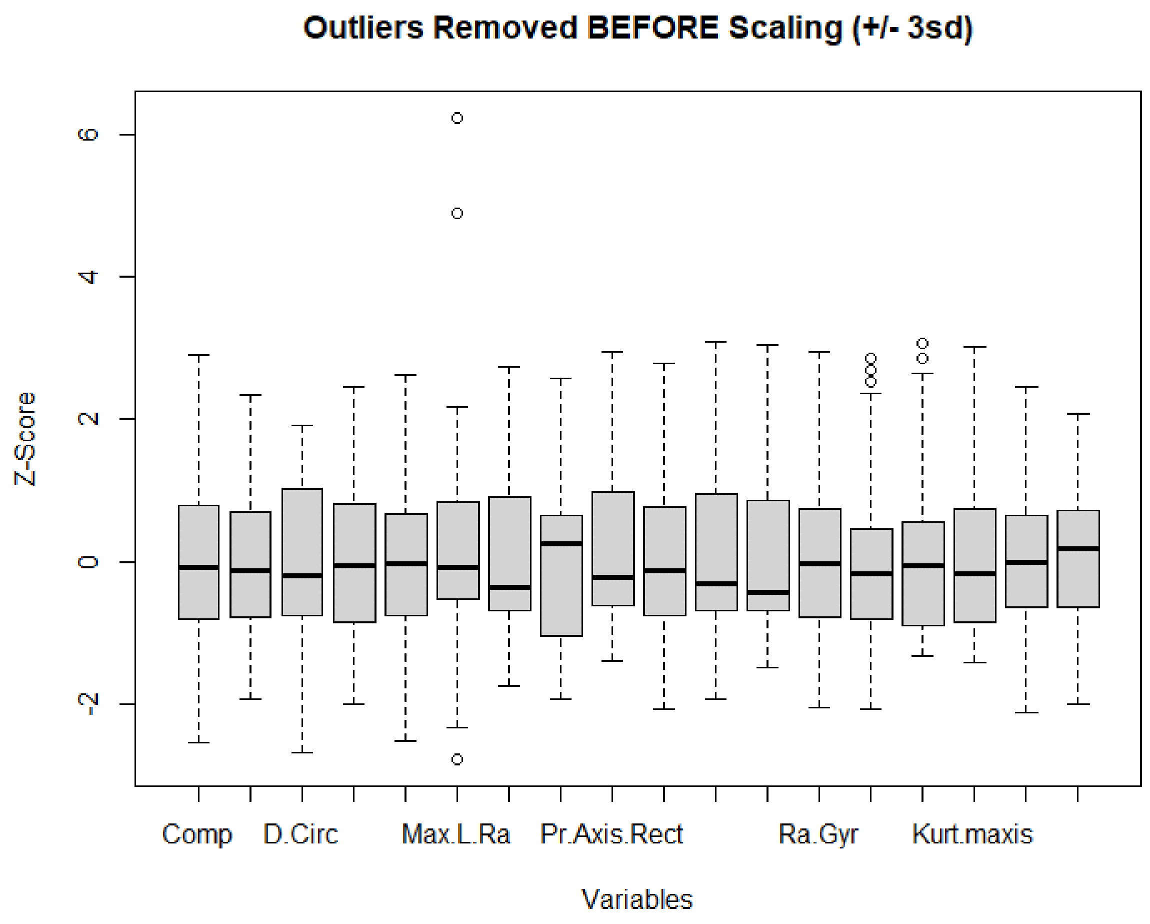


As you can see, especially from the z-score data (right), there are some obvious outliers in this data set.

Our next step would be to remove these outliers, as clustering can be very sensitive to those while calculating the Euclidean Distances. After doing some research on pre-processing data prior to clustering (EduPristine, 2015), I have concluded that removing outliers can be quite a tricky subject, as it often depends heavily on context. In order to explore all my options, I will run my clustering with datasets that have outliers removed BEFORE scaling, and also with datasets that have outliers removed AFTER clustering. The reason for this, is that clustering in essence relies on scaling, and if I were to scale my data before removing outliers, then I would be left with data which has an incorrect scaling, since the outliers used to calculate either the min-max or the mean and standard deviation for this scaling are now gone. However, if I scale the data after the outliers are removed, the scaling might show new 'outliers' that I wouldn't have considered outliers the first time I have removed them. By trying both approaches I will be able to directly compare the clustering results and will be able to see the impact that removing outliers before or after has on the accuracy of the model.

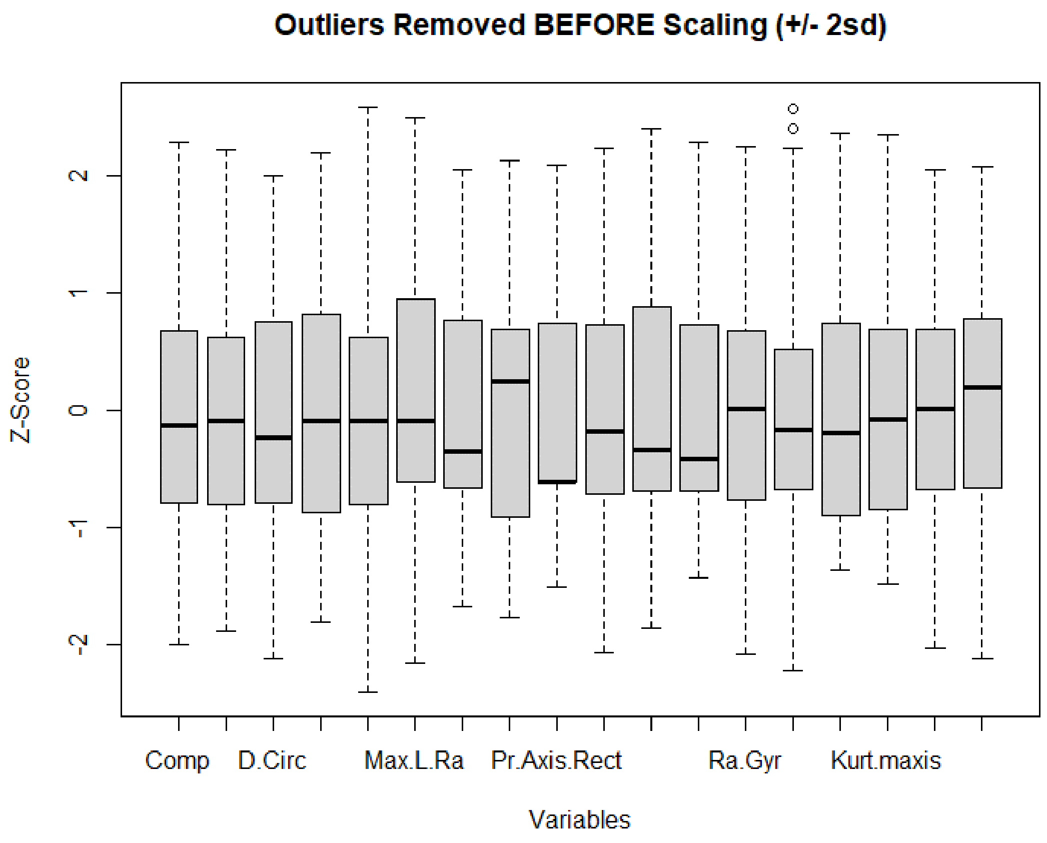
I will start by trying to remove the outliers BEFORE scaling the data. In order to remove these outliers, we have to first detect them and also define what an outlier is. Following research on the topic I decided that I will detect outliers by calculating the data's z-scores (ektamaini, no date). As we can see from the prior boxplot, the outliers are easily detectable after z-score normalisation. Next, we must decide how we define our outliers. Following the same source, I gathered that defining anything above or below +/- 3 standard deviations is a sensible approach (ektamaini, no date).

However, since I want to remove outliers before scaling the data, I simply use these z-scores to decide which rows of the dataset can be considered outliers, and remove these rows from the original, un-scaled data. Once this is done, I normalise the now-outlier-free data by calculating the z-scores again. This approach removed 22 outliers and left me with 97% of the data and the following boxplot:



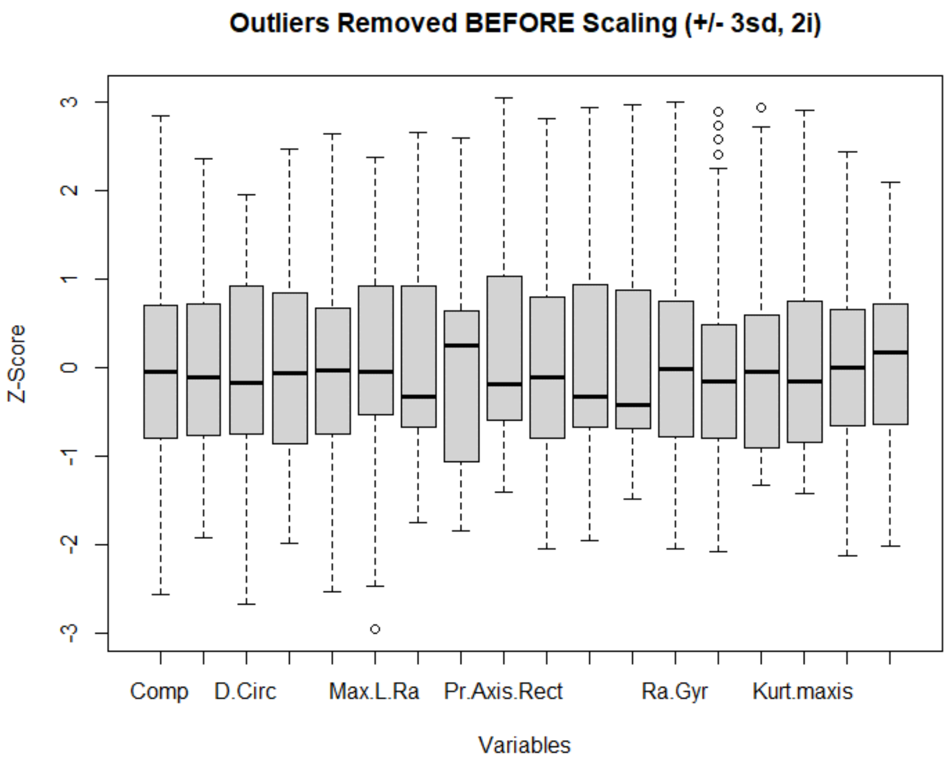
As you can see, after removing the outliers and scaling the data, we can see that there are still a few points which seem anomalous. After doing some more research, I came up with two solutions to this problem. First, I can try to define the outliers more broadly during removal. I could do this by setting the cut-off point for outliers at +/- 2 standard deviations, as opposed to 3. This would produce a much clearer data; however, it will also remove significantly more observations.

Having tried this method, I managed to get a very clear set of data by removing 173 outliers, leaving me with 80% of data and the following boxplot:



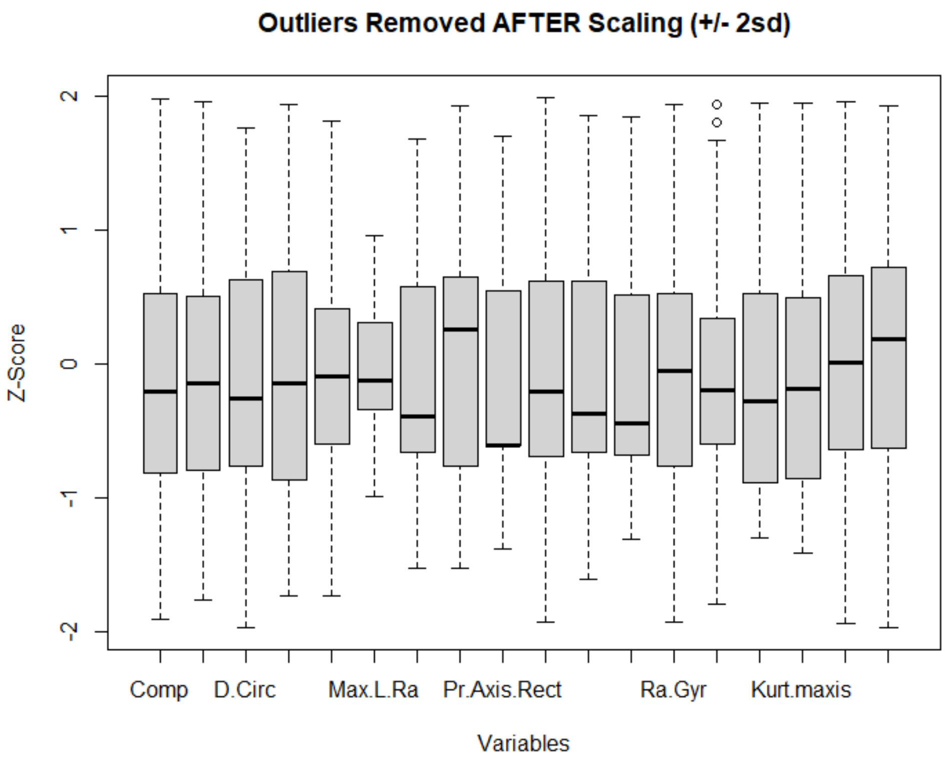
As you can see, this boxplot looks much clearer than the previous one. However, with just over 20% of the data removed, it seemed to me to be a bit too excessive. As such, I decided to try another approach based on a paper I found which talks about iterative outlier removal (Parrinello et al., 2016). I decided that I will remove the outliers at +/- 3 standard deviations again but will repeat this process twice. Since the scaling is only used to detect these outliers when removing them from the original dataset, the definition of what an outlier is will change dynamically with each iteration of this process.

To keep as much of the data as possible, I decided not to run this process more than twice. This method resulted in a clear data set with only 36 outliers removed. That means, that we are left with 96% of the data and the following boxplot:

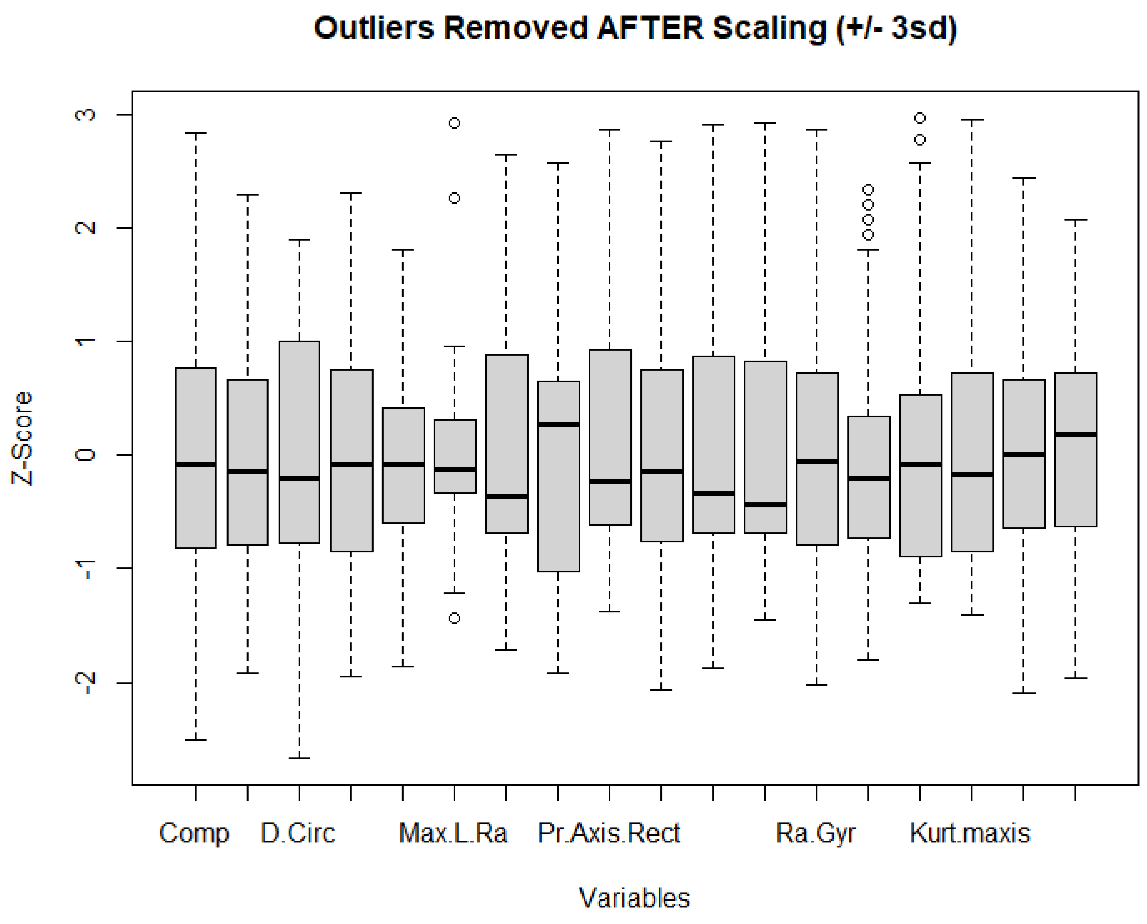


As can be seen, both of these methods produce a good set of data. At this stage, I am not sure which data set will produce better clustering results, so I will also attempt to produce data sets where the outliers are removed AFTER scaling, for comparison against the above.

I decided to stick to the same values used to determine an outlier, which are +/- 2 and +/- 3 standard deviations when z-score normalised. First, removing the outliers at +/- 2sd AFTER scaling the data also removes 173 outliers and the boxplot looks as follows:



Next, I removed the outliers at +/- 3sd AFTER scaling the data, which also removed 22 outliers and produced the boxplot below:



Though it may seem a little excessive that I have created so many different methods for removing outliers both before and after scaling, I feel like it was a necessary step in this coursework. Having done the research, I found that outlier removal is really dependant on the scenario, and as I haven't got too much experience with machine learning, I decided to try every approach and to decide from the results which approach worked the best. To supplement all of this, I will also be running the future sections for scaled data with no outliers removed, just for comparison.

One case that I have not looked at would be to remove the outliers by hand. This would involve removing every singe 'o' from the boxplot by either checking the row index of each outlier and removing the row from the data, or by setting up upper and lower boundaries individually for every single attribute. As there are 18 attributes, I decided to remove outliers in a more 'automatic' fashion, by scaling the data and setting the same boundary for all attributes. Removing these outliers by hand would probably provide better clustering results but would also be much more time consuming and not very scalable, considering that our data could have many more attributes than 18.

In addition, writing this report after I've already done some work on the latter section, I can see that these different methods do indeed give different results. I have tried each and every one of these methods, so please refer to the R transcript at the end of the Clustering section to see my work process through this task.

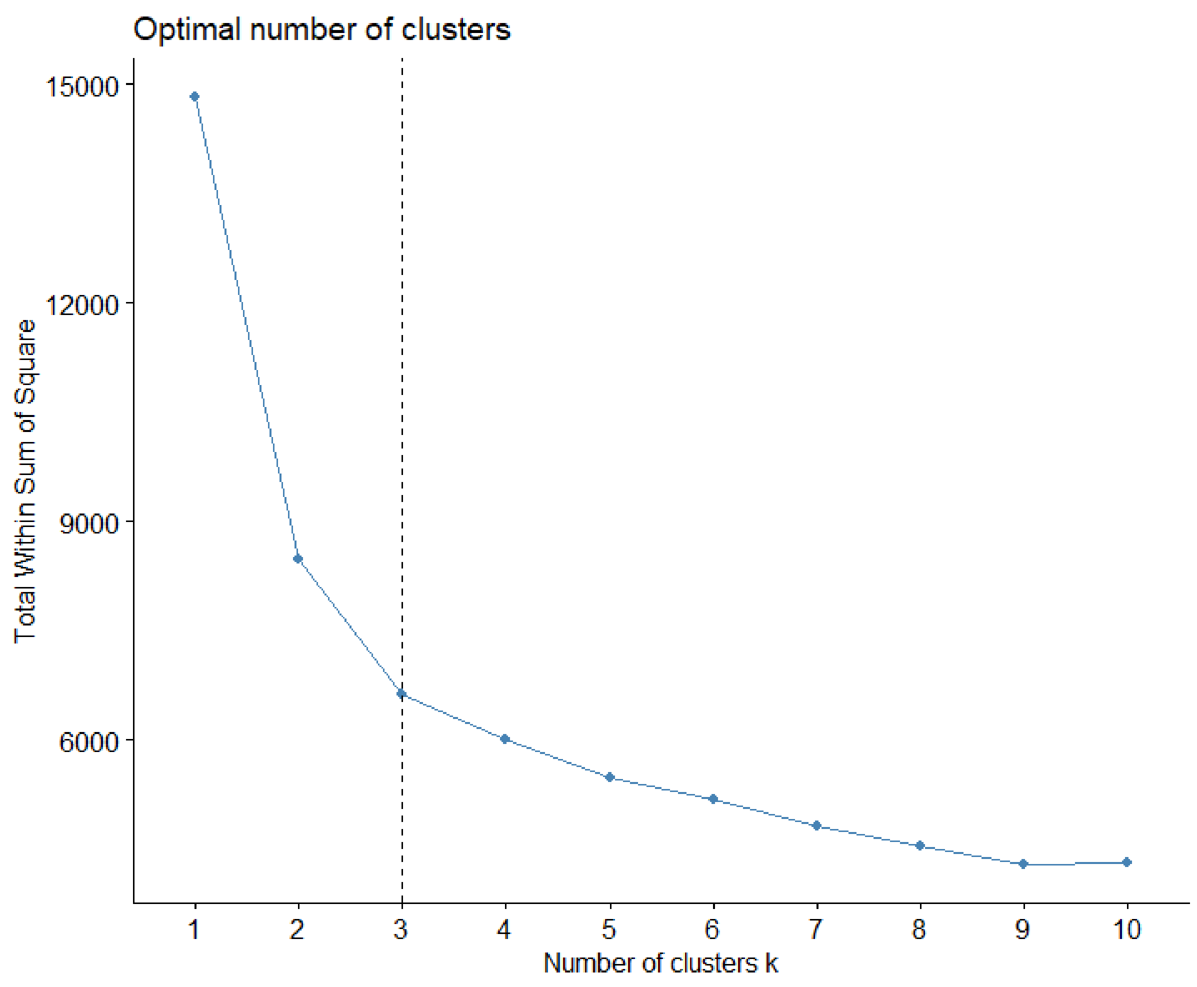
NUMBER OF CLUSTERS (K) DEFINITION

After doing a bit of research on determining the optimal number of clusters (Soetewey, 2020), I found that there are 4 main methods for doing this:

- Elbow Method
- Silhouette Method
- Gap Statistic Method
- 30 Indices Method

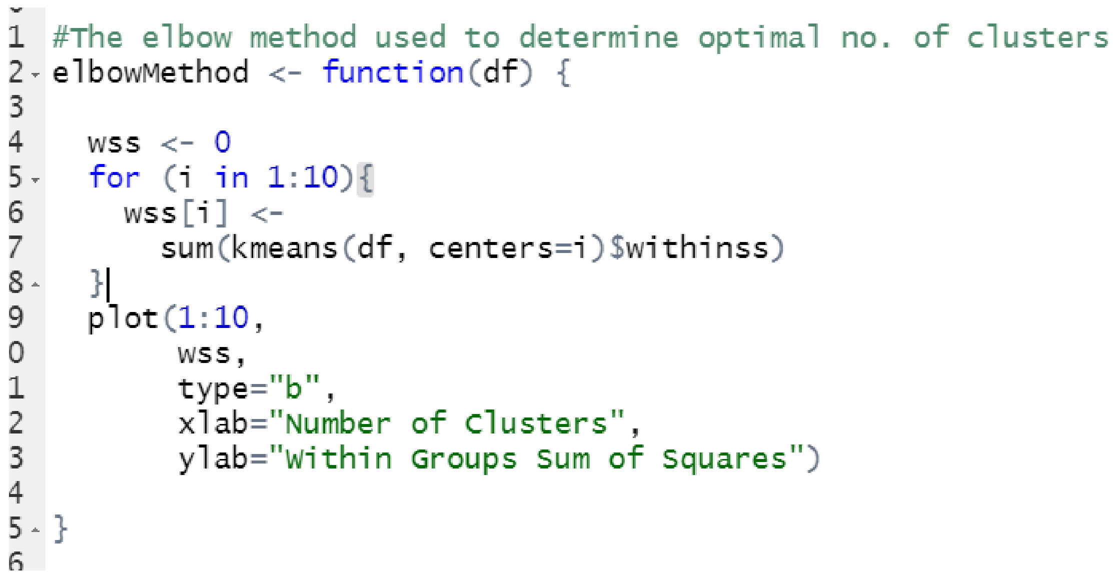
I will go over each method and talk a little about how I performed it on each of the 6 datasets generated from the previous section. I will also include some plots from a single dataset to visualise what the results looked like. The final optimal cluster numbers from across all methods performed on all datasets will be presented in a table at the end of this section. For methodology and plots on each one, please refer to the R transcript at the end of the clustering section of this report.

Elbow Method – This method revolves around the total within-cluster sum of square to determine the optimal number of clusters. The place in the plot that is the 'knee', or the bend, likely indicates an appropriate number of clusters. At that location, adding another cluster does not provide much of an improvement towards the partition. Here is an example of this method used with the dataset which had outliers before scaling at +/- 3sd shown below:

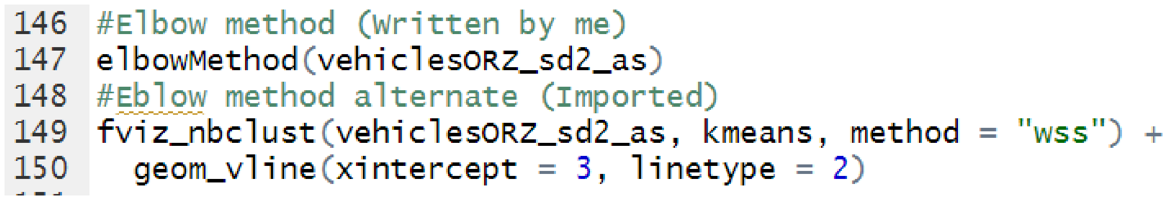


This method can be a little vague, as we have to decide where the 'knee' of this plot is. In the plot above, I decided that k = 3 appears to be an appropriate number of clusters, but there is certainly also a case to be made for k = 4 or potentially even k = 5.

I have written two different methods for the elbow method. One done by hand and another one which I've imported from a library.

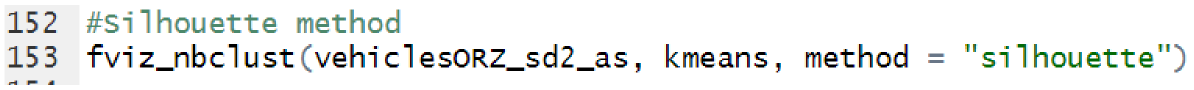


```
#The elbow method used to determine optimal no. of clusters
elbowMethod <- function(df) {

  wss <- 0
  for (i in 1:10){
    wss[i] <-
      sum(kmeans(df, centers=i)$withinss)
  }
  plot(1:10,
       wss,
       type="b",
       xlab="Number of Clusters",
       ylab="Within Groups Sum of Squares")

}
```



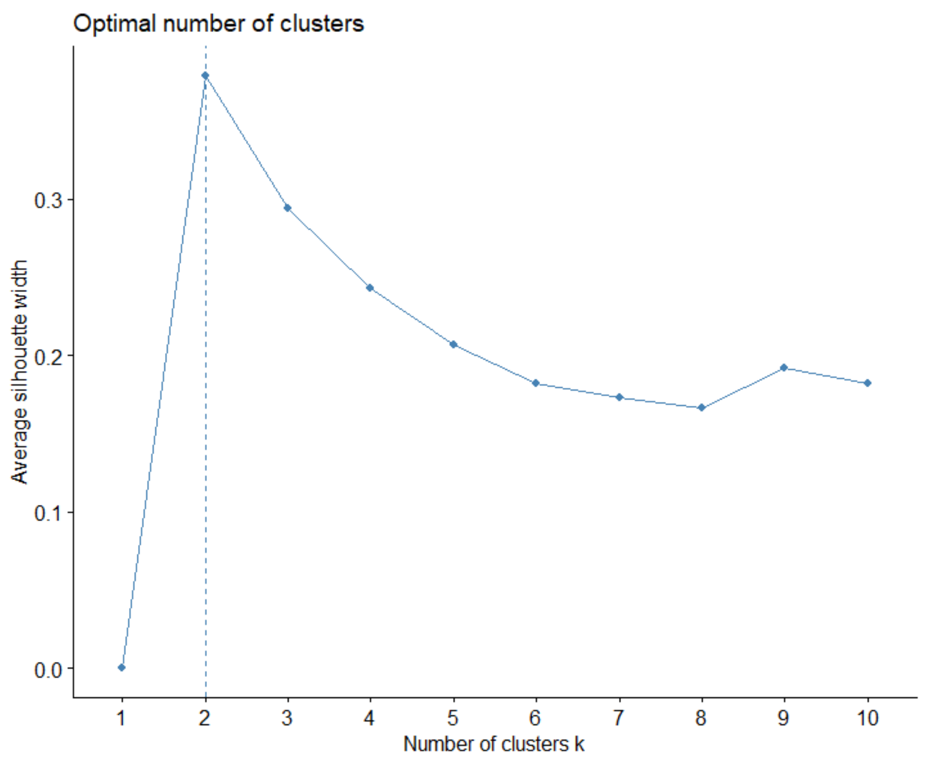
```
#Elbow method (Written by me)
elbowMethod(vehiclesORZ_sd2_as)
#Eblow method alternate (Imported)
fviz_nbclust(vehiclesORZ_sd2_as, kmeans, method = "wss") +
  geom_vline(xintercept = 3, linetype = 2)
```

Silhouette Method – An alternate to the elbow method which measures clustering quality by determining how well each point is positioned in its cluster. Usually, the number of clusters is denoted by the high point on the plot. For this method, I called a function from the same package imported for the elbow method:



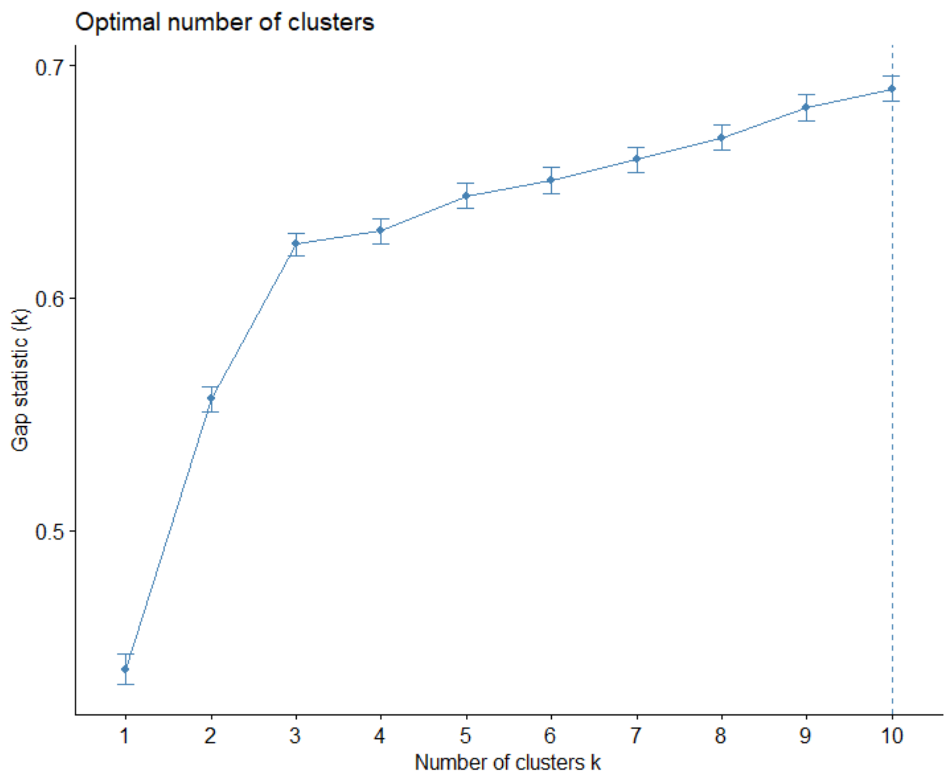
```
#Silhouette method
fviz_nbclust(vehiclesORZ_sd2_as, kmeans, method = "silhouette")
```

For an example of this, below is the Silhouette method performed on the same dataset as our previous method, one with outliers removed before scaling at +/- 3sd:



This method seems to suggest that 2 clusters would be appropriate.

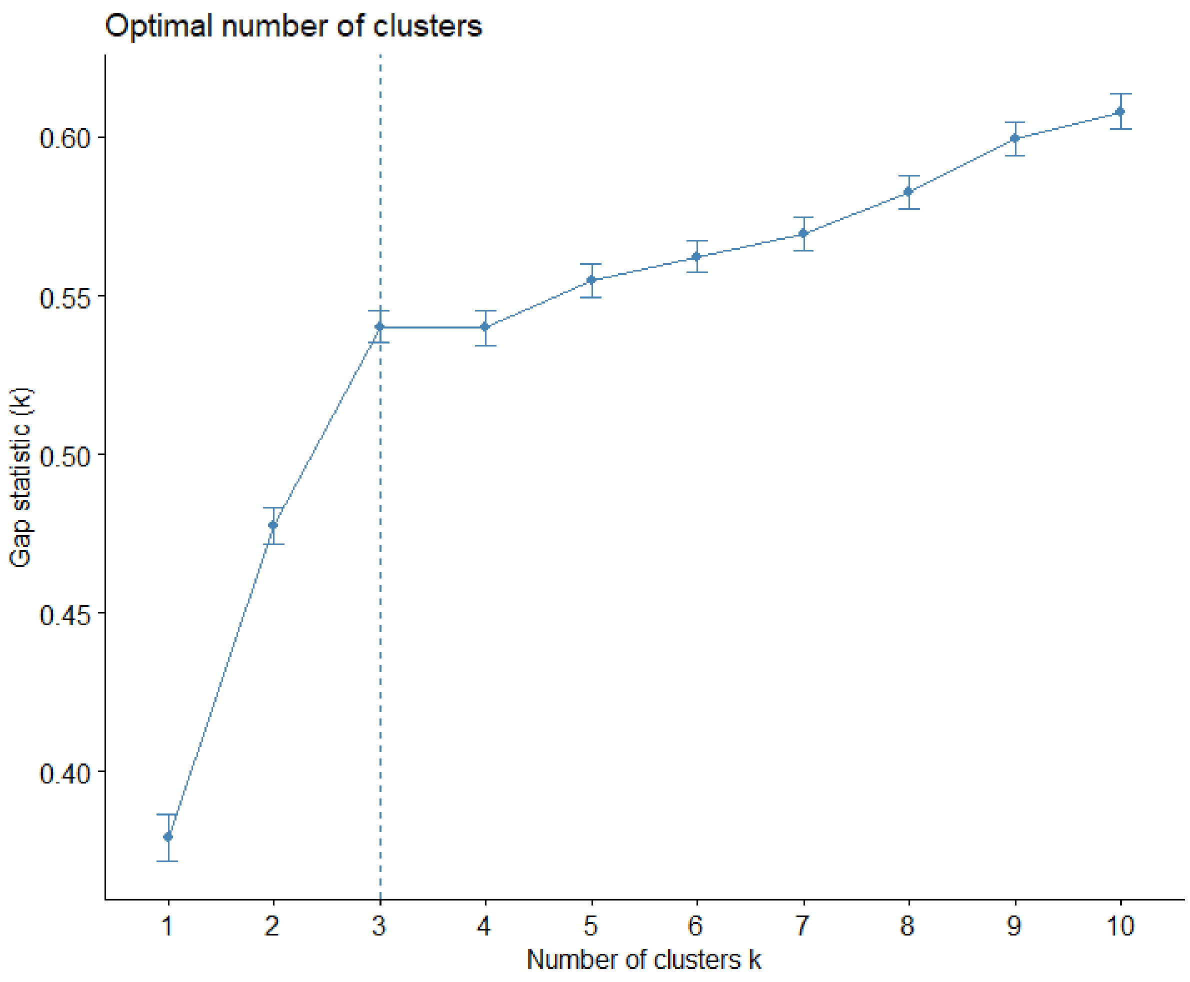
Gap Statistic Method – This is a method which calculates a gap statistic which compares total within intra-cluster variation for different amounts of clusters with their expected values under null reference distribution of the data (Datanovia, no date). An example of this method used with the same dataset as in the previous two methods can be seen below:



As you can see, for this data set the method determines that 10 clusters are the most appropriate for the data. It doesn't seem reasonable knowing the number of original classes in the data, however, this method seems to work better with different datasets. For example, running this method with the dataset where outliers were removed AFTER scaling the data at +/- 3sd, the method suggests that 3 clusters are optimal.

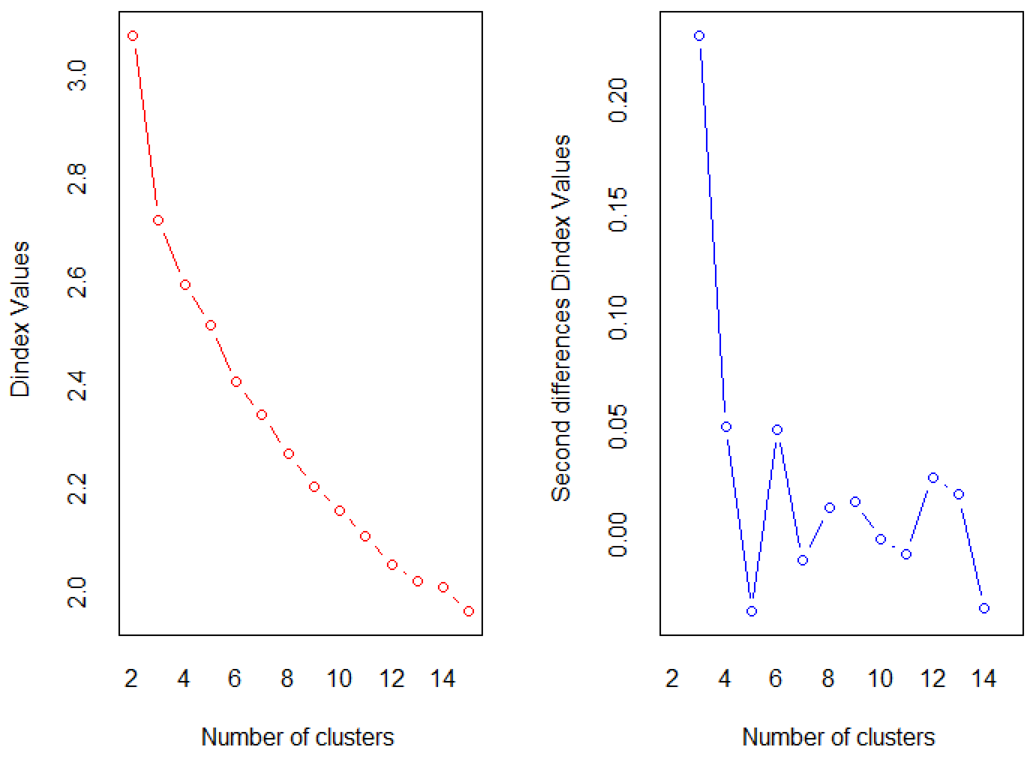
The code used for the gap statistic is another function call from an imported package:

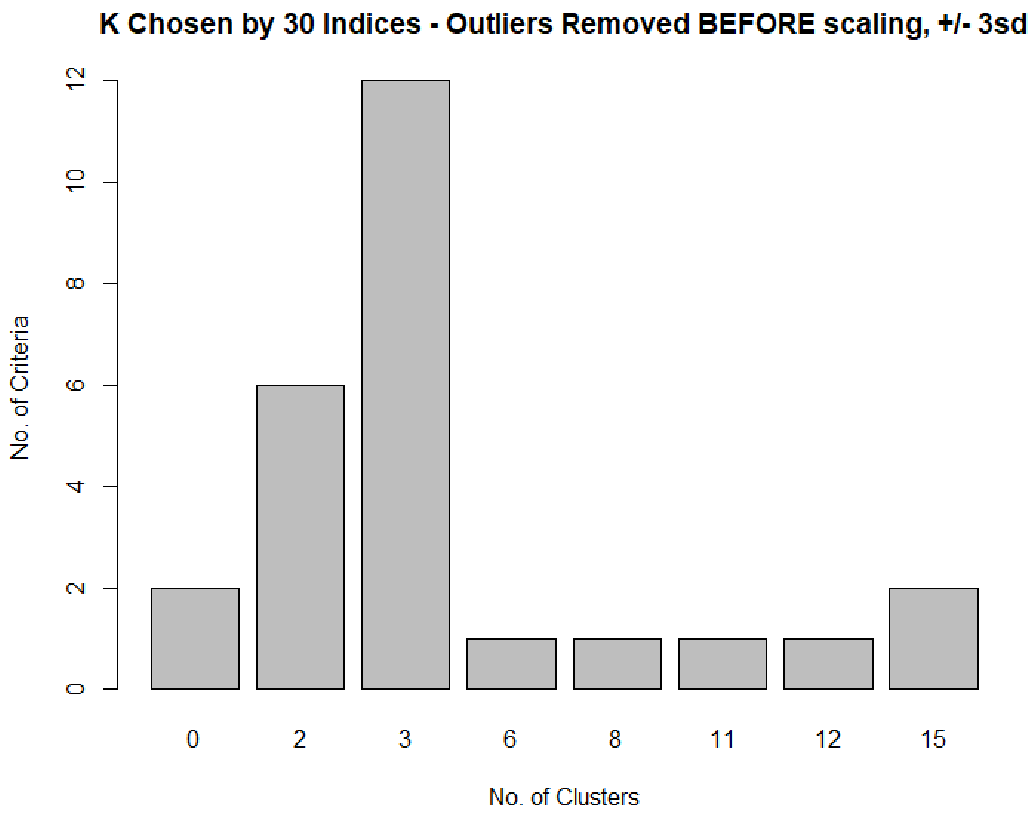
```
#Gap Statistic method
set.seed(100)
fviz_nbclust(vehiclesORZ_sd2_as, kmeans, nstart = 25,  method = "gap_stat", nboot = 50)
```



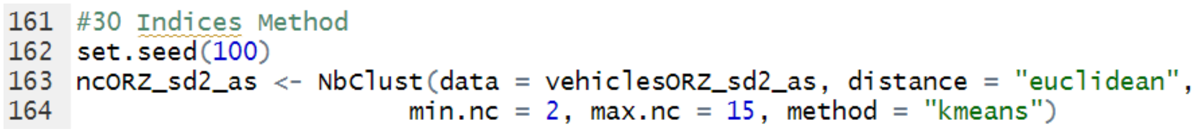
This is probably due to the fact that the dataset which was scaled after removing outliers in the first iteration still had a few data points with z-scores of 6-8sd, since the scaling was re-calculated. As such, that dataset might not be the best one to use. One last thing to mention about the gap method is that it is important to set a seed, as the method uses randomised values, and a seed allows for our result to be reproducible.

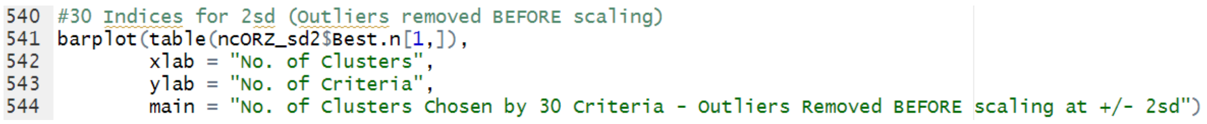
NbClust() (30 Indices) Method – The final method includes using the NbClust() method, which provides 30 indices for choosing the best value of k (Soetewey, 2020). For this method, the optimal number of clusters is usually decided by the value of k which was chosen by a majority of the 30 indices. Similarly, to the gap statistic method, this method also uses random values. As such, we need to set a seed so that our results can be reproducible. An example of this method used with the dataset with outliers removed before scaling at +/- 3sd, along with the results it produced can be seen in the plots displayed below:





As can be seen from the above plot, the majority of criteria has determined that 3 clusters would be optimal for this dataset. I will paste the code for this method and visualisation of the results below.


```
#30 Indices Method
set.seed(100)
ncORZ_sd2_as <- NbClust(data = vehiclesORZ_sd2_as, distance = "euclidean",
                        min.nc = 2, max.nc = 15, method = "kmeans")
```


```
#30 Indices for 2sd (Outliers removed BEFORE scaling)
barplot(table(ncORZ_sd2$Best.n[1,]),
        xlab = "No. of Clusters",
        ylab = "No. of Criteria",
        main = "No. of Clusters Chosen by 30 Criteria - Outliers Removed BEFORE scaling at +/- 2sd")
```

As with the previous methods, this method produces a different outcome for each of the different datasets. As a result, I decided to put all the results for all of these methods performed on each dataset in a table.

Optimal Number of Clusters Determined per Method per Dataset

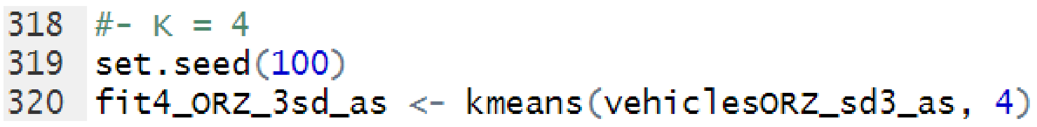
	Dataset					
Method	**Only Scaling**	**OR AS +/- 2sd**	**OR AS +/- 3sd**	**OR BS +/- 2sd**	**OR BS +/- 3sd 1i**	**OR BS +/- 3sd 2i**
Elbow	3, 7	3	3	3	3	3
Silhouette	2, 4, 6	2	2	2	2	2
Gap Statistic	3	9	3	9	10	10
30 Indices	2	2	3	3	3	2, 3
Majority	3	2	3	3	3	2, 3

*OR - Outliers removed
*AS/BS - After/Before scaling
*sd - Standard deviation
*i - Iteration

As can be seen from the above table, for most of the datasets the majority of methods suggest that 3 is the optimal value of K, with a few exceptions where the optimal number of clusters could instead / also be 2. This said, I would just like to mention that different methods of outlier removal had a great effect on the Gap Statistic method, causing it to occasionally suggest a really high number of clusters. I decided to take these results into consideration while calculating the majority for the sake of consistency.

K-MEANS ANALYSIS

I have performed my k-means clustering with the following code and have run it for each outlier-removed dataset for different values of k; the values I chose to cluster for are 2, 3 and 4. The reason for this, is that 2 and 3 were the most commonly determined optimal numbers of clusters by our methods in the section above, and 4 because that is the actual amount of classes we have in our dataset.

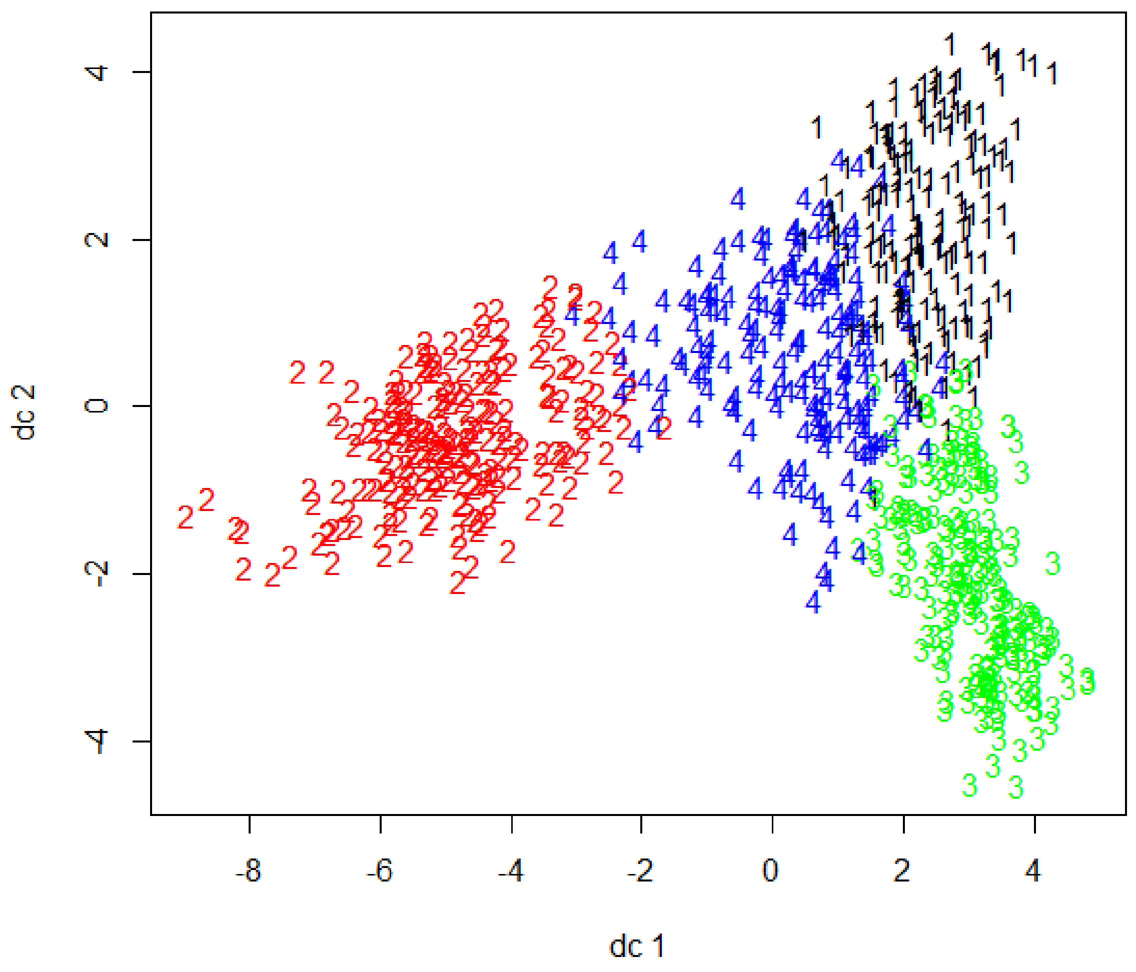

```
#- K = 4
set.seed(100)
fit4_ORZ_3sd_as <- kmeans(vehiclesORZ_sd3_as, 4)
```

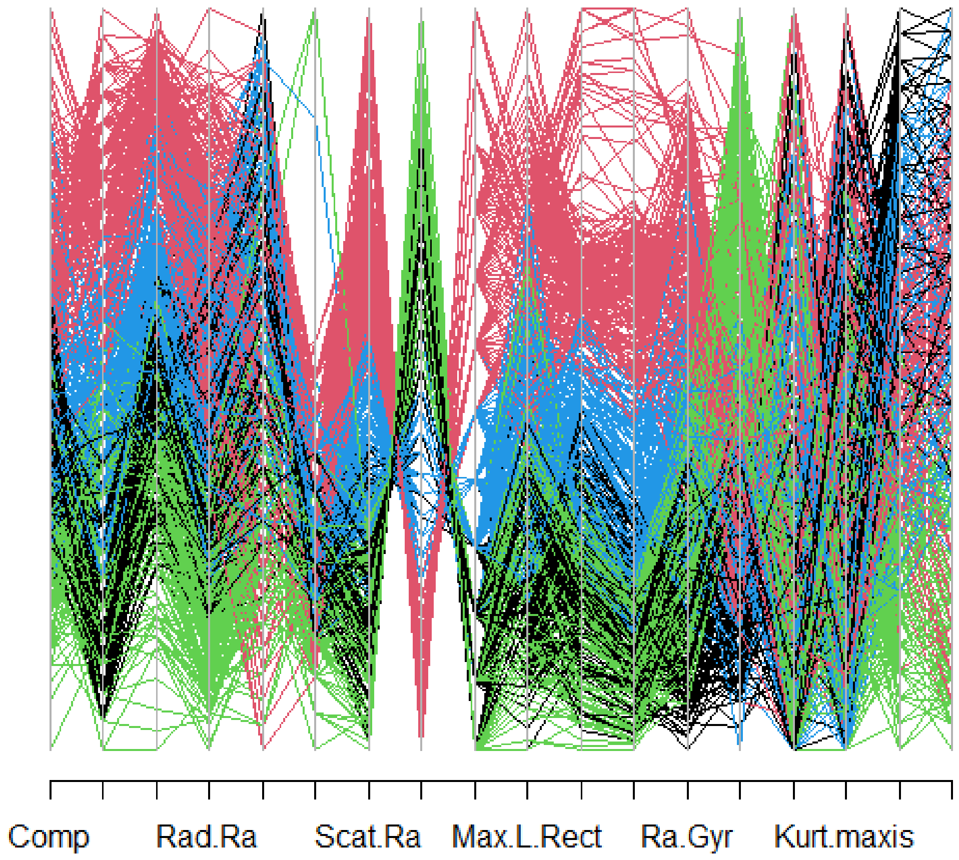
The clustering for the data above (with outliers removed AFTER scaling at 3 standard deviations) produced the following results when checking the fit summary and cluster centres (output taken directly from the console):

```
K-means clustering with 4 clusters of sizes 179, 231, 219, 195

Cluster means:
        Comp        Circ      D.Circ     Rad.Ra  Pr.Axis.Ra    Max.L.Ra     Scat.Ra      Elong Pr.Axis.Rect
1 -0.3612345 -0.97450099 -0.5744984 -0.2806334  0.04518440 -0.25322460 -0.73186584  0.6585632   -0.7399057
2  1.1861596  1.25098097  1.2693486  0.9798991  0.07942101  0.26525656  1.34644361 -1.2484047    1.3550572
3 -0.9984688 -0.60517497 -1.0018672 -1.1418268 -0.59272816 -0.37041555 -0.81995346  0.9262099   -0.7849879
4 -0.0188731  0.03322026  0.1036313  0.2220788  0.27479919 -0.03749097 -0.06936116 -0.1234805   -0.1179795
   Max.L.Rect Sc.Var.Maxis Sc.Var.maxis      Ra.Gyr  Skew.Maxis Skew.maxis  Kurt.maxis  Kurt.Maxis
1 -0.88742138  -0.69486712   -0.7011871 -1.03760341 -0.78402785 -0.2470456  0.09428142  1.03146186
2  1.18405496   1.22715146    1.3542603  1.14302502 -0.08948437  0.1416174  0.27800845 -0.06000269
3 -0.59886267  -0.83993391   -0.8156358 -0.44671257  0.85980910 -0.1221573 -0.27007180 -1.10470350
4  0.02551817  -0.03690508   -0.1197999  0.02266653 -0.35968392  0.1235265 -0.15208024  0.39280369
     Holl.Ra
1  0.8620083
2  0.1791930
3 -1.2034233
4  0.3692935

Within cluster sum of squares by cluster:
[1] 1020.914 1521.980 1221.493 1217.239
 (between_SS / total_SS =  62.0 %)
```

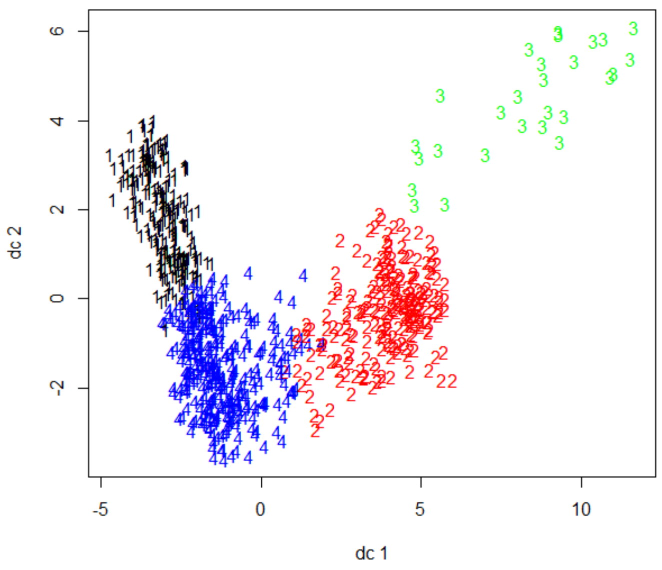
Seeing the results, I decided to visualise each clustering by plotting the clusters as well as their representation in attributes.

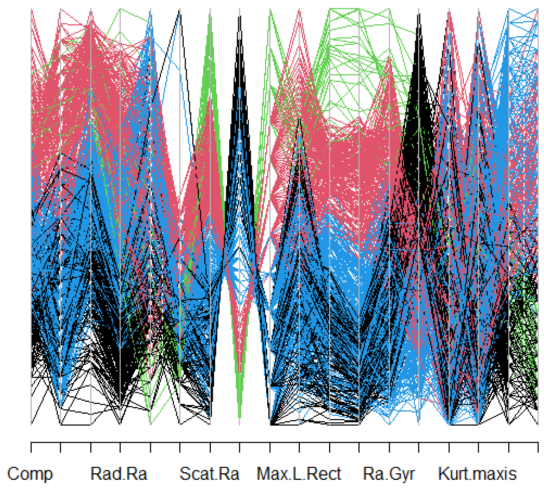
```
plotcluster(vehiclesORZ_sd3_as, fit4_ORZ_3sd_as$cluster)
parcoord(vehiclesORZ_sd3_as, fit4_ORZ_3sd_as$cluster)
```





As you can see from the visualisation alone, there seems to be quite a bit of overlap, which could potentially mean that we have a poor clustering result. I have performed clustering for a value of k = 2,3,4 for each of my 5 datasets (with differently removed outliers) as well as my control dataset with no outliers removed. Differently pre-processed data gave different solutions, as can be seen here form data similar to above, but with outlier removed BEFORE scaling instead:

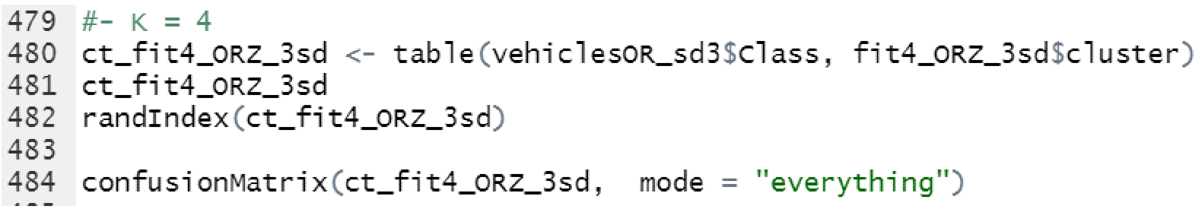




As mentioned previously, the clustering for this section was performed on each dataset, but in order to spare the length of this report, I have only included the examples above, as all datasets visualised look rather similar. Their true comparison will be done using a confusion matrix in the next section. To see the visualisation of the rest of the datasets, please refer to the R script at the end of the clustering section of this report.

EVALUATION OF PRODUCED OUTPUT AGAINST 19TH COLUMN & EVALUATION INDICES

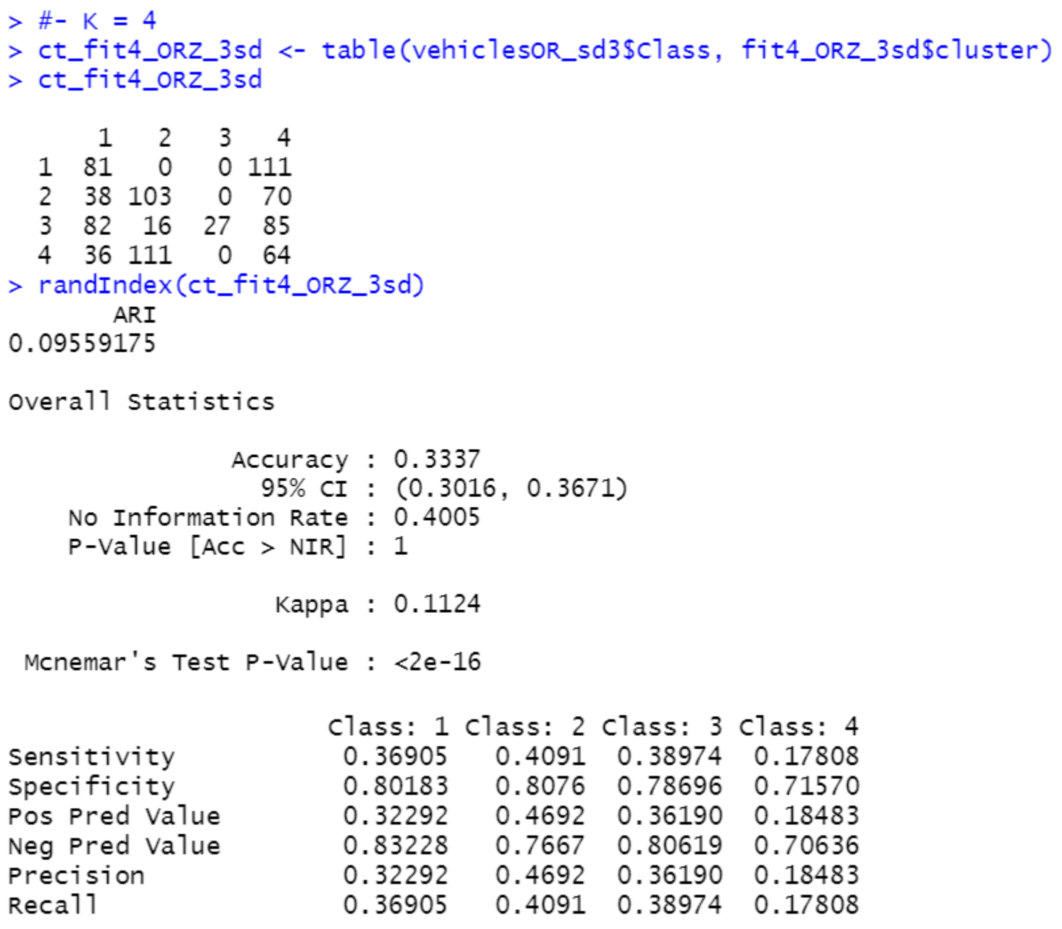
To evaluate the outputs of my clustering to the original classes of the observations denoted in the 19th column, I compare the two using a confusion matrix. I do this by creating a table from the Class column of the dataset and the clusters produced by the clustering. I do this with the following code:



```r
#- K = 4
ct_fit4_ORZ_3sd <- table(vehiclesOR_sd3$Class, fit4_ORZ_3sd$cluster)
ct_fit4_ORZ_3sd
randIndex(ct_fit4_ORZ_3sd)

confusionMatrix(ct_fit4_ORZ_3sd,  mode = "everything")
```

The above displays the confusion matrix, the adjusted random index (ARI) value as well as a summary of statistics using the confusionMatrix() method. The ARI value denotes the agreement between our class types and our clustering results. Among the summary statistics provided by the confustionMatrix method are the accuracy, precision and recall values of our clustering.

For the data above, with outliers removed before scaling at +/- 3 standard deviations, the confusion matrix, ARI, accuracy, recall and precision look as follows:

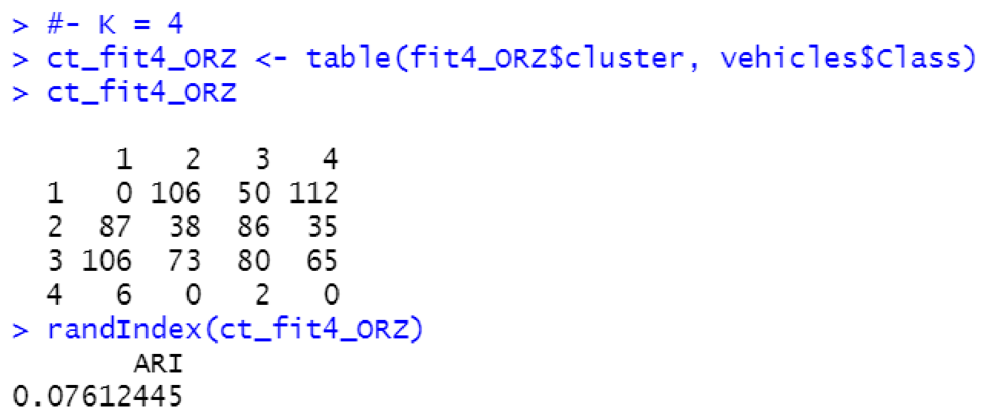


```
> #- K = 4
> ct_fit4_ORZ_3sd <- table(vehiclesOR_sd3$Class, fit4_ORZ_3sd$cluster)
> ct_fit4_ORZ_3sd

      1   2   3   4
  1  81   0   0 111
  2  38 103   0  70
  3  82  16  27  85
  4  36 111   0  64
> randIndex(ct_fit4_ORZ_3sd)
       ARI
0.09559175

Overall Statistics

               Accuracy : 0.3337
                 95% CI : (0.3016, 0.3671)
    No Information Rate : 0.4005
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1124

 Mcnemar's Test P-Value : <2e-16

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity           0.36905   0.4091  0.38974  0.17808
Specificity           0.80183   0.8076  0.78696  0.71570
Pos Pred Value        0.32292   0.4692  0.36190  0.18483
Neg Pred Value        0.83228   0.7667  0.80619  0.70636
Precision             0.32292   0.4692  0.36190  0.18483
Recall                0.36905   0.4091  0.38974  0.17808
```

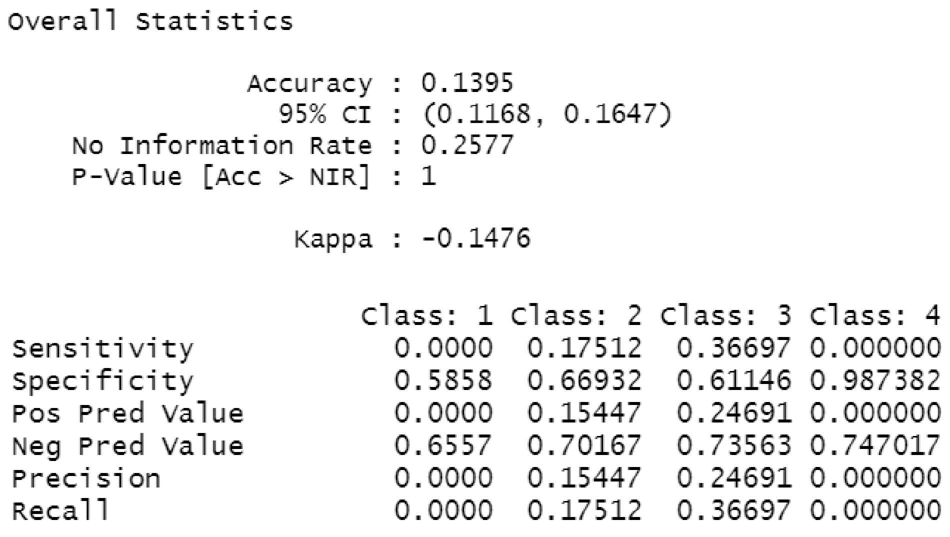
As we can see from the above confusion matrix and the clustering statistics, the clustering results are not as great as we might expect. We can see that there is quite a bit of overlap on the confusion matrix. We can also tell that our results aren't ideal from the fact that our ARI value is only at 0.095, which means that our results are only ever so slightly agreeable with the original class types of the data. Another statistic we can look to is the accuracy, which at 33% is not as convincing as it could be. The recall and precision statistics are within the 30-47% range for classes 1-3, and around 18% for class 4.

These results could mean that the pre-processing steps for this dataset were not optimal. Perhaps not all of the outliers were removed (which is likely the main culprit), or perhaps the data ought to have been scaled before the outliers were removed. It could aslo be that the data itself was not of good quality, which would mean that the clustering results are not very clear cut.

In order to compare whether the outlier removal had any effect on the data, I will compare these results with the original data that has only been scaled (with no outliers removed):



```
> #- K = 4
> ct_fit4_ORZ <- table(fit4_ORZ$cluster, vehicles$Class)
> ct_fit4_ORZ

      1   2   3   4
  1   0 106  50 112
  2  87  38  86  35
  3 106  73  80  65
  4   6   0   2   0
> randIndex(ct_fit4_ORZ)
       ARI
0.07612445
```



```
Overall Statistics

               Accuracy : 0.1395
                 95% CI : (0.1168, 0.1647)
    No Information Rate : 0.2577
    P-Value [Acc > NIR] : 1

                  Kappa : -0.1476


                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.0000  0.17512  0.36697 0.000000
Specificity            0.5858  0.66932  0.61146 0.987382
Pos Pred Value         0.0000  0.15447  0.24691 0.000000
Neg Pred Value         0.6557  0.70167  0.73563 0.747017
Precision              0.0000  0.15447  0.24691 0.000000
Recall                 0.0000  0.17512  0.36697 0.000000
```

As we can see, when clustering was performed (with k = 4) on the data without any outliers removed, the results are much worse, with the ARI value dropping to a 0.076 and the clustering accuracy dropping to about 14%, with recall and precision being 0% for two classes and hovering in the 15-35% range for classes 2/3. This shows that our outlier removal certainly had a positive effect on the data, but perhaps wasn't effective enough and that more outliers may have needed to be removed.

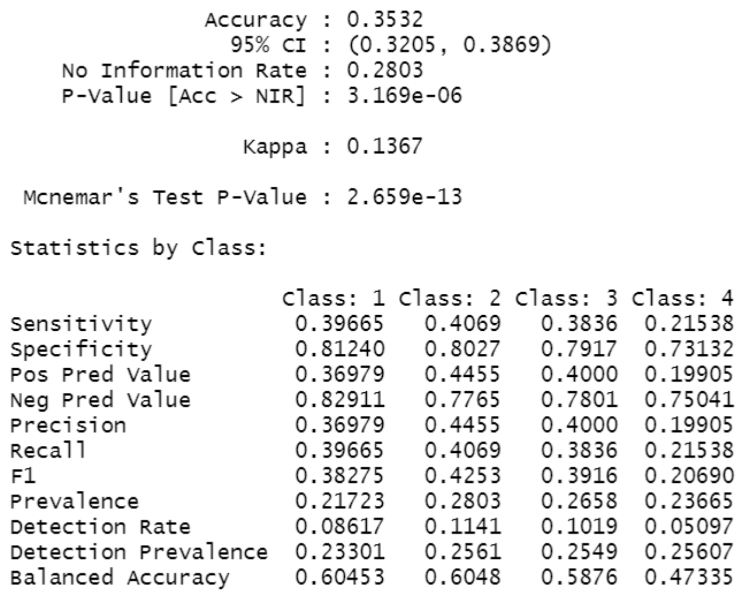
To conclude which dataset and which value of k performed the most optimally, I will display the evaluation of each clustering result in a table below:

Evaluation of Clustering Results from Differently Pre-Processed Datasets						
Statistic (No. of Clusters)	**Dataset**					
	Only Scaling	**OR AS +/- 2sd**	**OR AS +/- 3sd**	**OR BS +/- 2sd**	**OR BS +/- 3sd 1i**	**OR BS +/- 3sd 2i**
ARI (k = 2)	0.083	0.102	0.089	0.104	0.089	0.090
ARI (k = 3)	0.073	0.085	0.080	0.079	0.077	0.080
ARI (k = 4)	0.076	0.086	0.074	0.086	0.079	0.085
Accuracy (k = 4)	13.95%	21.99%	35.32%	21.69%	33.50%	24.32%
Precision (k = 4)	10.03%	21.35%	35.36%	21.07%	33.47%	24.65%
Recall (k = 4)	13.55%	21.10%	35.06%	17.25%	33.65%	25.01%

*OR - Outliers removed
*AS/BS - After/Before scaling
*sd - Standard deviation
*i – Iteration

As you can see from the table above, the best performing dataset when measured by accuracy, recall and precision was the one that had outliers removed after scaling, with the outliers removed at +/- 3 standard deviations. As for the best performing 'winner' cluster case when judged by ARI is value of k = 2, with the dataset which had outliers removed before scaling at +/- 2 standard deviations.

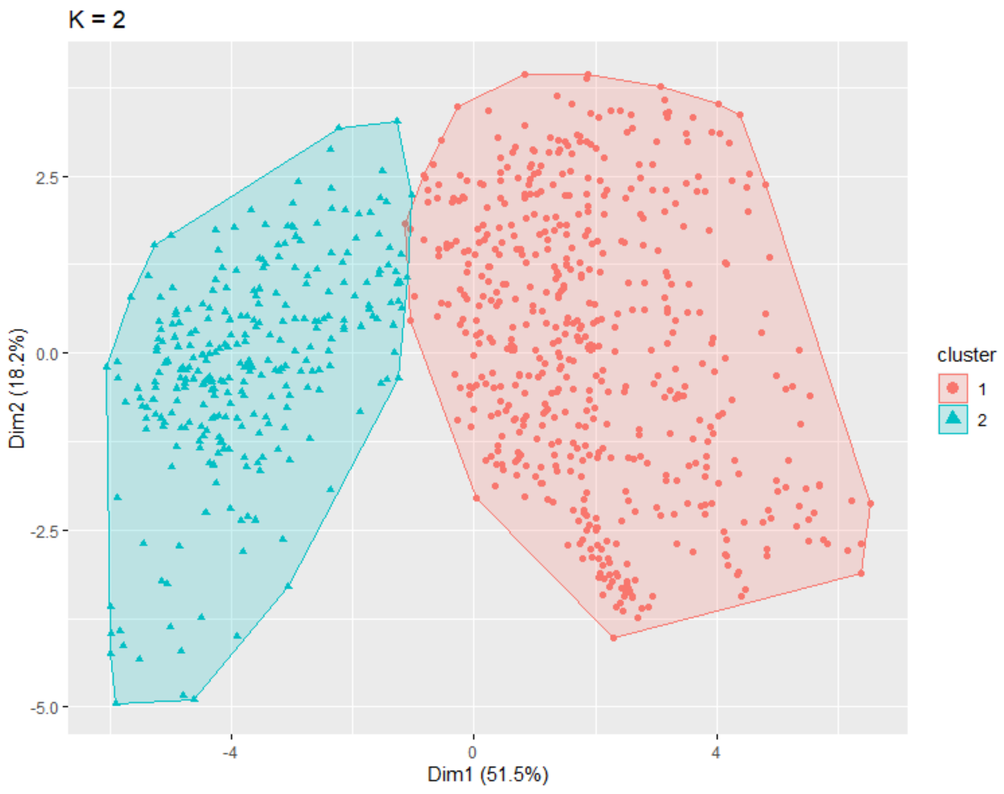
That said, looking at the detailed statistics for the dataset with the highest accuracy, we can see that the best cluster in the set when the value of k was set to 4, is cluster #2, with a balanced accuracy of 60.48%, a precision of 44.55% and recall of 40.69%.

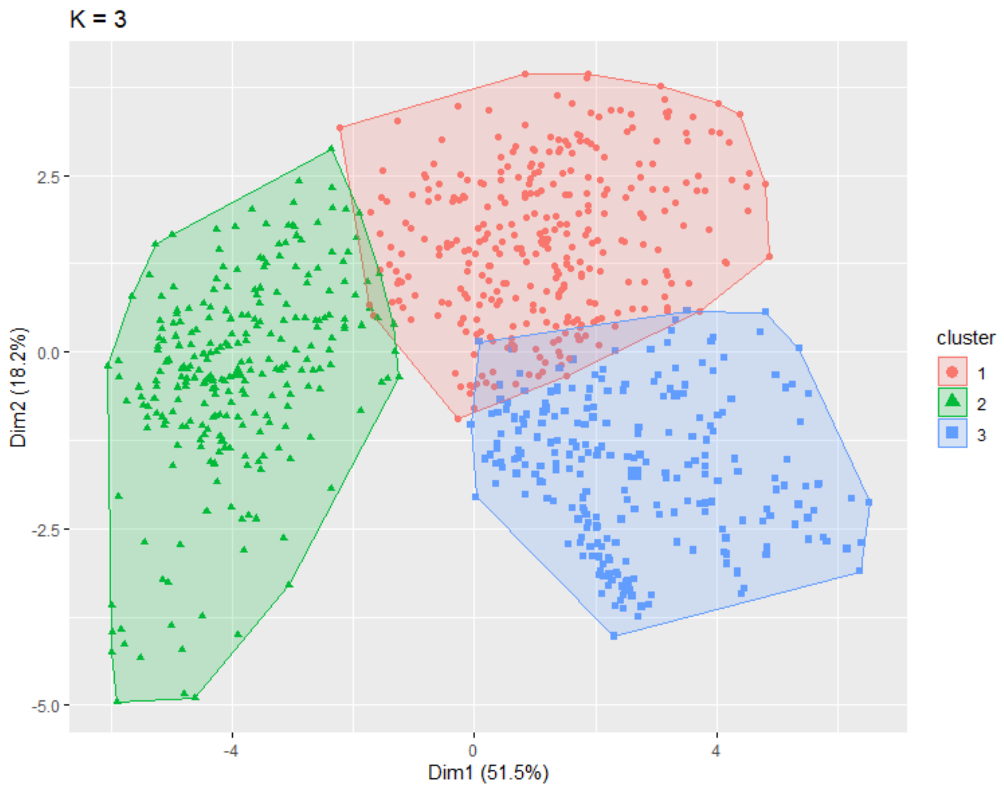


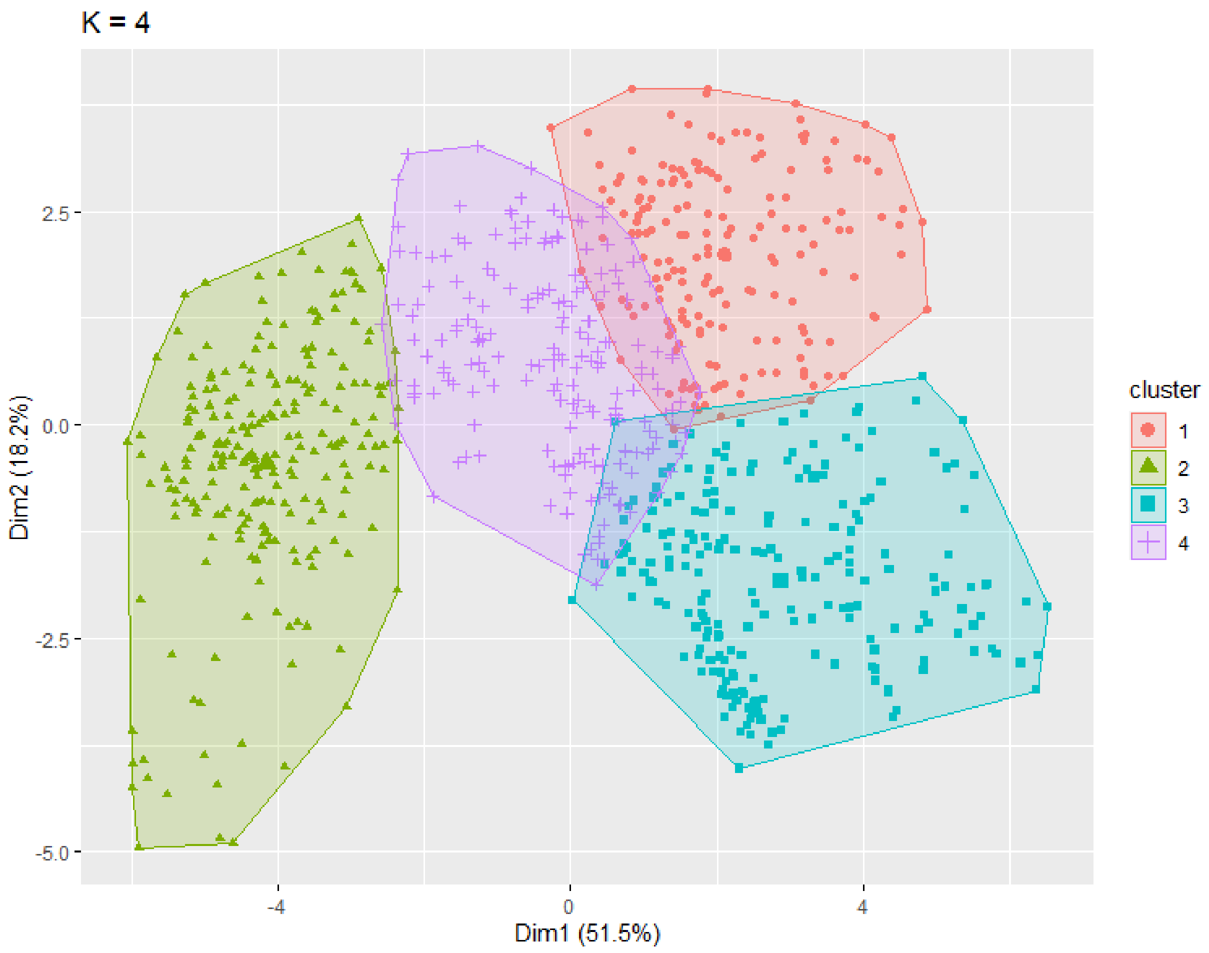
```
               Accuracy : 0.3532
                 95% CI : (0.3205, 0.3869)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : 3.169e-06

                  Kappa : 0.1367

 Mcnemar's Test P-Value : 2.659e-13

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity           0.39665   0.4069   0.3836  0.21538
Specificity           0.81240   0.8027   0.7917  0.73132
Pos Pred Value        0.36979   0.4455   0.4000  0.19905
Neg Pred Value        0.82911   0.7765   0.7801  0.75041
Precision             0.36979   0.4455   0.4000  0.19905
Recall                0.39665   0.4069   0.3836  0.21538
F1                    0.38275   0.4253   0.3916  0.20690
Prevalence            0.21723   0.2803   0.2658  0.23665
Detection Rate        0.08617   0.1141   0.1019  0.05097
Detection Prevalence  0.23301   0.2561   0.2549  0.25607
Balanced Accuracy     0.60453   0.6048   0.5876  0.47335
```

VISUALISING CLUSTER CENTRES

For the winning dataset (outliers removed after scaling at +/- 3 standard deviations) I will show below the visualised clusters along with their centres illustrated for each value of k that was used in our clustering:







The plots above were achieved with the function below from the factoextra library:

```
fviz_cluster(fit4_ORZ_3sd_as, vehiclesOR_sd3[,-1], geom = "point") + ggtitle("K = 4")
```

CLUSTERING APPENDIX (R SCRIPT)

```
library("readxl")
library("ggplot2")
library("factoextra")
library("NbClust")
library("fpc")
library("MASS")
library("flexclust")
library("caret")

#----------------------- Functions -----------------------------

#Function used for min-max normalisation
normalise <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

#This function displays the z-score distribution and displays outliers
plotZScoreOutliers <- function(df, zscore) {

  ggplot(df, aes(zscore)) +
     geom_histogram(aes(x=zscore, y=..count..), bins = 50) +
     geom_vline(xintercept = 3, color = "red") +
     geom_vline(xintercept = -3, color = "red")

}

#A method which uses scaled data to remove outliers from original, NON-
SCALED data
removeOutliers <- function(df, dfz, x) {


  for (i in 1:18) {
    df <- df[dfz[[i]] < x & dfz[[i]] > -x, ]
    dfz <- dfz[dfz[[i]] < x & dfz[[i]] > -x, ]
  }

  return(df)

}

#A method which removes outliers from already SCALED data
scaleAndRemoveOutliers <- function(dfz, x) {

  for (i in 1:18) {
    dfz <- dfz[dfz[[i]] < x & dfz[[i]] > -x, ]
  }

  return(dfz)

}

#The elbow method used to determine optimal no. of clusters
elbowMethod <- function(df) {

  wss <- 0
  for (i in 1:10){
    wss[i] <-
      sum(kmeans(df, centers=i)$withinss)
  }
  plot(1:10,
       wss,
       type="b",
       xlab="Number of Clusters",
       ylab="Within Groups Sum of Squares")

}


createBoxplot <- function(df, mtitle, ytitle, xtitle) {

  boxplot(df,
          main = mtitle,
          ylab = ytitle,
          xlab = xtitle)

}

#---------------------- Initial Data Frames ------------------------------
------
```

```
vehicles = read_excel("vehicles2.xlsx")

#Calculate Z scores for each value
vehiclesZScores <- as.data.frame(scale(vehicles[2:19]))
vehiclesZScoresWithClass <- vehiclesZScores
vehiclesZScoresWithClass$Class <- vehicles$Class

#MinMax Normalisation
vehiclesNormalised <- as.data.frame(lapply(vehicles[2:19], normalise))

#---------------------- Calling method to remove outliers ----------------
------

#--- Scaling BEFORE removing outliers:

#2 standard deviations
vehiclesORZ_sd2_as <- scaleAndRemoveOutliers(vehiclesZScores, 2)

#3 standard deviations
vehiclesORZ_sd3_as <- scaleAndRemoveOutliers(vehiclesZScores, 3)

#--- Scaling AFTER removing outliers:

#2 standard deviations
vehiclesOR_sd2 <- removeOutliers(vehicles[2:20],
vehiclesZScoresWithClass, 2)
vehiclesORZ_sd2 <- as.data.frame(scale(vehiclesOR_sd2[1:18]))

#3 standard deviations - iteration 1
vehiclesOR_sd3 <- removeOutliers(vehicles[2:20],
vehiclesZScoresWithClass, 3)
vehiclesORZ_sd3 <- as.data.frame(scale(vehiclesOR_sd3[1:18]))
vehiclesORZ_sd3_WithClass <- vehiclesORZ_sd3
vehiclesORZ_sd3_WithClass$Class <- vehiclesOR_sd3$Class

#3 standard deviations - iteration 2
vehiclesOR2_sd3 <- removeOutliers(vehiclesOR_sd3,
vehiclesORZ_sd3_WithClass, 3)
vehiclesORZ2_sd3 <- as.data.frame(scale(vehiclesOR2_sd3[1:18]))

#---------------------- Defining K (No. of Clusters) ---------------------
------

#--- No Outliers Removed:

#- Manual Methods:

#Elbow method (Written by me)
elbowMethod(vehiclesZScores)
#Eblow method alternate (Imported)
fviz_nbclust(vehiclesZScores, kmeans, method = "wss") +
  geom_vline(xintercept = 3, linetype = 2) +
  geom_vline(xintercept = 7, linetype = 2)

#Silhouette method
fviz_nbclust(vehiclesZScores, kmeans, method = "silhouette")

#- Automated Methods:

#Gap Statistic method
```

```
set.seed(100)
fviz_nbclust(vehiclesZScores, kmeans, nstart = 25,  method = "gap_stat",
nboot = 50)

#30 Indices Method
set.seed(100)
ncZ <- NbClust(data = vehiclesZScores, distance = "euclidean",
                        min.nc = 2, max.nc = 15, method = "kmeans")

#--- Outliers Removed AFTER Scaling 2sd

#- Manual Methods:

#Elbow method (Written by me)
elbowMethod(vehiclesORZ_sd2_as)
#Eblow method alternate (Imported)
fviz_nbclust(vehiclesORZ_sd2_as, kmeans, method = "wss") +
  geom_vline(xintercept = 3, linetype = 2)

#Silhouette method
fviz_nbclust(vehiclesORZ_sd2_as, kmeans, method = "silhouette")

#- Automated Methods:

#Gap Statistic method
set.seed(100)
fviz_nbclust(vehiclesORZ_sd2_as, kmeans, nstart = 25,  method =
"gap_stat", nboot = 50)

#30 Indices Method
set.seed(100)
ncORZ_sd2_as <- NbClust(data = vehiclesORZ_sd2_as, distance =
"euclidean",
                      min.nc = 2, max.nc = 15, method = "kmeans")

#--- Outliers Removed AFTER Scaling 3sd

#- Manual Methods:

#Elbow method (Written by me)
elbowMethod(vehiclesORZ_sd3_as)
#Eblow method alternate (Imported)
fviz_nbclust(vehiclesORZ_sd3_as, kmeans, method = "wss") +
  geom_vline(xintercept = 3, linetype = 2)

#Silhouette method
fviz_nbclust(vehiclesORZ_sd3_as, kmeans, method = "silhouette")

#- Automated Methods:

#Gap Statistic method
set.seed(100)
fviz_nbclust(vehiclesORZ_sd3_as, kmeans, nstart = 25,  method =
"gap_stat", nboot = 50)

#30 Indices Method
set.seed(100)
ncORZ_sd3_as <- NbClust(data = vehiclesORZ_sd3_as, distance =
"euclidean",
                      min.nc = 2, max.nc = 15, method = "kmeans")
```

```
#--- Outliers Removed BEFORE Scaling 2sd

#- Manual Methods:

#Elbow method (Written by me)
elbowMethod(vehiclesORZ_sd2)
#Eblow method alternate (Imported)
fviz_nbclust(vehiclesORZ_sd2, kmeans, method = "wss") +
  geom_vline(xintercept = 3, linetype = 2)

#Silhouette method
fviz_nbclust(vehiclesORZ_sd2, kmeans, method = "silhouette")

#- Automated Methods:

#Gap Statistic method
set.seed(100)
fviz_nbclust(vehiclesORZ_sd2, kmeans, nstart = 25,  method = "gap_stat",
nboot = 50)

#30 Indices Method
set.seed(100)
ncORZ_sd2 <- NbClust(data = vehiclesORZ_sd2, distance = "euclidean",
                     min.nc = 2, max.nc = 15, method = "kmeans")

#--- Outliers Removed BEFORE Scaling 3sd (1 Iteration)

#- Manual Methods:

#Elbow method (Written by me)
elbowMethod(vehiclesORZ_sd3)
#Eblow method alternate (Imported)
fviz_nbclust(vehiclesORZ_sd3, kmeans, method = "wss") +
  geom_vline(xintercept = 3, linetype = 2)

#Silhouette method
fviz_nbclust(vehiclesORZ_sd3, kmeans, method = "silhouette")

#- Automated Methods:

#Gap Statistic method
set.seed(100)
fviz_nbclust(vehiclesORZ_sd3, kmeans, nstart = 25,  method = "gap_stat",
nboot = 50)

#30 Indices Method
set.seed(100)
ncORZ_sd3 <- NbClust(data = vehiclesORZ_sd3, distance = "euclidean",
                     min.nc = 2, max.nc = 15, method = "kmeans")

#--- Outliers Removed BEFORE Scaling 3sd (2 Iterations)

#- Manual Methods:

#Elbow method (Written by me)
elbowMethod(vehiclesORZ2_sd3)
#Eblow method alternate (Imported)
fviz_nbclust(vehiclesORZ2_sd3, kmeans, method = "wss") +
  geom_vline(xintercept = 4, linetype = 2)
```

```
#Silhouette method
fviz_nbclust(vehiclesORZ2_sd3, kmeans, method = "silhouette")

#- Automated Methods:

#Gap Statistic method
set.seed(100)
fviz_nbclust(vehiclesORZ2_sd3, kmeans, nstart = 25,  method = "gap_stat",
nboot = 50)

#30 Indecies Method
set.seed(100)
ncORZ2_sd3 <- NbClust(data = vehiclesORZ2_sd3, distance = "euclidean",
        min.nc = 2, max.nc = 15, method = "kmeans")

#--------------------- K-Means Analysis & Visualisation -----------------
----------------------

#--- No Outliers Removed:

#- K = 2
set.seed(100)
fit2_ORZ <- kmeans(vehiclesZScores, 2)
plotcluster(vehiclesZScores, fit2_ORZ$cluster)
parcoord(vehiclesZScores, fit2_ORZ$cluster)

#- K = 3
set.seed(100)
fit3_ORZ <- kmeans(vehiclesZScores, 3)
plotcluster(vehiclesZScores, fit3_ORZ$cluster)
parcoord(vehiclesZScores, fit3_ORZ$cluster)

#- K = 4
set.seed(100)
fit4_ORZ <- kmeans(vehiclesZScores, 4)
plotcluster(vehiclesZScores, fit4_ORZ$cluster)
parcoord(vehiclesZScores, fit4_ORZ$cluster)

#--- Outliers Removed AFTER Scaling 2sd

#- K = 2
set.seed(100)
fit2_ORZ_2sd_as <- kmeans(vehiclesORZ_sd2_as, 2)
plotcluster(vehiclesORZ_sd2_as, fit2_ORZ_2sd_as$cluster)
parcoord(vehiclesORZ_sd2_as, fit2_ORZ_2sd_as$cluster)

#- K = 3
set.seed(100)
fit3_ORZ_2sd_as <- kmeans(vehiclesORZ_sd2_as, 3)
plotcluster(vehiclesORZ_sd2_as, fit3_ORZ_2sd_as$cluster)
parcoord(vehiclesORZ_sd2_as, fit3_ORZ_2sd_as$cluster)

#- K = 4
set.seed(100)
fit4_ORZ_2sd_as <- kmeans(vehiclesORZ_sd2_as, 4)
plotcluster(vehiclesORZ_sd2_as, fit4_ORZ_2sd_as$cluster)
parcoord(vehiclesORZ_sd2_as, fit4_ORZ_2sd_as$cluster)

#--- Outliers Removed AFTER Scaling 3sd
```

```
#- K = 2
set.seed(100)
fit2_ORZ_3sd_as <- kmeans(vehiclesORZ_sd3_as, 2)
plotcluster(vehiclesORZ_sd3_as, fit2_ORZ_3sd_as$cluster)
parcoord(vehiclesORZ_sd3_as, fit2_ORZ_3sd_as$cluster)

#- K = 3
set.seed(100)
fit3_ORZ_3sd_as <- kmeans(vehiclesORZ_sd3_as, 3)
plotcluster(vehiclesORZ_sd3_as, fit3_ORZ_3sd_as$cluster)
parcoord(vehiclesORZ_sd3_as, fit3_ORZ_3sd_as$cluster)

#- K = 4
set.seed(100)
fit4_ORZ_3sd_as <- kmeans(vehiclesORZ_sd3_as, 4)
fit4_ORZ_3sd_as
fit4_ORZ_3sd_as$centers
fit4_ORZ_3sd_as$size

plotcluster(vehiclesORZ_sd3_as, fit4_ORZ_3sd_as$cluster)
parcoord(vehiclesORZ_sd3_as, fit4_ORZ_3sd_as$cluster)

#--- Outliers Removed BEFORE Scaling 2sd

#- K = 2
set.seed(100)
fit2_ORZ_2sd <- kmeans(vehiclesORZ_sd2, 2)
plotcluster(vehiclesORZ_sd2, fit2_ORZ_2sd$cluster)
parcoord(vehiclesORZ_sd2, fit2_ORZ_2sd$cluster)

#- K = 3
set.seed(100)
fit3_ORZ_2sd <- kmeans(vehiclesORZ_sd2, 3)
plotcluster(vehiclesORZ_sd2, fit3_ORZ_2sd$cluster)
parcoord(vehiclesORZ_sd2, fit3_ORZ_2sd$cluster)

#- K = 4
set.seed(100)
fit4_ORZ_2sd <- kmeans(vehiclesORZ_sd2, 4)
plotcluster(vehiclesORZ_sd2, fit4_ORZ_2sd$cluster)
parcoord(vehiclesORZ_sd2, fit4_ORZ_2sd$cluster)

#--- Outliers Removed BEFORE Scaling 3sd (1 Iteration)

#- K = 2
set.seed(100)
fit2_ORZ_3sd <- kmeans(vehiclesORZ_sd3, 2)
plotcluster(vehiclesORZ_sd3, fit2_ORZ_3sd$cluster)
parcoord(vehiclesORZ_sd3, fit2_ORZ_3sd$cluster)

#- K = 3
set.seed(100)
fit3_ORZ_3sd <- kmeans(vehiclesORZ_sd3, 3)
plotcluster(vehiclesORZ_sd3, fit3_ORZ_3sd$cluster)
parcoord(vehiclesORZ_sd3, fit3_ORZ_3sd$cluster)

#- K = 4
set.seed(100)
fit4_ORZ_3sd <- kmeans(vehiclesORZ_sd3, 4)
```

```
plotcluster(vehiclesORZ_sd3, fit4_ORZ_3sd$cluster)
parcoord(vehiclesORZ_sd3, fit4_ORZ_3sd$cluster)

#--- Outliers Removed BEFORE Scaling 3sd (2 Iterations)

#- K = 2
set.seed(100)
fit2_ORZ2_3sd <- kmeans(vehiclesORZ2_sd3, 2)
plotcluster(vehiclesORZ2_sd3, fit2_ORZ2_3sd$cluster)
parcoord(vehiclesORZ2_sd3, fit2_ORZ2_3sd$cluster)

#- K = 3
set.seed(100)
fit3_ORZ2_3sd <- kmeans(vehiclesORZ2_sd3, 3)
plotcluster(vehiclesORZ2_sd3, fit3_ORZ2_3sd$cluster)
parcoord(vehiclesORZ2_sd3, fit3_ORZ2_3sd$cluster)

#- K = 4
set.seed(100)
fit4_ORZ2_3sd <- kmeans(vehiclesORZ2_sd3, 4)
plotcluster(vehiclesORZ2_sd3, fit4_ORZ2_3sd$cluster)
parcoord(vehiclesORZ2_sd3, fit4_ORZ2_3sd$cluster)

#--------------------- Cluster Evaluation -------------------------------
------

#--- No Outliers Removed:

#- K = 2
ct_fit2_ORZ <- table(vehicles$Class, fit2_ORZ$cluster)
ct_fit2_ORZ
randIndex(ct_fit2_ORZ)

#- K = 3
ct_fit3_ORZ <- table(vehicles$Class, fit3_ORZ$cluster)
ct_fit3_ORZ
randIndex(ct_fit3_ORZ)

#- K = 4
ct_fit4_ORZ <- table(fit4_ORZ$cluster, vehicles$Class)
ct_fit4_ORZ
randIndex(ct_fit4_ORZ)

confusionMatrix(ct_fit4_ORZ, mode = "everything")


#--- Outliers Removed AFTER Scaling 2sd

#- K = 2
ct_fit2_ORZ_2sd_as <- table(vehiclesOR_sd2$Class,
fit2_ORZ_2sd_as$cluster)
ct_fit2_ORZ_2sd_as
randIndex(ct_fit2_ORZ_2sd_as)

#- K = 3
ct_fit3_ORZ_2sd_as <- table(vehiclesOR_sd2$Class,
fit3_ORZ_2sd_as$cluster)
ct_fit3_ORZ_2sd_as
randIndex(ct_fit3_ORZ_2sd_as)
```

```
#- K = 4
ct_fit4_ORZ_2sd_as <- table(vehiclesOR_sd2$Class,
fit4_ORZ_2sd_as$cluster)
ct_fit4_ORZ_2sd_as
randIndex(ct_fit4_ORZ_2sd_as)

confusionMatrix(ct_fit4_ORZ_2sd_as,  mode = "everything")

#--- Outliers Removed AFTER Scaling 3sd

#- K = 2
ct_fit2_ORZ_3sd_as <- table(vehiclesOR_sd3$Class,
fit2_ORZ_3sd_as$cluster)
ct_fit2_ORZ_3sd_as
randIndex(ct_fit2_ORZ_3sd_as)

fviz_cluster(fit2_ORZ_3sd_as, vehiclesOR_sd3[,-1], geom = "point") +
ggtitle("K = 2")

#- K = 3
ct_fit3_ORZ_3sd_as <- table(vehiclesOR_sd3$Class,
fit3_ORZ_3sd_as$cluster)
ct_fit3_ORZ_3sd_as
randIndex(ct_fit3_ORZ_3sd_as)

fviz_cluster(fit3_ORZ_3sd_as, vehiclesOR_sd3[,-1], geom = "point") +
ggtitle("K = 3")

#- K = 4
ct_fit4_ORZ_3sd_as <- table(vehiclesOR_sd3$Class,
fit4_ORZ_3sd_as$cluster)
ct_fit4_ORZ_3sd_as
randIndex(ct_fit4_ORZ_3sd_as)

confusionMatrix(ct_fit4_ORZ_3sd_as,  mode = "everything")

fviz_cluster(fit4_ORZ_3sd_as, vehiclesOR_sd3[,-1], geom = "point") +
ggtitle("K = 4")

#--- Outliers Removed BEFORE Scaling 2sd

#- K = 2
ct_fit2_ORZ_2sd <- table(vehiclesOR_sd2$Class, fit2_ORZ_2sd$cluster)
ct_fit2_ORZ_2sd
randIndex(ct_fit2_ORZ_2sd)

#- K = 3
ct_fit3_ORZ_2sd <- table(vehiclesOR_sd2$Class, fit3_ORZ_2sd$cluster)
ct_fit3_ORZ_2sd
randIndex(ct_fit3_ORZ_2sd)

#- K = 4
ct_fit4_ORZ_2sd <- table(vehiclesOR_sd2$Class, fit4_ORZ_2sd$cluster)
ct_fit4_ORZ_2sd
randIndex(ct_fit4_ORZ_2sd)

confusionMatrix(ct_fit4_ORZ_2sd,  mode = "everything")

#--- Outliers Removed BEFORE Scaling 3sd (1 Iteration)
```

```
#- K = 2
ct_fit2_ORZ_3sd <- table(vehiclesOR_sd3$Class, fit2_ORZ_3sd$cluster)
ct_fit2_ORZ_3sd
randIndex(ct_fit2_ORZ_3sd)

#- K = 3
ct_fit3_ORZ_3sd <- table(vehiclesOR_sd3$Class, fit3_ORZ_3sd$cluster)
ct_fit3_ORZ_3sd
randIndex(ct_fit3_ORZ_3sd)

#- K = 4
ct_fit4_ORZ_3sd <- table(vehiclesOR_sd3$Class, fit4_ORZ_3sd$cluster)
ct_fit4_ORZ_3sd
randIndex(ct_fit4_ORZ_3sd)

confusionMatrix(ct_fit4_ORZ_3sd,  mode = "everything")

#--- Outliers Removed BEFORE Scaling 3sd (2 Iterations)


#- K = 2
ct_fit2_ORZ2_3sd <- table(vehiclesOR2_sd3$Class, fit2_ORZ2_3sd$cluster)
ct_fit2_ORZ2_3sd
randIndex(ct_fit2_ORZ2_3sd)

#- K = 3
ct_fit3_ORZ2_3sd <- table(vehiclesOR2_sd3$Class, fit3_ORZ2_3sd$cluster)
ct_fit3_ORZ2_3sd
randIndex(ct_fit3_ORZ2_3sd)

#- K = 4
ct_fit4_ORZ2_3sd <- table(vehiclesOR2_sd3$Class, fit4_ORZ2_3sd$cluster)
ct_fit4_ORZ2_3sd
randIndex(ct_fit4_ORZ2_3sd)

confusionMatrix(ct_fit4_ORZ2_3sd, mode = "everything")

#---------------------- Plots --------------------------------------------
------

#--- Outlier Removal:

#Standard & Scaled data
createBoxplot(vehicles[2:19], "Original Vehicle Data", "Observation
Count", "Variables")
createBoxplot(vehiclesZScores, "Scaled Vehicle Data (Z-Score)", "Z-
Score", "Variables")

#Outliers removed after scaling
createBoxplot(vehiclesORZ_sd2_as, "Outliers Removed AFTER Scaling (+/-
2sd)", "Z-Score", "Variables")
createBoxplot(vehiclesORZ_sd3_as, "Outliers Removed AFTER Scaling (+/-
3sd)", "Z-Score", "Variables")

#Outliers removed before scaling
createBoxplot(vehiclesORZ_sd2, "Outliers Removed BEFORE Scaling (+/-
2sd)", "Z-Score", "Variables")
createBoxplot(vehiclesORZ_sd3, "Outliers Removed BEFORE Scaling (+/-
3sd)", "Z-Score", "Variables")
```

```
createBoxplot(vehiclesORZ2_sd3, "Outliers Removed BEFORE Scaling (+/-
3sd, 2i)", "Z-Score", "Variables")


#--- Model Fitting (Determine no. of Clusters):

#30 Indices for scaled with no outliers removed.
barplot(table(ncZ$Best.n[1,]),
        xlab = "No. of Clusters",
        ylab = "No. of Criteria",
        main = "No. of Clusters Chosen by 30 Criteria - No Outlier
Removal")

#30 Indices for 2sd (Outliers removed AFTER scaling)
barplot(table(ncORZ_sd2_as$Best.n[1,]),
        xlab = "No. of Clusters",
        ylab = "No. of Criteria",
        main = "No. of Clusters Chosen by 30 Criteria - Outliers Removed
AFTER scaling at +/- 2sd")

#30 Indices for 3sd (Outliers removed AFTER scaling)
barplot(table(ncORZ_sd3_as$Best.n[1,]),
        xlab = "No. of Clusters",
        ylab = "No. of Criteria",
        main = "No. of Clusters Chosen by 30 Criteria - Outliers Removed
AFTER scaling at +/- 3sd")

#30 Indices for 2sd (Outliers removed BEFORE scaling)
barplot(table(ncORZ_sd2$Best.n[1,]),
        xlab = "No. of Clusters",
        ylab = "No. of Criteria",
        main = "No. of Clusters Chosen by 30 Criteria - Outliers Removed
BEFORE scaling at +/- 2sd")

#30 Indices for 3sd (i1) (Outliers removed BEFORE scaling)
barplot(table(ncORZ_sd3$Best.n[1,]),
        xlab = "No. of Clusters",
        ylab = "No. of Criteria",
        main = "K Chosen by 30 Indices - Outliers Removed BEFORE scaling,
+/- 3sd")

#30 Indices for 3sd (i2) (Outliers removed BEFORE scaling)
barplot(table(ncORZ2_sd3$Best.n[1,]),
        xlab = "No. of Clusters",
        ylab = "No. of Criteria",
        main = "No. of Clusters Chosen by 30 Criteria - Outliers Removed
BEFORE scaling at +/- 3sd, i2")


#Misc:

plotZScoreOutliers(vehiclesORZ2, vehiclesORZ2$Sc.Var.Maxis)
plotZScoreOutliers(vehiclesORZ, vehiclesORZ$Max.L.Ra)

plot(vehicles$Max.L.Ra, type = "p")
plot(vehiclesORZ$Max.L.Ra, type = "p")

plotZScoreOutliers(vehiclesZScores, vehiclesZScores$Comp)
plotZScoreOutliers(vehiclesZScores, vehiclesZScores$Circ)
plotZScoreOutliers(vehiclesZScores, vehiclesZScores$Max.L.Ra)
plotZScoreOutliers(vehiclesZScores, vehiclesZScores$Sc.Var.Maxis)
plotZScoreOutliers(vehiclesZScores, vehiclesZScores$Max.L.Rect)

plot(vehiclesZScores$Max.L.Rect, type="o", col="blue")

plot(vehicles$Max.L.Ra, type = "p")
plot(vehiclesZScores$Max.L.Ra, type = "p")

plot(vehiclesNormalised$Comp, type = "p")
boxplot(vehicles$Comp, ylab = "Max.L.Ra")
```

MLP

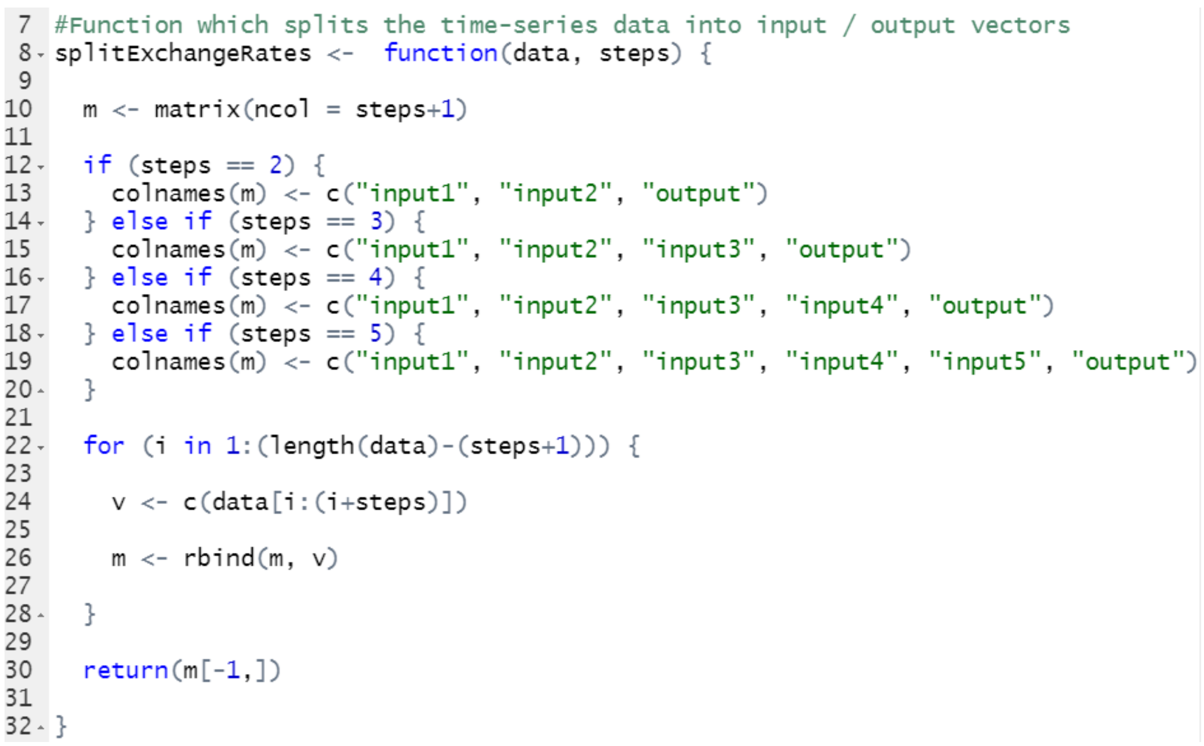
METHODS FOR DEFINING INPUT VECTOR IN TIME-SERIES PROBLEMS

Determining the input vector for a time-series problem is highly dependent on certain factors of the time-series data, such as trend and seasonality (Crone and Kourentzes, 2007). There are multiple different methods that are used for the selection of input variables; some of these methods include:

- Auto-Regression Method – a regression model, for example linear regression, which models and output value based on a linear function of input variables (Brownlee, 2017).
- Moving Average Method – a method which takes the average of a value over a period of time, which is automatically adjusted for new values over time. For example, if the period of time is 3 months, the first MA would be the average of month 1, 2 and 3. When the value for month 4 is available, the MA becomes the average of months 2, 3, 4, and so on.
- Box-Jenkins Method – a method that is a mixture of Auto Regressive and Moving Average methods, often used to identify seasonality in the data.

VARIOUS ADOPTED INPUT VECTORS AND INPUT / OUTPUT MATRICES

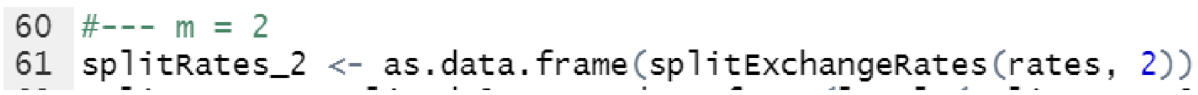
I've created my input and output matrices following the brief, from the USD/EUR column by writing a function which will create a matrix with an x number of inputs and 1 output. The output value is created from the previous x number of values, which become inputs. You can see the function that I created for this purpose below:



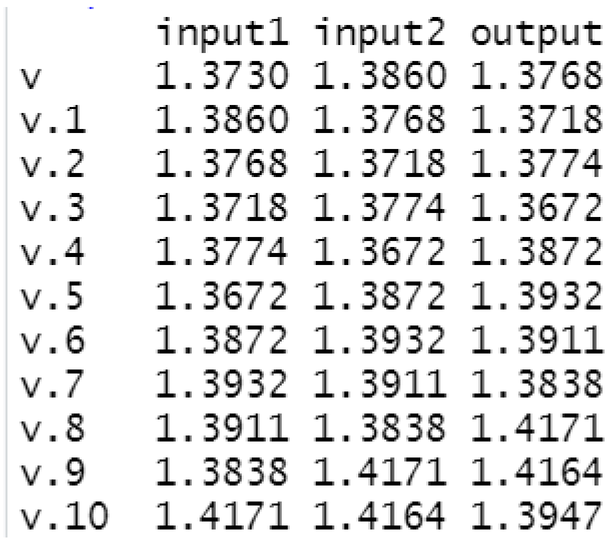
```r
#Function which splits the time-series data into input / output vectors
splitExchangeRates <-  function(data, steps) {

  m <- matrix(ncol = steps+1)

  if (steps == 2) {
    colnames(m) <- c("input1", "input2", "output")
  } else if (steps == 3) {
    colnames(m) <- c("input1", "input2", "input3", "output")
  } else if (steps == 4) {
    colnames(m) <- c("input1", "input2", "input3", "input4", "output")
  } else if (steps == 5) {
    colnames(m) <- c("input1", "input2", "input3", "input4", "input5", "output")
  }

  for (i in 1:(length(data)-(steps+1))) {

    v <- c(data[i:(i+steps)])

    m <- rbind(m, v)

  }

  return(m[-1,])

}
```

I decided to create a total of 4 different input / output matrices, which essentially stands for creating the forecast from the previous 2, 3, 4 or 5 values. I decided to stop at 5, as in our data, 5 represents a whole week. As such, I would also expect that the matrix with 5 inputs would perform the best.

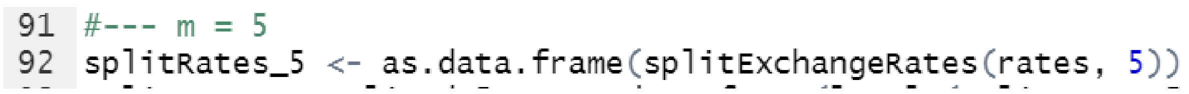
Below, you can see the matrices that are returned when the functions are called with the following arguments:



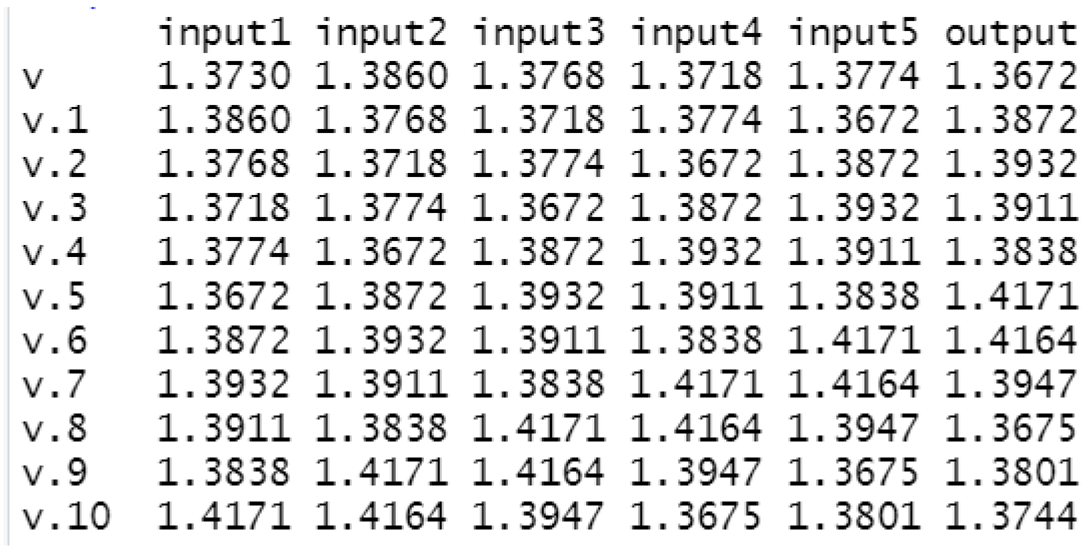
```
#--- m = 2
splitRates_2 <- as.data.frame(splitExchangeRates(rates, 2))
```



```
       input1 input2 output
v      1.3730 1.3860 1.3768
v.1    1.3860 1.3768 1.3718
v.2    1.3768 1.3718 1.3774
v.3    1.3718 1.3774 1.3672
v.4    1.3774 1.3672 1.3872
v.5    1.3672 1.3872 1.3932
v.6    1.3872 1.3932 1.3911
v.7    1.3932 1.3911 1.3838
v.8    1.3911 1.3838 1.4171
v.9    1.3838 1.4171 1.4164
v.10   1.4171 1.4164 1.3947
```



```
#--- m = 5
splitRates_5 <- as.data.frame(splitExchangeRates(rates, 5))
```

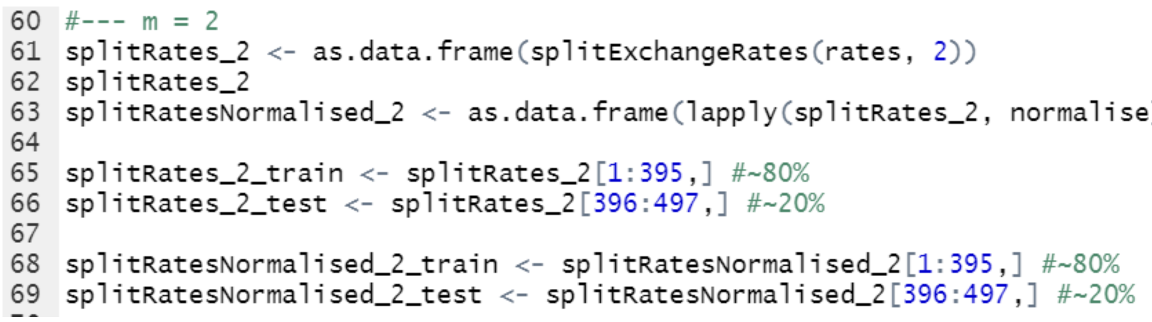


```
       input1 input2 input3 input4 input5 output
v      1.3730 1.3860 1.3768 1.3718 1.3774 1.3672
v.1    1.3860 1.3768 1.3718 1.3774 1.3672 1.3872
v.2    1.3768 1.3718 1.3774 1.3672 1.3872 1.3932
v.3    1.3718 1.3774 1.3672 1.3872 1.3932 1.3911
v.4    1.3774 1.3672 1.3872 1.3932 1.3911 1.3838
v.5    1.3672 1.3872 1.3932 1.3911 1.3838 1.4171
v.6    1.3872 1.3932 1.3911 1.3838 1.4171 1.4164
v.7    1.3932 1.3911 1.3838 1.4171 1.4164 1.3947
v.8    1.3911 1.3838 1.4171 1.4164 1.3947 1.3675
v.9    1.3838 1.4171 1.4164 1.3947 1.3675 1.3801
v.10   1.4171 1.4164 1.3947 1.3675 1.3801 1.3744
```

I have created 4 of these matrices in total, ranging from 2 inputs to 5 inputs.

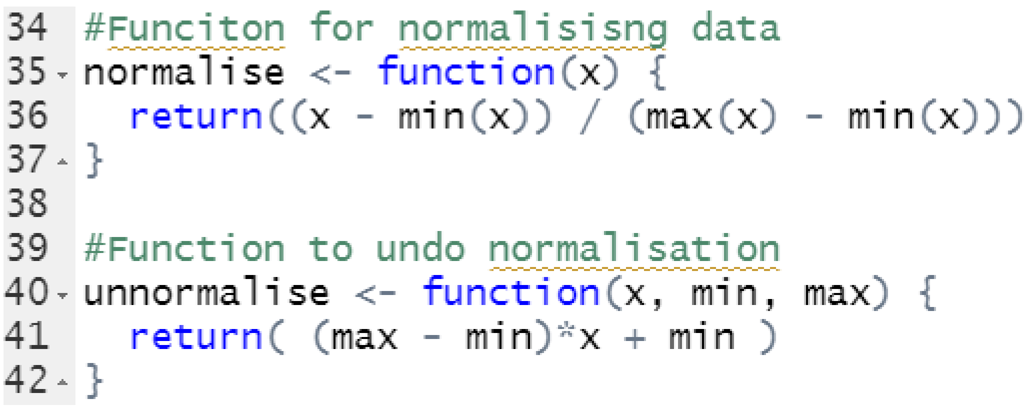
NORMALISATION AND DICSUSSION OF IMPORTANCE

As I split the data into an input/output matrix, I then could normalise this matrix and also split it into two smaller matrices, one for training and one for testing. In order to aid evaluation, I chose to do this for both the un-normalised matrix, and the normalised matrix. I have normalised the data with the following code:



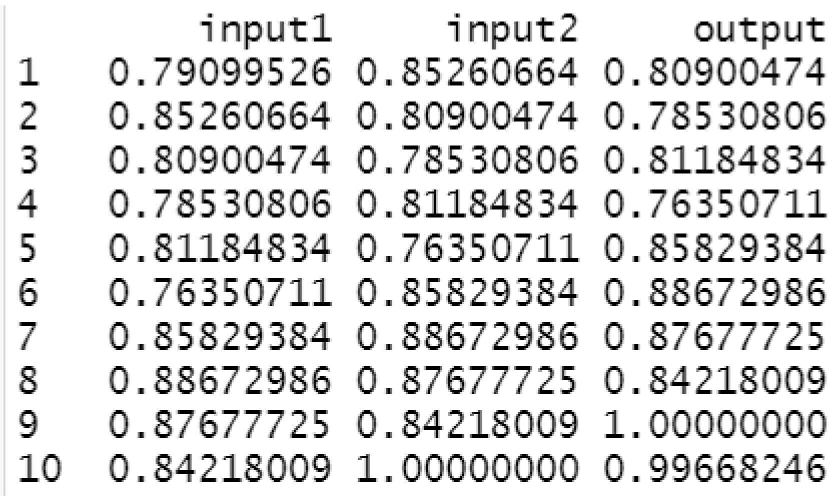
```r
#--- m = 2
splitRates_2 <- as.data.frame(splitExchangeRates(rates, 2))
splitRates_2
splitRatesNormalised_2 <- as.data.frame(lapply(splitRates_2, normalise))

splitRates_2_train <- splitRates_2[1:395,] #~80%
splitRates_2_test <- splitRates_2[396:497,] #~20%

splitRatesNormalised_2_train <- splitRatesNormalised_2[1:395,] #~80%
splitRatesNormalised_2_test <- splitRatesNormalised_2[396:497,] #~20%
```

Using the normalise function shown below:



```r
#Funciton for normalisisng data
normalise <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

#Function to undo normalisation
unnormalise <- function(x, min, max) {
  return( (max - min)*x + min )
}
```

Which has produced the following normalised input/output matrix:



```
       input1     input2     output
1  0.79099526 0.85260664 0.80900474
2  0.85260664 0.80900474 0.78530806
3  0.80900474 0.78530806 0.81184834
4  0.78530806 0.81184834 0.76350711
5  0.81184834 0.76350711 0.85829384
6  0.76350711 0.85829384 0.88672986
7  0.85829384 0.88672986 0.87677725
8  0.88672986 0.87677725 0.84218009
9  0.87677725 0.84218009 1.00000000
10 0.84218009 1.00000000 0.99668246
```

Please take not that these matrices all have around 500 observations. I am choosing to only show the first 10 for the sake of keeping this report as brief as possible. If needed, please refer to the code in the appendix to see the full matrix.

After doing a little bit of research, I found out that scaling or normalising data before training a NN with it is extremely important for the sake of efficiency. Normalising all inputs allows our network to learn the optimal parameters for each input node more quickly. Additionally, from the same source, normalisation is useful to ensure that the inputs are withing a range of +1 to -1, to avoid mathematical artifacts associated with floating point number precision. (Jordan, 2018). This is because computers lose accuracy when working with very large, or very small numbers.

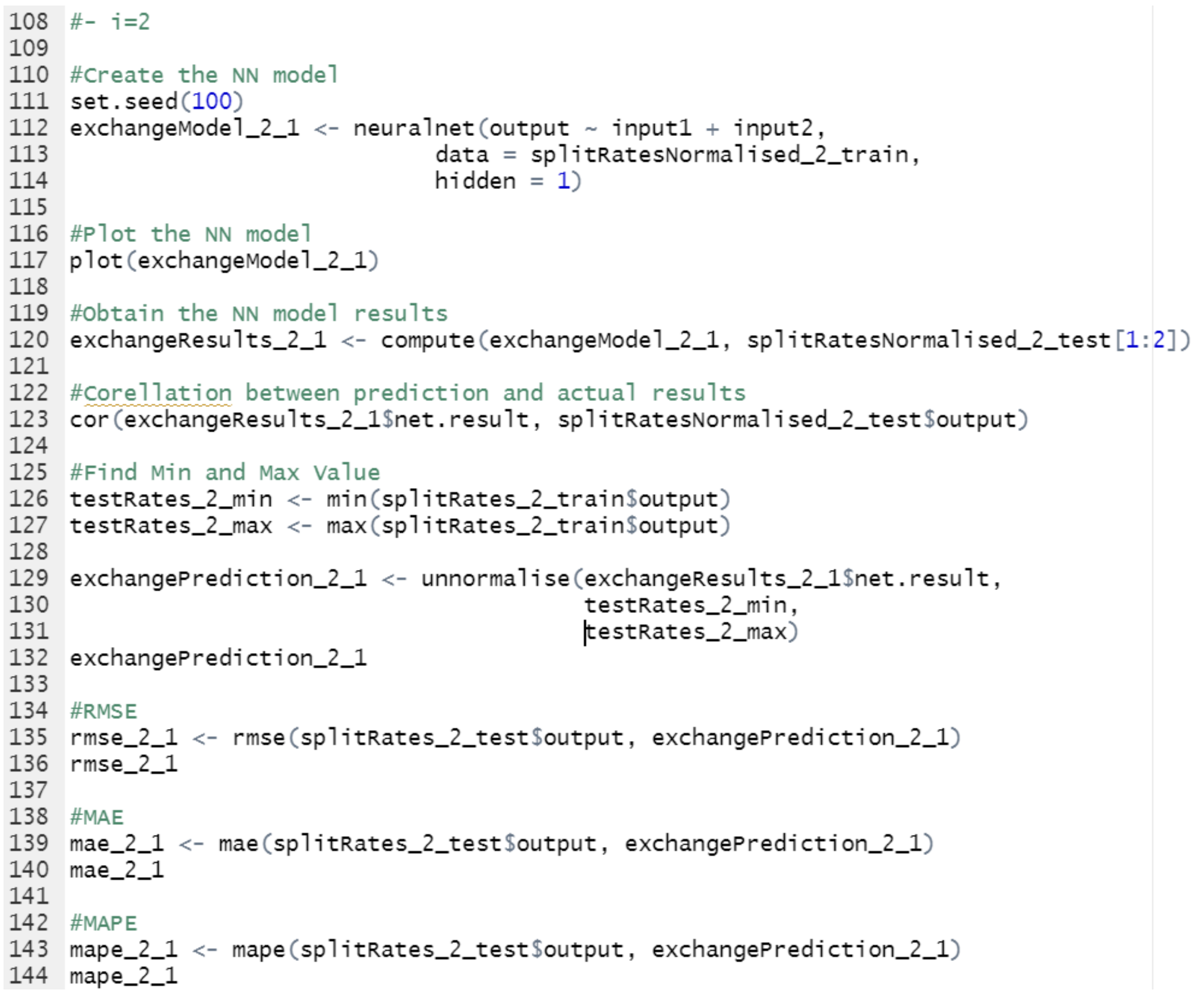
IMPLEMENT MLP USING VARIOUS STRUCTURES AND COMPARE PERFORMANCES

For this task, I have implemented a total of 38 MLPs, using a variety of structures. I have created MLPs with the following hidden layers/nodes:

- 1 hidden layer with 1 node
- 1 hidden layer with 3 nodes
- 1 hidden layer with 5 nodes
- 2 hidden layers; 1 node in each
- 2 hidden layers; 3 nodes in the first and 1 node in the second
- 2 hidden layers; 5 nodes in the first and 1 node in the second
- 2 hidden layers; 3 nodes in the first and 3 nodes in the second
- 2 hidden layers; 5 nodes in the first and 3 nodes in the second
- 2 hidden layers; 5 nodes in the first and 5 nodes in the second

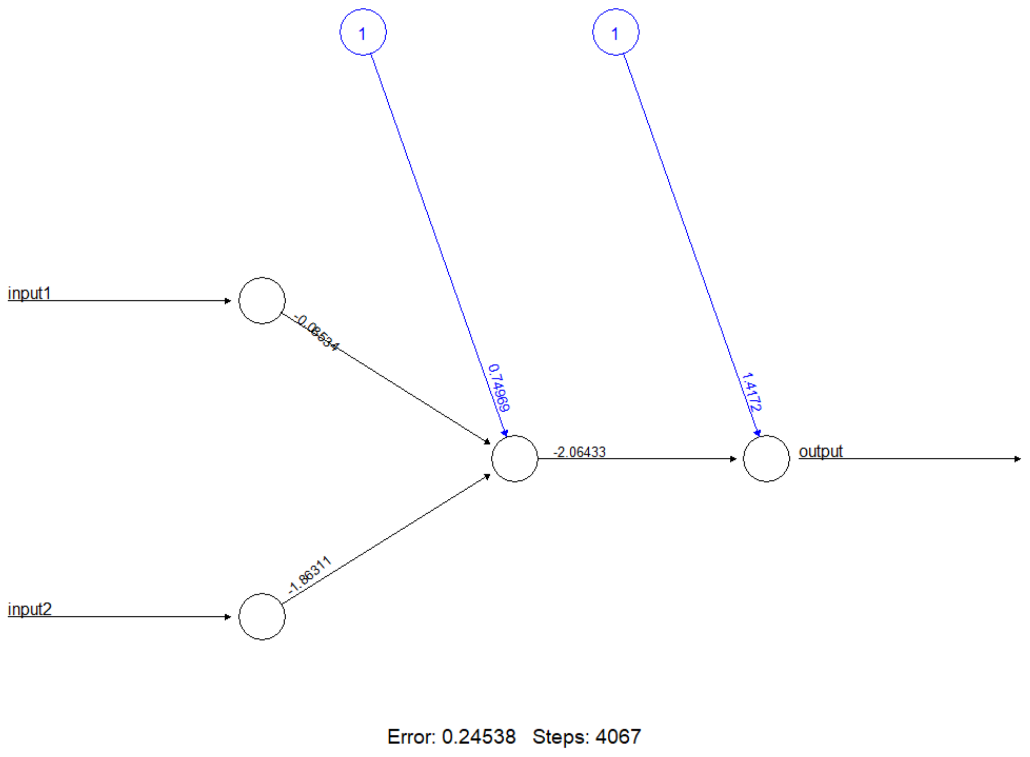
Across these 9 structures, I have tested every single input / output matrix, providing 2, 3, 4 and 5 input nodes for each of the above. This in total creates (9 * 4) 38 MLPs.

I will post below the methodology that I have used to implement a single neural network, but the same was applied to every single case listed above. For this example, I will show the MLP implemented with 2 input nodes, 1 hidden layer with 1 hidden node:



```r
#- i=2

#Create the NN model
set.seed(100)
exchangeModel_2_1 <- neuralnet(output ~ input1 + input2,
                               data = splitRatesNormalised_2_train,
                               hidden = 1)

#Plot the NN model
plot(exchangeModel_2_1)

#Obtain the NN model results
exchangeResults_2_1 <- compute(exchangeModel_2_1, splitRatesNormalised_2_test[1:2])

#Corellation between prediction and actual results
cor(exchangeResults_2_1$net.result, splitRatesNormalised_2_test$output)

#Find Min and Max Value
testRates_2_min <- min(splitRates_2_train$output)
testRates_2_max <- max(splitRates_2_train$output)

exchangePrediction_2_1 <- unnormalise(exchangeResults_2_1$net.result,
                                      testRates_2_min,
                                      testRates_2_max)
exchangePrediction_2_1

#RMSE
rmse_2_1 <- rmse(splitRates_2_test$output, exchangePrediction_2_1)
rmse_2_1

#MAE
mae_2_1 <- mae(splitRates_2_test$output, exchangePrediction_2_1)
mae_2_1

#MAPE
mape_2_1 <- mape(splitRates_2_test$output, exchangePrediction_2_1)
mape_2_1
```

As you can see from the above, I set a seed in order to keep the result reproducible. I then created the model with the neuralnet function by passing in the formula (output ~ number of inputs), the data and the number of hidden nodes as arguments. I then plotted the model to ensure that I have passed the arguments correctly. The plot for the above looks like this:



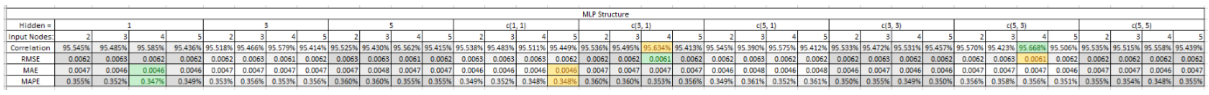
I have then used the test data on this model to calculate our prediction results. I only took the 2 input values from the test data set, so that the output value does not interfere with the results. Once I've found these results, I have calculated the correlation between the predictions and the actual, expected test values. The correlation for this examples was:

```
> cor(exchangeResults_2_1$net.result, splitRatesNormalised_2_test$output)
          [,1]
[1,] 0.9554477
```

As you can see, 96% suggests quite high closeness between the normalised predictions and the normalised test results. However, this is not the exact method to find the models accuracy. For this I needed to unnormalize the data in order to revert it to actual exchange values again. I have done this with the unnormalize function shown before, buy passing in the max and min values calculated form the unnormalized training set.

I could then revert the normalisation on the predicted data and calculate the statistical indices, including RMSE, MAE, and MAPE. I have calculated these statistics using the Metrics library, although I also have an example of one of these functions written by hand. I just found that using the library proved to be a bit faster. These statistical indices for this example are: RMSE = 0.006198, MAE = 0.004674, MAPE = 0.003545. As you can see, the values of all of these are below 1 percent, suggesting high accuracy, despite the simplicity of the MLP structure.

In order to compare all of the MLPs implemented for this coursework, I have created a table which compares each method, with each number of inputs by their prediction vs actual correlations and their statistical indices:



	MLP Structure											
Hidden =	1				3				5			
Input Nodes:	2	3	4	5	2	3	4	5	2	3	4	5
Correlation	95.545%	95.485%	95.585%	95.436%	95.518%	95.466%	95.579%	95.414%	95.525%	95.430%	95.562%	95.415%
RMSE	0.0062	0.0063	0.0062	0.0062	0.0062	0.0063	0.0061	0.0062	0.0063	0.0063	0.0061	0.0062
MAE	0.0047	0.0046	0.0046	0.0046	0.0047	0.0047	0.0047	0.0047	0.0047	0.0048	0.0047	0.0047
MAPE	0.355%	0.352%	0.347%	0.349%	0.353%	0.356%	0.353%	0.356%	0.360%	0.360%	0.355%	0.355%

	MLP Structure											
Hidden =	c(1, 1)				c(3, 1)				c(5, 1)			
Input Nodes:	2	3	4	5	2	3	4	5	2	3	4	5
Correlation	95.538%	95.483%	95.511%	95.449%	95.536%	95.495%	95.634%	95.413%	95.545%	95.390%	95.575%	95.412%
RMSE	0.0063	0.0063	0.0063	0.0062	0.0062	0.0062	0.0061	0.0062	0.0062	0.0063	0.0062	0.0062
MAE	0.0046	0.0046	0.0046	0.0046	0.0047	0.0047	0.0047	0.0047	0.0046	0.0048	0.0046	0.0048
MAPE	0.349%	0.352%	0.348%	0.348%	0.360%	0.360%	0.353%	0.356%	0.349%	0.361%	0.352%	0.361%

	MLP Structure											
Hidden =	c(3, 3)				c(5, 3)				c(5, 5)			
Input Nodes:	2	3	4	5	2	3	4	5	2	3	4	5
Correlation	95.533%	95.472%	95.531%	95.457%	95.570%	95.423%	95.668%	95.506%	95.535%	95.515%	95.558%	95.439%
RMSE	0.0062	0.0063	0.0062	0.0062	0.0062	0.0063	0.0061	0.0062	0.0062	0.0062	0.0062	0.0062
MAE	0.0046	0.0047	0.0046	0.0046	0.0047	0.0047	0.0047	0.0046	0.0047	0.0047	0.0046	0.0047
MAPE	0.350%	0.355%	0.349%	0.350%	0.356%	0.358%	0.356%	0.351%	0.355%	0.354%	0.348%	0.355%

Since I’ve had 38 different MLP implementations to compare, the table became rather large. As such, I will split up this table in to 9 pieces, to represent each hidden layer/node combination with different inputs. That said, the comparison and result has been between all 38 of these.

Hidden =	1			
Input Nodes:	2	3	4	5
Correlation	95.545%	95.485%	95.585%	95.436%
RMSE	0.0061984	0.0062779	0.0061600	0.0062170
MAE	0.0046741	0.0046324	0.0045797	0.0045986
MAPE	0.355%	0.352%	0.347%	0.349%

Hidden =	3			
Input Nodes:	2	3	4	5
Correlation	95.518%	95.466%	95.579%	95.414%
RMSE	0.0062390	0.0062629	0.0061378	0.0062049
MAE	0.0046544	0.0046916	0.0046563	0.0046920
MAPE	0.353%	0.356%	0.353%	0.356%

Hidden =	5			
Input Nodes:	2	3	4	5
Correlation	95.525%	95.430%	95.562%	95.415%
RMSE	0.0062571	0.0062849	0.0061426	0.0062137
MAE	0.0047444	0.0047506	0.0046841	0.0046852
MAPE	0.360%	0.360%	0.355%	0.355%

Hidden =	c(1, 1)			
Input Nodes:	2	3	4	5
Correlation	95.538%	95.483%	95.511%	95.449%
RMSE	0.0062581	0.0062902	0.0062585	0.0062125
MAE	0.0045937	0.0046332	0.0045871	0.0045834
MAPE	0.349%	0.352%	0.348%	0.348%

Hidden =	c(3, 1)			
Input Nodes:	2	3	4	5
Correlation	95.536%	95.495%	95.634%	95.413%
RMSE	0.0062291	0.0062467	0.0061235	0.0062120
MAE	0.0047451	0.0047462	0.0046508	0.0046985
MAPE	0.360%	0.360%	0.353%	0.356%

Hidden =	c(5, 1)			
Input Nodes:	2	3	4	5
Correlation	95.545%	95.390%	95.575%	95.412%
RMSE	0.0061977	0.0063194	0.0061705	0.0062206
MAE	0.0045933	0.0047639	0.0046487	0.0047683
MAPE	0.349%	0.361%	0.352%	0.361%

Hidden =	c(3, 3)			
Input Nodes:	2	3	4	5
Correlation	95.533%	95.472%	95.531%	95.457%
RMSE	0.0062179	0.0062677	0.0061879	0.0061775
MAE	0.0046113	0.0046766	0.0046071	0.0046226
MAPE	0.350%	0.355%	0.349%	0.350%

Hidden =	c(5, 3)			
Input Nodes:	2	3	4	5
Correlation	95.570%	95.423%	95.668%	95.506%
RMSE	0.0061917	0.0062867	0.0061317	0.0061505
MAE	0.0046938	0.0047197	0.0046990	0.0046371
MAPE	0.356%	0.358%	0.356%	0.351%

Hidden =	c(5, 5)			
Input Nodes:	2	3	4	5
Correlation	95.535%	95.515%	95.558%	95.439%
RMSE	0.0062051	0.0062159	0.0061540	0.0062013
MAE	0.0046758	0.0046657	0.0045988	0.0046850
MAPE	0.355%	0.354%	0.348%	0.355%

In the tables above, the cells which are highlighted green are those which have the best performance given by that statistic out of the 38 MLP implementations. Cells highlighted in yellow are the second best performing out of the 38 for that given statistic.

From this comparison table, we can see that all models were rather great, with little variation between them. That said, the one that has the best performance given our 4 testing statistics would be the MLP with 4 input nodes, and one hidden layer with a single hidden node. This is because the MAE and MAPE values for this model are the lowest from the whole lot, meaning that the accuracy is the closest to the real, expected result, with the smallest margin of error. The values of MAE and MAPE for this model are 0.00458 and 0.347% respectively.

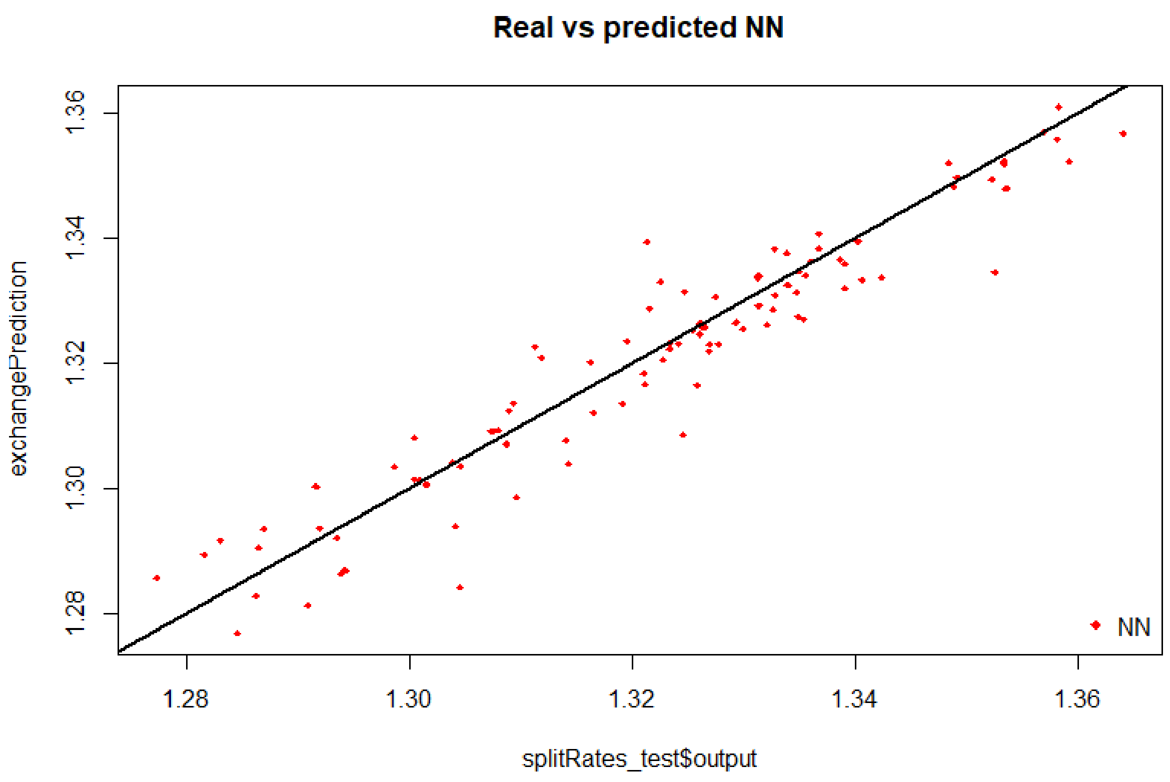
The second best, judged by these same indices, would be a MLP NN with 5 inputs and 2 hidden layers, with 1 hidden node each. This model had the performance with MAE at 0.00458 and a MAPE of 0.348%.

If we judge by Correlation and RMSE instead, we can see that the second best is a tie between a model with 4 inputs, 2 hidden layers; one with 3 hidden nodes and one with 1 hidden node, and a model with 4 inputs, 2 hidden layers; one with 5 hidden nodes and one with 3 hidden nodes. The first has the lowest RMSE from the lot at 0.00612 and the second highest correlation at 95.634%, while the latter has the best correlation of all with 95.668% and the second lowest RMSE at 0.00613.

DISCUSSION ON THE MEANING OF USED STATISTICAL INDICES

The three statistical indices that were used were RMSE (Root Mean Squared Error), MAE (Mean Absolute Error) and MAPE (Mean Absolute Percentage Error).

RMSE is the standard deviation of prediction errors within our result. It is a measure of how spread out our predictions are from the line of best fit. If we were to plot our predictions against the actual exchange rate values, we would get a plot that would look something like this:



RMSE would show us how spread out the deviations are of the red points from the line of best fit. The lower the value of RMSE, the better the accuracy of our model. The issue with RMSE is that it is not scaled, meaning that two datasets with different scaling will produce vastly different RMSE values, which might make them difficult to compare.

MAE shows the mean absolute error in the results. This is the average error across the data averaged as if the value was a positive, no matter whether it is positive or negative. This is good, because it shows you the actual average error in the data, however, it suffers from the same issue that RMSE does, in that it is not scaled. Meaning that it is very difficult if a value of MAE is good, or bad, without the context of the scaling of the original data.

MAPE is similar to MAE, but the value is scaled and given as a percentage. As such, it is obvious whether a MAPE value is good or bad, as you do not need to relate it to the scaling of the original dataset. That said, MAPE alone might not be the best indicator of accuracy for forecasting applications, as each error is divided individually, and as such may be skewed (Vandeput, 2019).

DISCUSS THE ISSUE OF EFFICIENCY

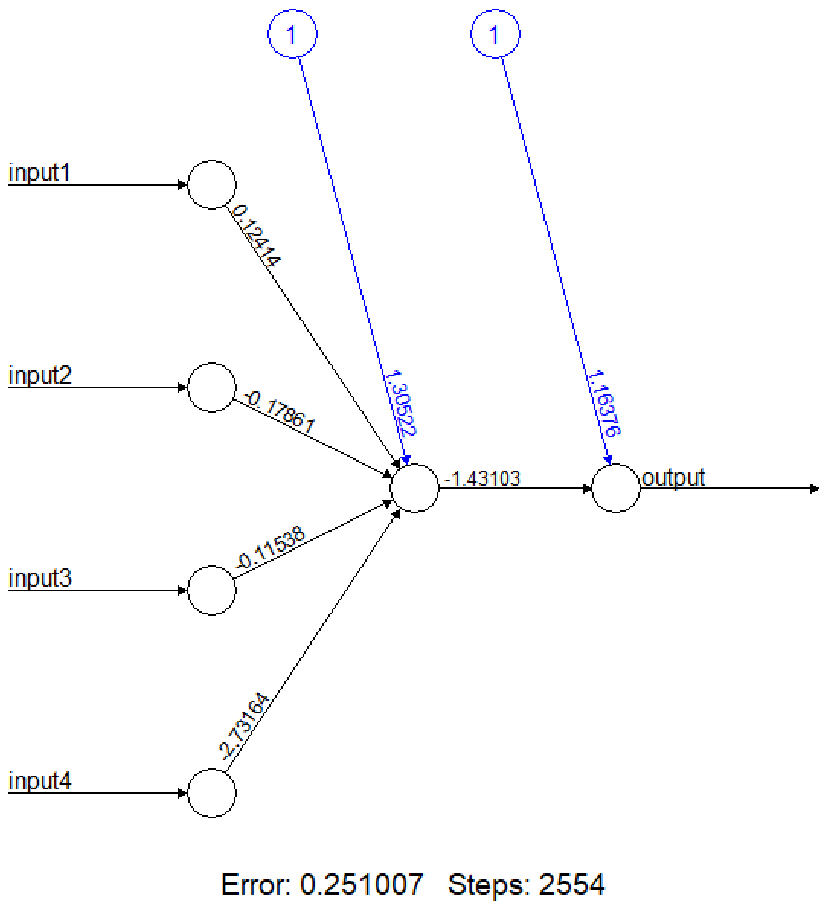
For my two, best performing MLPs, the main issue with efficiency comes from the model's structure complexity, since the initial efficiency issues have been handled by scaling the data, making it easier for the NN to find optimal values for each input node.

Currently however, one of the NNs that had the second-best performance had 4 input nodes and 2 hidden layers with 5 and 3 hidden input nodes, respectively. This NN would have to perform more calculations to arrive at a predictions, when compared to the other NN that had the actual best performance, which also had 4 input nodes, but only had one hidden layer with a single hidden node inside of it.

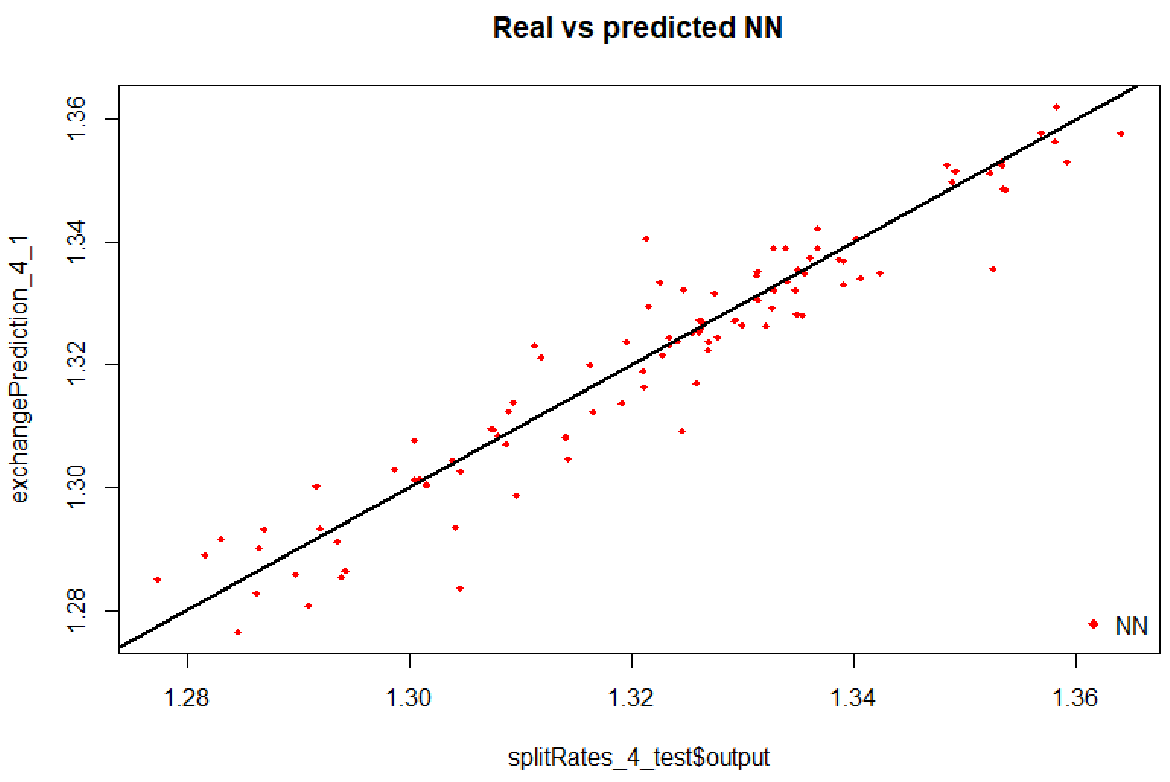
It seems that, depending on the statistical indices used for measurement, more hidden layers and more nodes may have a positive effect on the result accuracy, however, this may also come at the cost of increased computational cost, both during training of the model, as well as during use for forecasting.

PROVIDE BEST RESULTS GRAPHICALLY AND VIA PERFORMANCE INDICES

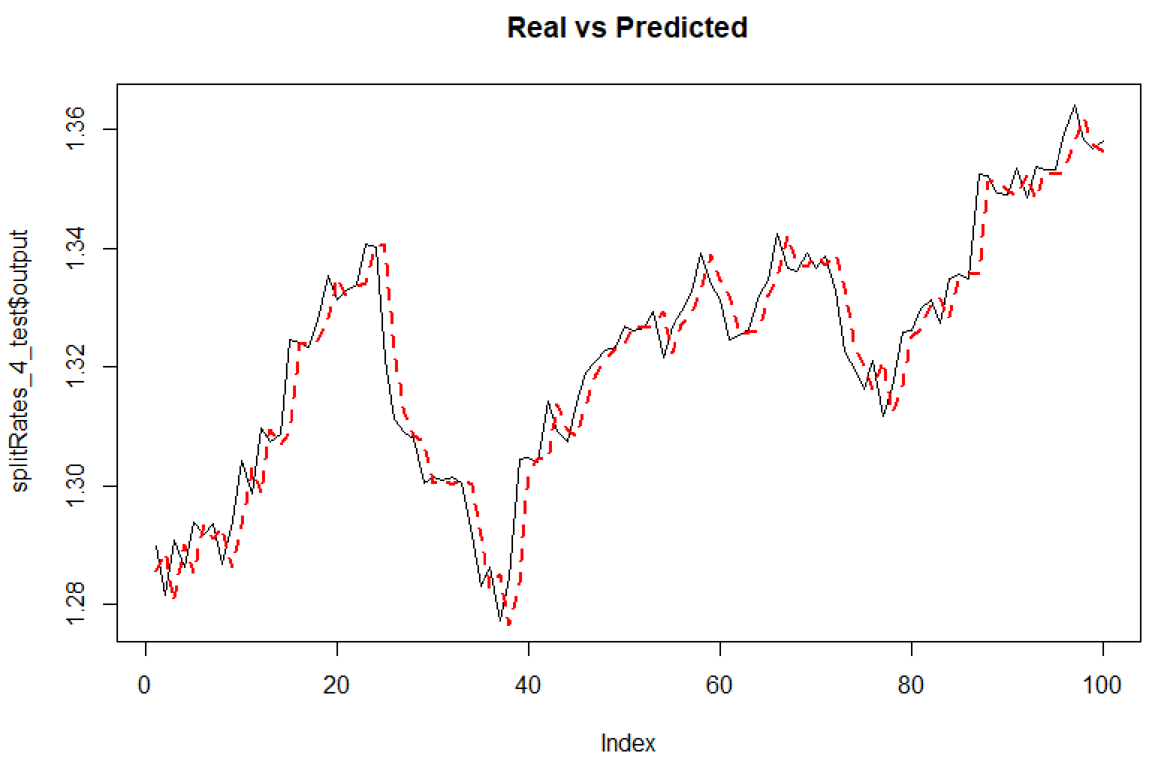
From my comparison table, I have deducted that the best performing model was the one with 4 input nodes and 1 hidden layer with a single hidden node. When plotted, this model appears like this:



This model has produced the following prediction when plotted against the actual results:



And when the prediction and actual rates are plotted side by side, the results look like this:

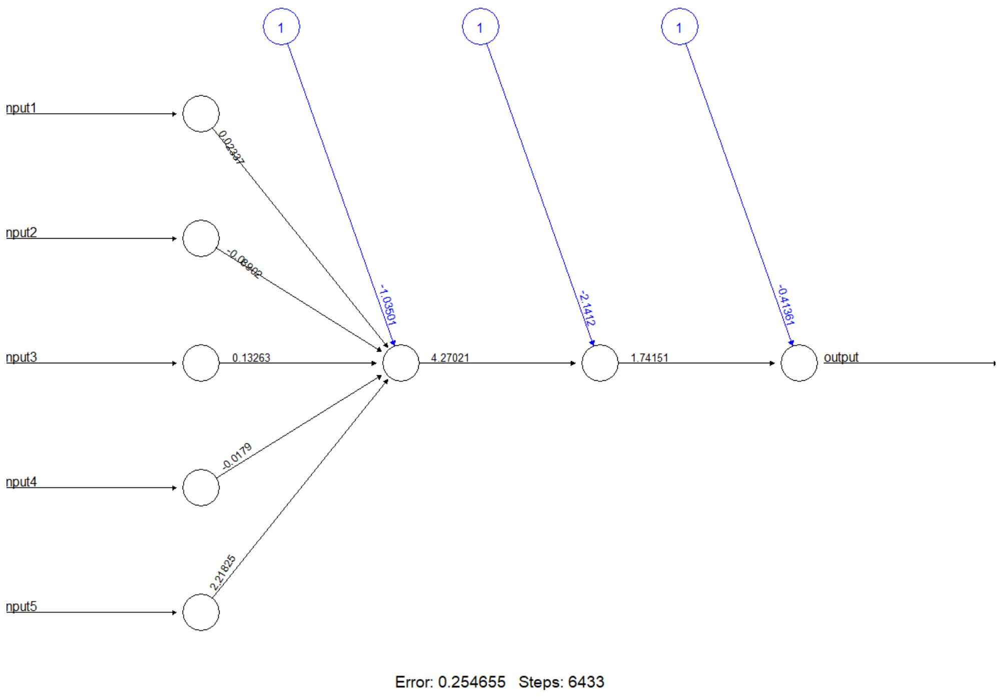


In the plot above, the black line represents actual exchange rate values, and the red line represents predictions.

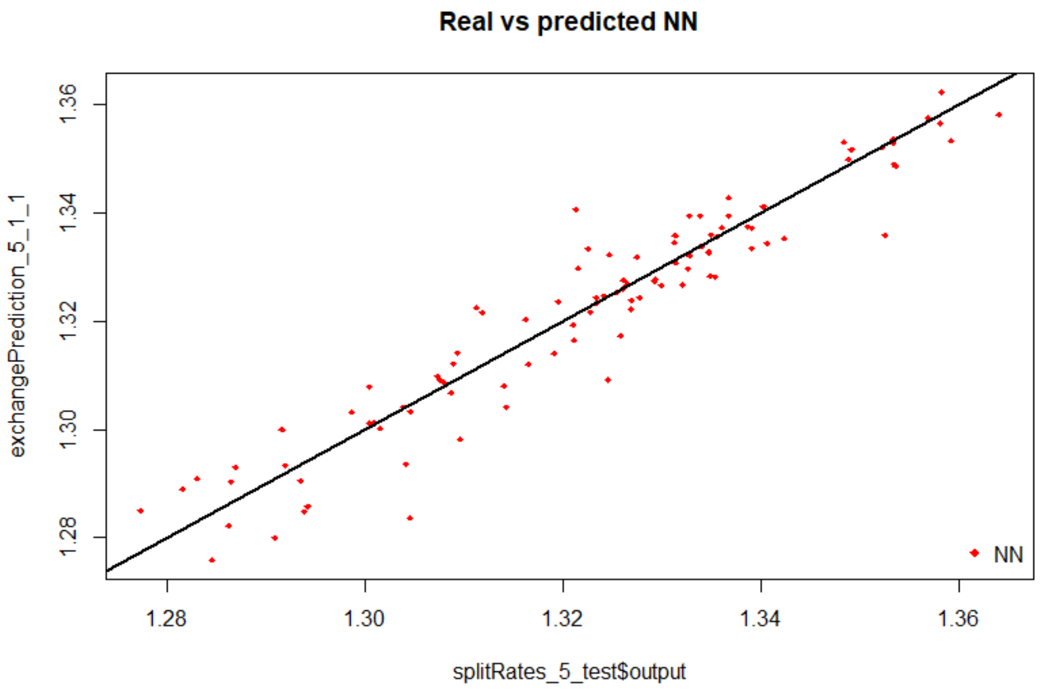
This model has produced the following values for the statistical indices used to measure its performance:

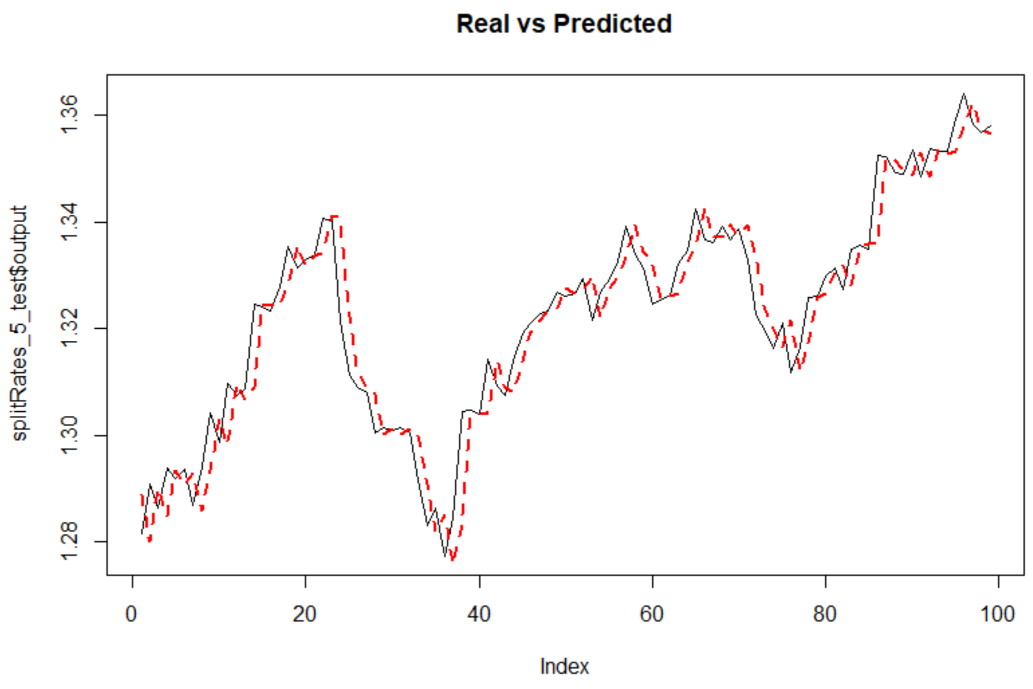
- Correlation: **95.585%**
- RMSE: **0.0061600**
- MAE: **0.0045797** *(Lowest)*
- MAPE: **0.347%** *(Lowest)*

The second-best performing model, when judged by the same statistical indices, was the one with 5 input nodes and 2 hidden layers, with one hidden node in each. When plotted, this model looks like so:



The model has produced the following results, plotted in a similar fashion to the previous one:

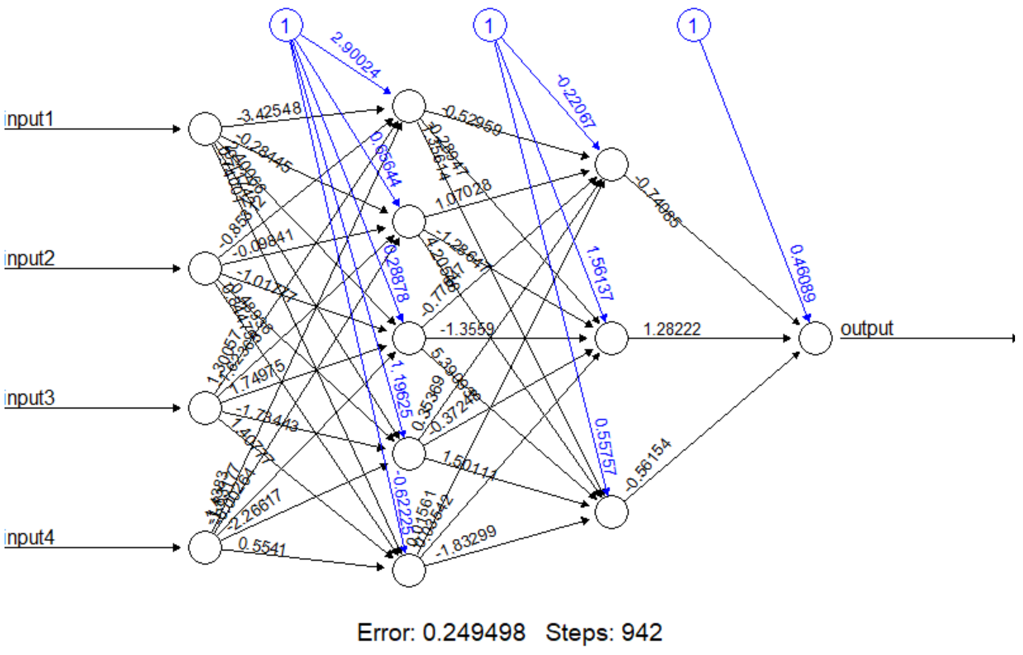




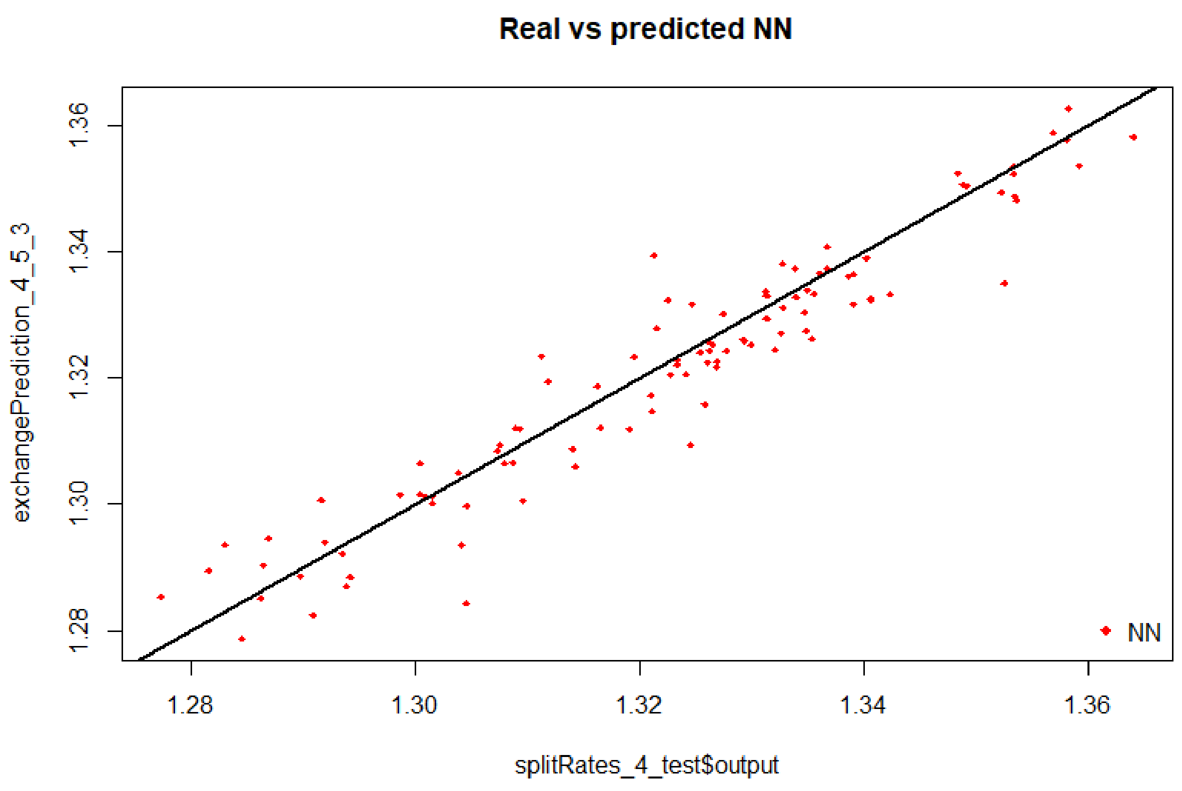
This model has produced the following values for the statistical indices used to measure its performance:

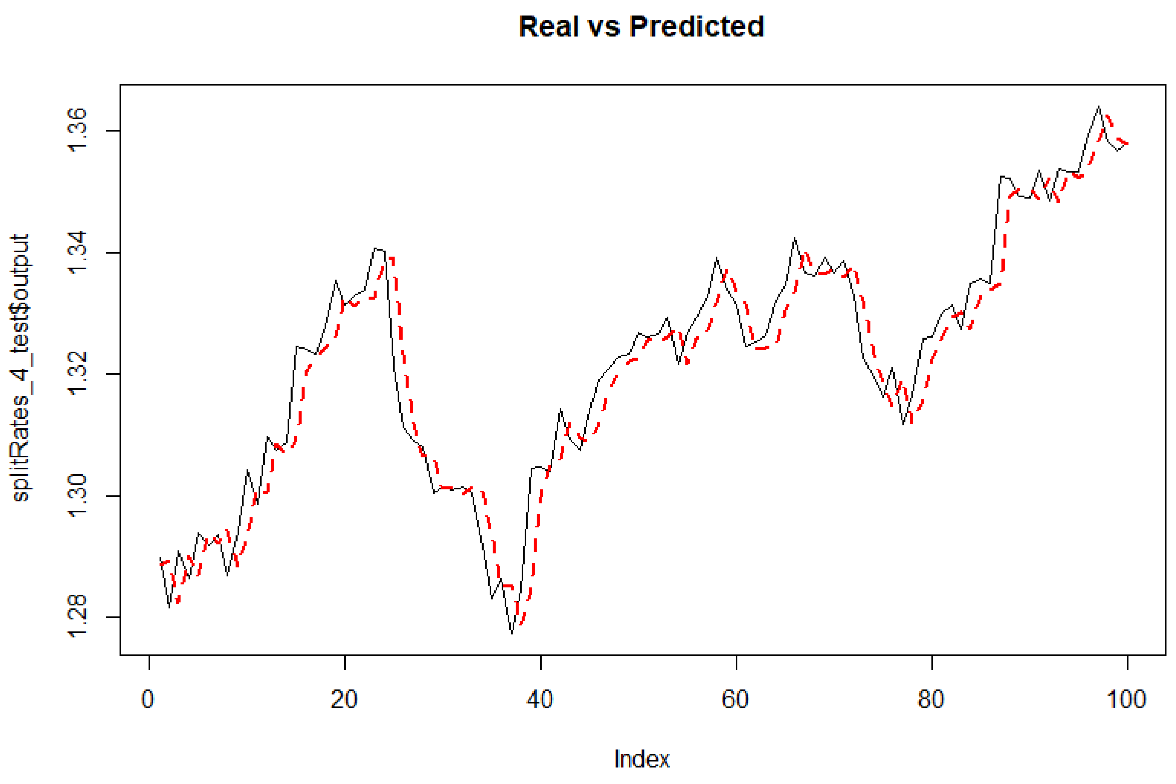
- Correlation: **95.449%**
- RMSE: **0.0062125**
- MAE: **0.0045834** *(Second Lowest)*
- MAPE: **0.348%** *(Second Lowest)*

There were two other models which showed to have promising performance. One which had the highest correlation, and second lowest RMSE and one which had the lowest RMSE value, and the second highest correlation. The first one, with the highest correlation, was the model with 4 input nodes and 2 hidden layers, with 5 and 3 hidden nodes, respectively. Here is this model plotted below:



The results produced by this model are plotted below:

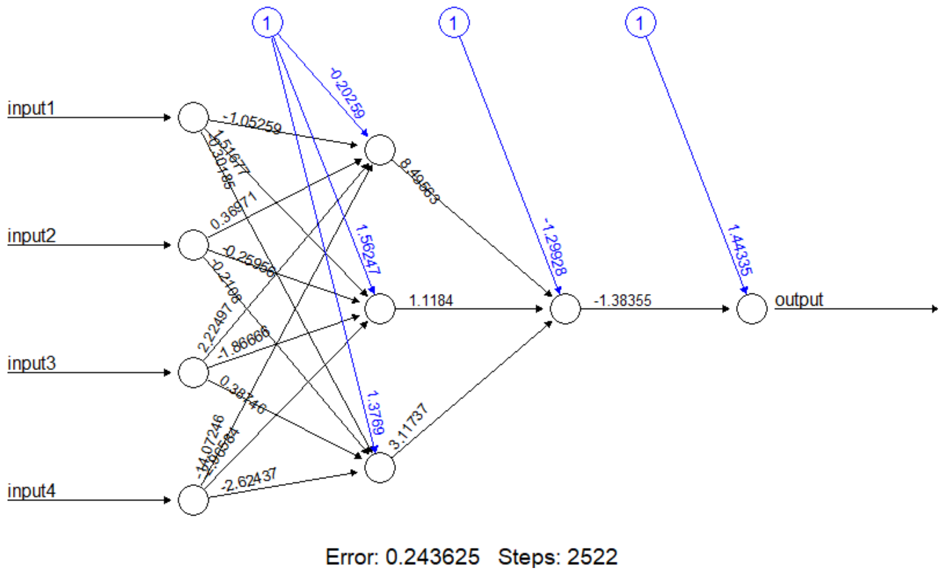




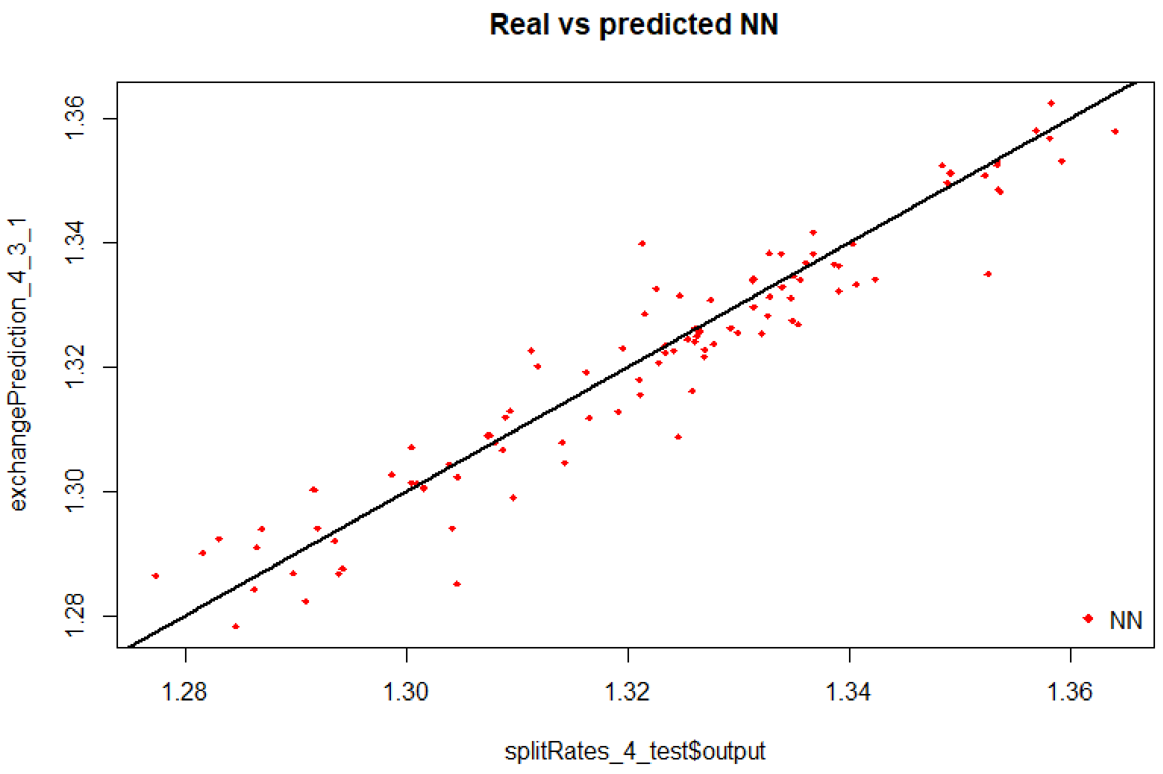
This model has produced the following values for the statistical indices used to measure its performance:

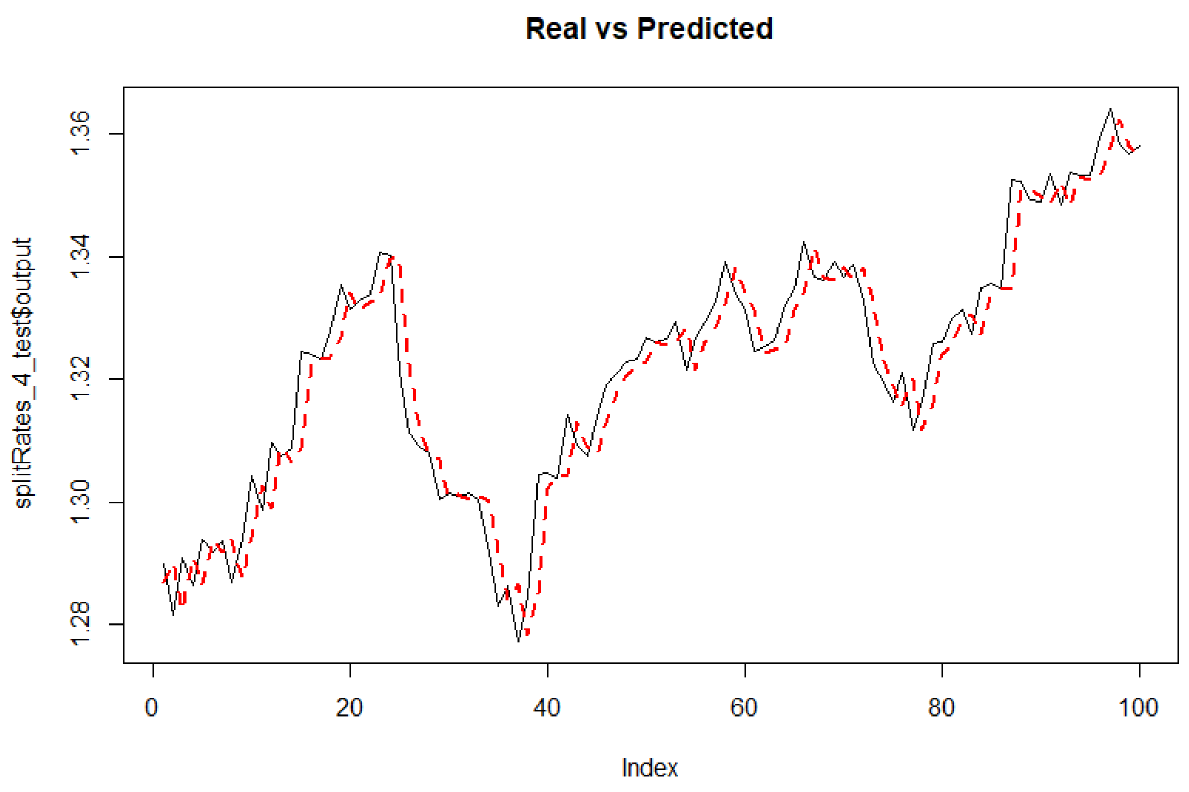
- Correlation: **95.668%** *(Highest)*
- RMSE: **0.0061317** *(Second Lowest)*
- MAE: **0.0046990**
- MAPE: **0.356%**

The second model was the one with the lowest RMSE and the second highest correlation. This model was the one with 4 input nodes and 2 hidden layers which had 3 and 1 hidden nodes, respectively. When plotted, this model looks like this:



This model produced the following results plotted against actual exchange rate values:





This model has produced the following values for the statistical indices used to measure its performance:

- Correlation: **95.634%** *(Second Highest)*
- RMSE: **0.0061235** *(Lowest)*
- MAE: **0.0046508**
- MAPE: **0.353%**

All of these result plots have been created with the following code, passing in different test datasets and predictions as arguments:

```
par(mfrow=c(1,1))
plot(splitRates_4_test$output, exchangePrediction_4_3_1 ,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN', pch=18,col='red', bty='n')

plot(splitRates_4_test$output, type = "l", col="black", main = "Real vs Predicted")
lines(exchangePrediction_4_3_1, lty=2, lwd=2, col="red")
```

MLP APPENDIX (R SCRIPT)

```
library("readxl")
library("neuralnet")
library("Metrics")

#-------------- Functions -----------------------------------------------
------

#Function which splits the time-series data into input / output vectors
splitExchangeRates <-  function(data, steps) {

  m <- matrix(ncol = steps+1)
  if (steps == 2) {
    colnames(m) <- c("input1", "input2", "output")
  } else if (steps == 3) {
    colnames(m) <- c("input1", "input2", "input3", "output")
  } else if (steps == 4) {
    colnames(m) <- c("input1", "input2", "input3", "input4", "output")
  } else if (steps == 5) {
    colnames(m) <- c("input1", "input2", "input3", "input4", "input5",
"output")
  }

  for (i in 1:(length(data)-(steps+1))) {

    v <- c(data[i:(i+steps)])

    m <- rbind(m, v)

  }

  return(m[-1,])

}

#Funciton for normalisisng data
normalise <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

#Function to undo normalisation
unnormalise <- function(x, min, max) {
  return( (max - min)*x + min )
}

#Function to calculate RMSE
###rmse <- function(error)
###{
###  sqrt(mean(error^2))
###}

#--------------- Initial Data Frames -------------------------------------
------

#Load data from Excel
exchangeRateData = read_excel("ExchangeUSD.xlsx")

#Just the column which only has the exchange rate value
rates <- exchangeRateData$`USD/EUR`

#--------------- Input/Output Matrices & Normalisation --------------------
---------

#--- m = 2
splitRates_2 <- as.data.frame(splitExchangeRates(rates, 2))
splitRates_2
splitRatesNormalised_2 <- as.data.frame(lapply(splitRates_2, normalise))
splitRatesNormalised_2

splitRates_2_train <- splitRates_2[1:395,] #~80%
splitRates_2_test <- splitRates_2[396:497,] #~20%
```

```
splitRatesNormalised_2_train <- splitRatesNormalised_2[1:395,] #~80%
splitRatesNormalised_2_test <- splitRatesNormalised_2[396:497,] #~20%

#--- m = 3
splitRates_3 <- as.data.frame(splitExchangeRates(rates, 3))
splitRatesNormalised_3 <- as.data.frame(lapply(splitRates_3, normalise))

splitRates_3_train <- splitRates_3[1:395,] #~80%
splitRates_3_test <- splitRates_3[396:496,] #~20%

splitRatesNormalised_3_train <- splitRatesNormalised_3[1:395,] #~80%
splitRatesNormalised_3_test <- splitRatesNormalised_3[396:496,] #~20%

#--- m = 4
splitRates_4 <- as.data.frame(splitExchangeRates(rates, 4))
splitRatesNormalised_4 <- as.data.frame(lapply(splitRates_4, normalise))

splitRates_4_train <- splitRates_4[1:395,] #~80%
splitRates_4_test <- splitRates_4[396:495,] #~20%

splitRatesNormalised_4_train <- splitRatesNormalised_4[1:395,] #~80%
splitRatesNormalised_4_test <- splitRatesNormalised_4[396:495,] #~20%

#--- m = 5
splitRates_5 <- as.data.frame(splitExchangeRates(rates, 5))
splitRates_5
splitRatesNormalised_5 <- as.data.frame(lapply(splitRates_5, normalise))

splitRates_5_train <- splitRates_5[1:395,] #~80%
splitRates_5_test <- splitRates_5[396:494,] #~20%

splitRatesNormalised_5_train <- splitRatesNormalised_5[1:395,] #~80%
splitRatesNormalised_5_test <- splitRatesNormalised_5[396:494,] #~20%


#-------------- NN Models & Evaluation ----------------------------------
-------------------

#--- hn(l1) = 1

#- i=2

#Create the NN model
set.seed(100)
exchangeModel_2_1 <- neuralnet(output ~ input1 + input2,
                           data = splitRatesNormalised_2_train,
                           hidden = 1)

#Plot the NN model
plot(exchangeModel_2_1)

#Obtain the NN model results
exchangeResults_2_1 <- compute(exchangeModel_2_1,
splitRatesNormalised_2_test[1:2])

#Corellation between prediction and actual results
cor(exchangeResults_2_1$net.result, splitRatesNormalised_2_test$output)

#Find Min and Max Value
testRates_2_min <- min(splitRates_2_train$output)
```

```
testRates_2_max <- max(splitRates_2_train$output)

exchangePrediction_2_1 <- unnormalise(exchangeResults_2_1$net.result,
                                      testRates_2_min,
                                      testRates_2_max)
exchangePrediction_2_1

#RMSE
rmse_2_1 <- rmse(splitRates_2_test$output, exchangePrediction_2_1)
rmse_2_1

#MAE
mae_2_1 <- mae(splitRates_2_test$output, exchangePrediction_2_1)
mae_2_1

#MAPE
mape_2_1 <- mape(splitRates_2_test$output, exchangePrediction_2_1)
mape_2_1

#- i=3

#Create the NN model
set.seed(100)
exchangeModel_3_1 <- neuralnet(output ~ input1 + input2 + input3,
                               data = splitRatesNormalised_3_train,
                               hidden = 1)

#Plot the NN model
plot(exchangeModel_3_1)

#Obtain the NN model results
exchangeResults_3_1 <- compute(exchangeModel_3_1,
splitRatesNormalised_3_test[1:3])

#Corellation between prediction and actual results
cor(exchangeResults_3_1$net.result, splitRatesNormalised_3_test$output)

#Find Min and Max Value
testRates_3_min <- min(splitRates_3_train$output)
testRates_3_max <- max(splitRates_3_train$output)

exchangePrediction_3_1 <- unnormalise(exchangeResults_3_1$net.result,
testRates_3_min, testRates_3_max)
exchangePrediction_3_1

#RMSE
rmse_3_1 <- rmse(splitRates_3_test$output, exchangePrediction_3_1)
rmse_3_1

#MAE
mae_3_1 <- mae(splitRates_3_test$output, exchangePrediction_3_1)
mae_3_1

#MAPE
mape_3_1 <- mape(splitRates_3_test$output, exchangePrediction_3_1)
mape_3_1

#- i=4

#Create the NN model
```

```
set.seed(100)
exchangeModel_4_1 <- neuralnet(output ~ input1 + input2 + input3 +
input4,
                               data = splitRatesNormalised_4_train,
                               hidden = 1)

#Plot the NN model
plot(exchangeModel_4_1)

#Obtain the NN model results
exchangeResults_4_1 <- compute(exchangeModel_4_1,
splitRatesNormalised_4_test[1:4])

#Corellation between prediction and actual results
cor(exchangeResults_4_1$net.result, splitRatesNormalised_4_test$output)

#Find Min and Max Value
testRates_4_min <- min(splitRates_4_train$output)
testRates_4_max <- max(splitRates_4_train$output)

exchangePrediction_4_1 <- unnormalise(exchangeResults_4_1$net.result,
testRates_4_min, testRates_4_max)
exchangePrediction_4_1

#RMSE
rmse_4_1 <- rmse(splitRates_4_test$output, exchangePrediction_4_1)
rmse_4_1

#MAE
mae_4_1 <- mae(splitRates_4_test$output, exchangePrediction_4_1)
mae_4_1

#MAPE
mape_4_1 <- mape(splitRates_4_test$output, exchangePrediction_4_1)
mape_4_1

#Plot Results
par(mfrow=c(1,1))
plot(splitRates_4_test$output, exchangePrediction_4_1
,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN', pch=18,col='red', bty='n')

plot(splitRates_4_test$output, type = "l", col="black", main = "Real vs
Predicted")
lines(exchangePrediction_4_1, lty=2, lwd=2, col="red")

#- i=5

#Create the NN model
set.seed(100)
exchangeModel_5_1 <- neuralnet(output ~ input1 + input2 + input3 + input4
+ input5,
                               data = splitRatesNormalised_5_train,
                               hidden = 1)

#Plot the NN model
plot(exchangeModel_5_1)

#Obtain the NN model results
```

```
exchangeResults_5_1 <- compute(exchangeModel_5_1,
splitRatesNormalised_5_test[1:5])

#Corellation between prediction and actual results
cor(exchangeResults_5_1$net.result, splitRatesNormalised_5_test$output)

#Find Min and Max Value
testRates_5_min <- min(splitRates_5_train$output)
testRates_5_max <- max(splitRates_5_train$output)

exchangePrediction_5_1 <- unnormalise(exchangeResults_5_1$net.result,
testRates_5_min, testRates_5_max)
exchangePrediction_5_1

#RMSE
rmse_5_1 <- rmse(splitRates_5_test$output, exchangePrediction_5_1)
rmse_5_1

#MAE
mae_5_1 <- mae(splitRates_5_test$output, exchangePrediction_5_1)
mae_5_1

#MAPE
mape_5_1 <- mape(splitRates_5_test$output, exchangePrediction_5_1)
mape_5_1

#--- hn(l1) = 3

#- i=2

#Create the NN model
set.seed(100)
exchangeModel_2_3 <- neuralnet(output ~ input1 + input2,
                               data = splitRatesNormalised_2_train,
                               hidden = 3)

#Plot the NN model
plot(exchangeModel_2_3)

#Obtain the NN model results
exchangeResults_2_3 <- compute(exchangeModel_2_3,
splitRatesNormalised_2_test[1:2])

#Corellation between prediction and actual results
cor(exchangeResults_2_3$net.result, splitRatesNormalised_2_test$output)

#Find Min and Max Value
testRates_2_min <- min(splitRates_2_train$output)
testRates_2_max <- max(splitRates_2_train$output)

exchangePrediction_2_3 <- unnormalise(exchangeResults_2_3$net.result,
testRates_2_min, testRates_2_max)
exchangePrediction_2_3

#RMSE
rmse_2_3 <- rmse(splitRates_2_test$output, exchangePrediction_2_3)
rmse_2_3

#MAE
mae_2_3 <- mae(splitRates_2_test$output, exchangePrediction_2_3)
```

```
mae_2_3

#MAPE
mape_2_3 <- mape(splitRates_2_test$output, exchangePrediction_2_3)
mape_2_3

#- i=3

#Create the NN model
set.seed(100)
exchangeModel_3_3 <- neuralnet(output ~ input1 + input2 + input3,
                               data = splitRatesNormalised_3_train,
                               hidden = 3)

#Plot the NN model
plot(exchangeModel_3_3)

#Obtain the NN model results
exchangeResults_3_3 <- compute(exchangeModel_3_3,
splitRatesNormalised_3_test[1:3])

#Corellation between prediction and actual results
cor(exchangeResults_3_3$net.result, splitRatesNormalised_3_test$output)

#Find Min and Max Value
testRates_3_min <- min(splitRates_3_train$output)
testRates_3_max <- max(splitRates_3_train$output)

exchangePrediction_3_3 <- unnormalise(exchangeResults_3_3$net.result,
testRates_3_min, testRates_3_max)
exchangePrediction_3_3

#RMSE
rmse_3_3 <- rmse(splitRates_3_test$output, exchangePrediction_3_3)
rmse_3_3

#MAE
mae_3_3 <- mae(splitRates_3_test$output, exchangePrediction_3_3)
mae_3_3

#MAPE
mape_3_3 <- mape(splitRates_3_test$output, exchangePrediction_3_3)
mape_3_3

#- i=4

#Create the NN model
set.seed(100)
exchangeModel_4_3 <- neuralnet(output ~ input1 + input2 + input3 +
input4,
                               data = splitRatesNormalised_4_train,
                               hidden = 3)

#Plot the NN model
plot(exchangeModel_4_3)

#Obtain the NN model results
exchangeResults_4_3 <- compute(exchangeModel_4_3,
splitRatesNormalised_4_test[1:4])
```

```
#Corellation between prediction and actual results
cor(exchangeResults_4_3$net.result, splitRatesNormalised_4_test$output)

#Find Min and Max Value
testRates_4_min <- min(splitRates_4_train$output)
testRates_4_max <- max(splitRates_4_train$output)

exchangePrediction_4_3 <- unnormalise(exchangeResults_4_3$net.result,
testRates_4_min, testRates_4_max)
exchangePrediction_4_3

#RMSE
rmse_4_3 <- rmse(splitRates_4_test$output, exchangePrediction_4_3)
rmse_4_3

#MAE
mae_4_3 <- mae(splitRates_4_test$output, exchangePrediction_4_3)
mae_4_3

#MAPE
mape_4_3 <- mape(splitRates_4_test$output, exchangePrediction_4_3)
mape_4_3

#- i=5

#Create the NN model
set.seed(100)
exchangeModel_5_3 <- neuralnet(output ~ input1 + input2 + input3 + input4
+ input5,
                              data = splitRatesNormalised_5_train,
                              hidden = 3)

#Plot the NN model
plot(exchangeModel_5_3)

#Obtain the NN model results
exchangeResults_5_3 <- compute(exchangeModel_5_3,
splitRatesNormalised_5_test[1:5])

#Corellation between prediction and actual results
cor(exchangeResults_5_3$net.result, splitRatesNormalised_5_test$output)

#Find Min and Max Value
testRates_5_min <- min(splitRates_5_train$output)
testRates_5_max <- max(splitRates_5_train$output)

exchangePrediction_5_3 <- unnormalise(exchangeResults_5_3$net.result,
testRates_5_min, testRates_5_max)
exchangePrediction_5_3

#RMSE
rmse_5_3 <- rmse(splitRates_5_test$output, exchangePrediction_5_3)
rmse_5_3

#MAE
mae_5_3 <- mae(splitRates_5_test$output, exchangePrediction_5_3)
mae_5_3

#MAPE
mape_5_3 <- mape(splitRates_5_test$output, exchangePrediction_5_3)
```

```
mape_5_3

#--- hn(l1) = 5

#- i=2

#Create the NN model
set.seed(100)
exchangeModel_2_5 <- neuralnet(output ~ input1 + input2,
                               data = splitRatesNormalised_2_train,
                               hidden = 5)

#Plot the NN model
plot(exchangeModel_2_5)

#Obtain the NN model results
exchangeResults_2_5 <- compute(exchangeModel_2_5,
splitRatesNormalised_2_test[1:2])

#Corellation between prediction and actual results
cor(exchangeResults_2_5$net.result, splitRatesNormalised_2_test$output)

#Find Min and Max Value
testRates_2_min <- min(splitRates_2_train$output)
testRates_2_max <- max(splitRates_2_train$output)

exchangePrediction_2_5 <- unnormalise(exchangeResults_2_5$net.result,
testRates_2_min, testRates_2_max)
exchangePrediction_2_5

#RMSE
rmse_2_5 <- rmse(splitRates_2_test$output, exchangePrediction_2_5)
rmse_2_5

#MAE
mae_2_5 <- mae(splitRates_2_test$output, exchangePrediction_2_5)
mae_2_5

#MAPE
mape_2_5 <- mape(splitRates_2_test$output, exchangePrediction_2_5)
mape_2_5

#- i=3

#Create the NN model
set.seed(100)
exchangeModel_3_5 <- neuralnet(output ~ input1 + input2 + input3,
                               data = splitRatesNormalised_3_train,
                               hidden = 5)

#Plot the NN model
plot(exchangeModel_3_5)

#Obtain the NN model results
exchangeResults_3_5 <- compute(exchangeModel_3_5,
splitRatesNormalised_3_test[1:3])

#Corellation between prediction and actual results
cor(exchangeResults_3_5$net.result, splitRatesNormalised_3_test$output)
```

```
#Find Min and Max Value
testRates_3_min <- min(splitRates_3_train$output)
testRates_3_max <- max(splitRates_3_train$output)

exchangePrediction_3_5 <- unnormalise(exchangeResults_3_5$net.result,
testRates_3_min, testRates_3_max)
exchangePrediction_3_5

#RMSE
rmse_3_5 <- rmse(splitRates_3_test$output, exchangePrediction_3_5)
rmse_3_5

#MAE
mae_3_5 <- mae(splitRates_3_test$output, exchangePrediction_3_5)
mae_3_5

#MAPE
mape_3_5 <- mape(splitRates_3_test$output, exchangePrediction_3_5)
mape_3_5

#- i=4

#Create the NN model
set.seed(100)
exchangeModel_4_5 <- neuralnet(output ~ input1 + input2 + input3 +
input4,
                              data = splitRatesNormalised_4_train,
                              hidden = 5)

#Plot the NN model
plot(exchangeModel_4_5)

#Obtain the NN model results
exchangeResults_4_5 <- compute(exchangeModel_4_5,
splitRatesNormalised_4_test[1:4])

#Corellation between prediction and actual results
cor(exchangeResults_4_5$net.result, splitRatesNormalised_4_test$output)

#Find Min and Max Value
testRates_4_min <- min(splitRates_4_train$output)
testRates_4_max <- max(splitRates_4_train$output)

exchangePrediction_4_5 <- unnormalise(exchangeResults_4_5$net.result,
testRates_4_min, testRates_4_max)
exchangePrediction_4_5

#RMSE
rmse_4_5 <- rmse(splitRates_4_test$output, exchangePrediction_4_5)
rmse_4_5

#MAE
mae_4_5 <- mae(splitRates_4_test$output, exchangePrediction_4_5)
mae_4_5

#MAPE
mape_4_5 <- mape(splitRates_4_test$output, exchangePrediction_4_5)
mape_4_5

#- i=5
```

```
#Create the NN model
set.seed(100)
exchangeModel_5_5 <- neuralnet(output ~ input1 + input2 + input3 + input4
+ input5,
                              data = splitRatesNormalised_5_train,
                              hidden = 5)

#Plot the NN model
plot(exchangeModel_5_5)

#Obtain the NN model results
exchangeResults_5_5 <- compute(exchangeModel_5_5,
splitRatesNormalised_5_test[1:5])

#Corellation between prediction and actual results
cor(exchangeResults_5_5$net.result, splitRatesNormalised_5_test$output)

#Find Min and Max Value
testRates_5_min <- min(splitRates_5_train$output)
testRates_5_max <- max(splitRates_5_train$output)

exchangePrediction_5_5 <- unnormalise(exchangeResults_5_5$net.result,
testRates_5_min, testRates_5_max)
exchangePrediction_5_5

#RMSE
rmse_5_5 <- rmse(splitRates_5_test$output, exchangePrediction_5_5)
rmse_5_5

#MAE
mae_5_5 <- mae(splitRates_5_test$output, exchangePrediction_5_5)
mae_5_5

#MAPE
mape_5_5 <- mape(splitRates_5_test$output, exchangePrediction_5_5)
mape_5_5

#--- hn(l1) = 1 | hn(l2) = 1

#- i=2

#Create the NN model
set.seed(100)
exchangeModel_2_1_1 <- neuralnet(output ~ input1 + input2,
                              data = splitRatesNormalised_2_train,
                              hidden = c(1, 1))

#Plot the NN model
plot(exchangeModel_2_1_1)

#Obtain the NN model results
exchangeResults_2_1_1 <- compute(exchangeModel_2_1_1,
splitRatesNormalised_2_test[1:2])

#Corellation between prediction and actual results
cor(exchangeResults_2_1_1$net.result, splitRatesNormalised_2_test$output)

#Find Min and Max Value
testRates_2_min <- min(splitRates_2_train$output)
```

```
testRates_2_max <- max(splitRates_2_train$output)

exchangePrediction_2_1_1 <- unnormalise(exchangeResults_2_1_1$net.result,
testRates_2_min, testRates_2_max)
exchangePrediction_2_1_1

#RMSE
rmse_2_1_1 <- rmse(splitRates_2_test$output, exchangePrediction_2_1_1)
rmse_2_1_1

#MAE
mae_2_1_1 <- mae(splitRates_2_test$output, exchangePrediction_2_1_1)
mae_2_1_1

#MAPE
mape_2_1_1 <- mape(splitRates_2_test$output, exchangePrediction_2_1_1)
mape_2_1_1

#- i=3

#Create the NN model
set.seed(100)
exchangeModel_3_1_1 <- neuralnet(output ~ input1 + input2 + input3,
                                 data = splitRatesNormalised_3_train,
                                 hidden = c(1, 1))

#Plot the NN model
plot(exchangeModel_3_1_1)

#Obtain the NN model results
exchangeResults_3_1_1 <- compute(exchangeModel_3_1_1,
splitRatesNormalised_3_test[1:3])

#Corellation between prediction and actual results
cor(exchangeResults_3_1_1$net.result, splitRatesNormalised_3_test$output)

#Find Min and Max Value
testRates_3_min <- min(splitRates_3_train$output)
testRates_3_max <- max(splitRates_3_train$output)

exchangePrediction_3_1_1 <- unnormalise(exchangeResults_3_1_1$net.result,
testRates_3_min, testRates_3_max)
exchangePrediction_3_1_1

#RMSE
rmse_3_1_1 <- rmse(splitRates_3_test$output, exchangePrediction_3_1_1)
rmse_3_1_1

#MAE
mae_3_1_1 <- mae(splitRates_3_test$output, exchangePrediction_3_1_1)
mae_3_1_1

#MAPE
mape_3_1_1 <- mape(splitRates_3_test$output, exchangePrediction_3_1_1)
mape_3_1_1

#- i=4

#Create the NN model
set.seed(100)
```

```
exchangeModel_4_1_1 <- neuralnet(output ~ input1 + input2 + input3 +
input4,
                                 data = splitRatesNormalised_4_train,
                                 hidden = c(1, 1))

#Plot the NN model
plot(exchangeModel_4_1_1)

#Obtain the NN model results
exchangeResults_4_1_1 <- compute(exchangeModel_4_1_1,
splitRatesNormalised_4_test[1:4])

#Corellation between prediction and actual results
cor(exchangeResults_4_1_1$net.result, splitRatesNormalised_4_test$output)

#Find Min and Max Value
testRates_4_min <- min(splitRates_4_train$output)
testRates_4_max <- max(splitRates_4_train$output)

exchangePrediction_4_1_1 <- unnormalise(exchangeResults_4_1_1$net.result,
testRates_4_min, testRates_4_max)
exchangePrediction_4_1_1

#RMSE
rmse_4_1_1 <- rmse(splitRates_4_test$output, exchangePrediction_4_1_1)
rmse_4_1_1

#MAE
mae_4_1_1 <- mae(splitRates_4_test$output, exchangePrediction_4_1_1)
mae_4_1_1

#MAPE
mape_4_1_1 <- mape(splitRates_4_test$output, exchangePrediction_4_1_1)
mape_4_1_1

#- i=5

#Create the NN model
set.seed(100)
exchangeModel_5_1_1 <- neuralnet(output ~ input1 + input2 + input3 +
input4 + input5,
                                 data = splitRatesNormalised_5_train,
                                 hidden = c(1, 1))

#Plot the NN model
plot(exchangeModel_5_1_1)

#Obtain the NN model results
exchangeResults_5_1_1 <- compute(exchangeModel_5_1_1,
splitRatesNormalised_5_test[1:5])

#Corellation between prediction and actual results
cor(exchangeResults_5_1_1$net.result, splitRatesNormalised_5_test$output)

#Find Min and Max Value
testRates_5_min <- min(splitRates_5_train$output)
testRates_5_max <- max(splitRates_5_train$output)

exchangePrediction_5_1_1 <- unnormalise(exchangeResults_5_1_1$net.result,
testRates_5_min, testRates_5_max)
```

```
exchangePrediction_5_1_1

#RMSE
rmse_5_1_1 <- rmse(splitRates_5_test$output, exchangePrediction_5_1_1)
rmse_5_1_1

#MAE
mae_5_1_1 <- mae(splitRates_5_test$output, exchangePrediction_5_1_1)
mae_5_1_1

#MAPE
mape_5_1_1 <- mape(splitRates_5_test$output, exchangePrediction_5_1_1)
mape_5_1_1

#Plot Results
par(mfrow=c(1,1))
plot(splitRates_5_test$output, exchangePrediction_5_1_1
,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN', pch=18,col='red', bty='n')

plot(splitRates_5_test$output, type = "l", col="black", main = "Real vs
Predicted")
lines(exchangePrediction_5_1_1, lty=2, lwd=2, col="red")


#--- hn(l1) = 3 | hn(l2) = 1

#- i=2

#Create the NN model
set.seed(100)
exchangeModel_2_3_1 <- neuralnet(output ~ input1 + input2,
                                  data = splitRatesNormalised_2_train,
                                  hidden = c(3, 1))

#Plot the NN model
plot(exchangeModel_2_3_1)

#Obtain the NN model results
exchangeResults_2_3_1 <- compute(exchangeModel_2_3_1,
splitRatesNormalised_2_test[1:2])

#Corellation between prediction and actual results
cor(exchangeResults_2_3_1$net.result, splitRatesNormalised_2_test$output)

#Find Min and Max Value
testRates_2_min <- min(splitRates_2_train$output)
testRates_2_max <- max(splitRates_2_train$output)

exchangePrediction_2_3_1 <- unnormalise(exchangeResults_2_3_1$net.result,
testRates_2_min, testRates_2_max)
exchangePrediction_2_3_1

#RMSE
rmse_2_3_1 <- rmse(splitRates_2_test$output, exchangePrediction_2_3_1)
rmse_2_3_1

#MAE
mae_2_3_1 <- mae(splitRates_2_test$output, exchangePrediction_2_3_1)
```

```
mae_2_3_1

#MAPE
mape_2_3_1 <- mape(splitRates_2_test$output, exchangePrediction_2_3_1)
mape_2_3_1

#- i=3

#Create the NN model
set.seed(100)
exchangeModel_3_3_1 <- neuralnet(output ~ input1 + input2 + input3,
                                 data = splitRatesNormalised_3_train,
                                 hidden = c(3, 1))

#Plot the NN model
plot(exchangeModel_3_3_1)

#Obtain the NN model results
exchangeResults_3_3_1 <- compute(exchangeModel_3_3_1,
splitRatesNormalised_3_test[1:3])

#Corellation between prediction and actual results
cor(exchangeResults_3_3_1$net.result, splitRatesNormalised_3_test$output)

#Find Min and Max Value
testRates_3_min <- min(splitRates_3_train$output)
testRates_3_max <- max(splitRates_3_train$output)

exchangePrediction_3_3_1 <- unnormalise(exchangeResults_3_3_1$net.result,
testRates_3_min, testRates_3_max)
exchangePrediction_3_3_1

#RMSE
rmse_3_3_1 <- rmse(splitRates_3_test$output, exchangePrediction_3_3_1)
rmse_3_3_1

#MAE
mae_3_3_1 <- mae(splitRates_3_test$output, exchangePrediction_3_3_1)
mae_3_3_1

#MAPE
mape_3_3_1 <- mape(splitRates_3_test$output, exchangePrediction_3_3_1)
mape_3_3_1

#- i=4

#Create the NN model
set.seed(100)
exchangeModel_4_3_1 <- neuralnet(output ~ input1 + input2 + input3 +
input4,
                                 data = splitRatesNormalised_4_train,
                                 hidden = c(3, 1))

#Plot the NN model
plot(exchangeModel_4_3_1)

#Obtain the NN model results
exchangeResults_4_3_1 <- compute(exchangeModel_4_3_1,
splitRatesNormalised_4_test[1:4])
```

```
#Corellation between prediction and actual results
cor(exchangeResults_4_3_1$net.result, splitRatesNormalised_4_test$output)

#Find Min and Max Value
testRates_4_min <- min(splitRates_4_train$output)
testRates_4_max <- max(splitRates_4_train$output)

exchangePrediction_4_3_1 <- unnormalise(exchangeResults_4_3_1$net.result,
testRates_4_min, testRates_4_max)
exchangePrediction_4_3_1

#RMSE
rmse_4_3_1 <- rmse(splitRates_4_test$output, exchangePrediction_4_3_1)
rmse_4_3_1

#MAE
mae_4_3_1 <- mae(splitRates_4_test$output, exchangePrediction_4_3_1)
mae_4_3_1

#MAPE
mape_4_3_1 <- mape(splitRates_4_test$output, exchangePrediction_4_3_1)
mape_4_3_1

par(mfrow=c(1,1))
plot(splitRates_4_test$output, exchangePrediction_4_3_1
,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN', pch=18,col='red', bty='n')

plot(splitRates_4_test$output, type = "l", col="black", main = "Real vs
Predicted")
lines(exchangePrediction_4_3_1, lty=2, lwd=2, col="red")

#- i=5

#Create the NN model
set.seed(100)
exchangeModel_5_3_1 <- neuralnet(output ~ input1 + input2 + input3 +
input4 + input5,
                                 data = splitRatesNormalised_5_train,
                                 hidden = c(3, 1))

#Plot the NN model
plot(exchangeModel_5_3_1)

#Obtain the NN model results
exchangeResults_5_3_1 <- compute(exchangeModel_5_3_1,
splitRatesNormalised_5_test[1:5])

#Corellation between prediction and actual results
cor(exchangeResults_5_3_1$net.result, splitRatesNormalised_5_test$output)

#Find Min and Max Value
testRates_5_min <- min(splitRates_5_train$output)
testRates_5_max <- max(splitRates_5_train$output)

exchangePrediction_5_3_1 <- unnormalise(exchangeResults_5_3_1$net.result,
testRates_5_min, testRates_5_max)
exchangePrediction_5_3_1
```

```
#RMSE
rmse_5_3_1 <- rmse(splitRates_5_test$output, exchangePrediction_5_3_1)
rmse_5_3_1

#MAE
mae_5_3_1 <- mae(splitRates_5_test$output, exchangePrediction_5_3_1)
mae_5_3_1

#MAPE
mape_5_3_1 <- mape(splitRates_5_test$output, exchangePrediction_5_3_1)
mape_5_3_1

#--- hn(l1) = 5 | hn(l2) = 1

#- i=2

#Create the NN model
set.seed(100)
exchangeModel_2_5_1 <- neuralnet(output ~ input1 + input2,
                                 data = splitRatesNormalised_2_train,
                                 hidden = c(5, 1))

#Plot the NN model
plot(exchangeModel_2_5_1)

#Obtain the NN model results
exchangeResults_2_5_1 <- compute(exchangeModel_2_5_1,
splitRatesNormalised_2_test[1:2])

#Corellation between prediction and actual results
cor(exchangeResults_2_5_1$net.result, splitRatesNormalised_2_test$output)

#Find Min and Max Value
testRates_2_min <- min(splitRates_2_train$output)
testRates_2_max <- max(splitRates_2_train$output)

exchangePrediction_2_5_1 <- unnormalise(exchangeResults_2_5_1$net.result,
testRates_2_min, testRates_2_max)
exchangePrediction_2_5_1

#RMSE
rmse_2_5_1 <- rmse(splitRates_2_test$output, exchangePrediction_2_5_1)
rmse_2_5_1

#MAE
mae_2_5_1 <- mae(splitRates_2_test$output, exchangePrediction_2_5_1)
mae_2_5_1

#MAPE
mape_2_5_1 <- mape(splitRates_2_test$output, exchangePrediction_2_5_1)
mape_2_5_1

#- i=3

#Create the NN model
set.seed(100)
exchangeModel_3_5_1 <- neuralnet(output ~ input1 + input2 + input3,
                                 data = splitRatesNormalised_3_train,
                                 hidden = c(5, 1))
```

```
#Plot the NN model
plot(exchangeModel_3_5_1)

#Obtain the NN model results
exchangeResults_3_5_1 <- compute(exchangeModel_3_5_1,
splitRatesNormalised_3_test[1:3])

#Corellation between prediction and actual results
cor(exchangeResults_3_5_1$net.result, splitRatesNormalised_3_test$output)

#Find Min and Max Value
testRates_3_min <- min(splitRates_3_train$output)
testRates_3_max <- max(splitRates_3_train$output)

exchangePrediction_3_5_1 <- unnormalise(exchangeResults_3_5_1$net.result,
testRates_3_min, testRates_3_max)
exchangePrediction_3_5_1

#RMSE
rmse_3_5_1 <- rmse(splitRates_3_test$output, exchangePrediction_3_5_1)
rmse_3_5_1

#MAE
mae_3_5_1 <- mae(splitRates_3_test$output, exchangePrediction_3_5_1)
mae_3_5_1

#MAPE
mape_3_5_1 <- mape(splitRates_3_test$output, exchangePrediction_3_5_1)
mape_3_5_1

#- i=4

#Create the NN model
set.seed(100)
exchangeModel_4_5_1 <- neuralnet(output ~ input1 + input2 + input3 +
input4,
                                 data = splitRatesNormalised_4_train,
                                 hidden = c(5, 1))

#Plot the NN model
plot(exchangeModel_4_5_1)

#Obtain the NN model results
exchangeResults_4_5_1 <- compute(exchangeModel_4_5_1,
splitRatesNormalised_4_test[1:4])

#Corellation between prediction and actual results
cor(exchangeResults_4_5_1$net.result, splitRatesNormalised_4_test$output)

#Find Min and Max Value
testRates_4_min <- min(splitRates_4_train$output)
testRates_4_max <- max(splitRates_4_train$output)

exchangePrediction_4_5_1 <- unnormalise(exchangeResults_4_5_1$net.result,
testRates_4_min, testRates_4_max)
exchangePrediction_4_5_1

#RMSE
rmse_4_5_1 <- rmse(splitRates_4_test$output, exchangePrediction_4_5_1)
rmse_4_5_1
```

```
#MAE
mae_4_5_1 <- mae(splitRates_4_test$output, exchangePrediction_4_5_1)
mae_4_5_1

#MAPE
mape_4_5_1 <- mape(splitRates_4_test$output, exchangePrediction_4_5_1)
mape_4_5_1

#- i=5

#Create the NN model
set.seed(100)
exchangeModel_5_5_1 <- neuralnet(output ~ input1 + input2 + input3 +
input4 + input5,
                                  data = splitRatesNormalised_5_train,
                                  hidden = c(5, 1))

#Plot the NN model
plot(exchangeModel_5_5_1)

#Obtain the NN model results
exchangeResults_5_5_1 <- compute(exchangeModel_5_5_1,
splitRatesNormalised_5_test[1:5])

#Corellation between prediction and actual results
cor(exchangeResults_5_5_1$net.result, splitRatesNormalised_5_test$output)

#Find Min and Max Value
testRates_5_min <- min(splitRates_5_train$output)
testRates_5_max <- max(splitRates_5_train$output)

exchangePrediction_5_5_1 <- unnormalise(exchangeResults_5_5_1$net.result,
testRates_5_min, testRates_5_max)
exchangePrediction_5_5_1

#RMSE
rmse_5_5_1 <- rmse(splitRates_5_test$output, exchangePrediction_5_5_1)
rmse_5_5_1

#MAE
mae_5_5_1 <- mae(splitRates_5_test$output, exchangePrediction_5_5_1)
mae_5_5_1

#MAPE
mape_5_5_1 <- mape(splitRates_5_test$output, exchangePrediction_5_5_1)
mape_5_5_1

#--- hn(l1) = 3 | hn(l2) = 3

#- i=2

#Create the NN model
set.seed(100)
exchangeModel_2_3_3 <- neuralnet(output ~ input1 + input2,
                                  data = splitRatesNormalised_2_train,
                                  hidden = c(3, 3))

#Plot the NN model
plot(exchangeModel_2_3_3)
```

```
#Obtain the NN model results
exchangeResults_2_3_3 <- compute(exchangeModel_2_3_3,
splitRatesNormalised_2_test[1:2])

#Corellation between prediction and actual results
cor(exchangeResults_2_3_3$net.result, splitRatesNormalised_2_test$output)

#Find Min and Max Value
testRates_2_min <- min(splitRates_2_train$output)
testRates_2_max <- max(splitRates_2_train$output)

exchangePrediction_2_3_3 <- unnormalise(exchangeResults_2_3_3$net.result,
testRates_2_min, testRates_2_max)
exchangePrediction_2_3_3

#RMSE
rmse_2_3_3 <- rmse(splitRates_2_test$output, exchangePrediction_2_3_3)
rmse_2_3_3

#MAE
mae_2_3_3 <- mae(splitRates_2_test$output, exchangePrediction_2_3_3)
mae_2_3_3

#MAPE
mape_2_3_3 <- mape(splitRates_2_test$output, exchangePrediction_2_3_3)
mape_2_3_3

#- i=3

#Create the NN model
set.seed(100)
exchangeModel_3_3_3 <- neuralnet(output ~ input1 + input2 + input3,
                                 data = splitRatesNormalised_3_train,
                                 hidden = c(3, 3))

#Plot the NN model
plot(exchangeModel_3_3_3)

#Obtain the NN model results
exchangeResults_3_3_3 <- compute(exchangeModel_3_3_3,
splitRatesNormalised_3_test[1:3])

#Corellation between prediction and actual results
cor(exchangeResults_3_3_3$net.result, splitRatesNormalised_3_test$output)

#Find Min and Max Value
testRates_3_min <- min(splitRates_3_train$output)
testRates_3_max <- max(splitRates_3_train$output)

exchangePrediction_3_3_3 <- unnormalise(exchangeResults_3_3_3$net.result,
testRates_3_min, testRates_3_max)
exchangePrediction_3_3_3

#RMSE
rmse_3_3_3 <- rmse(splitRates_3_test$output, exchangePrediction_3_3_3)
rmse_3_3_3

#MAE
mae_3_3_3 <- mae(splitRates_3_test$output, exchangePrediction_3_3_3)
```

```
mae_3_3_3

#MAPE
mape_3_3_3 <- mape(splitRates_3_test$output, exchangePrediction_3_3_3)
mape_3_3_3

#- i=4

#Create the NN model
set.seed(100)
exchangeModel_4_3_3 <- neuralnet(output ~ input1 + input2 + input3 +
input4,
                                  data = splitRatesNormalised_4_train,
                                  hidden = c(3, 3))

#Plot the NN model
plot(exchangeModel_4_3_3)

#Obtain the NN model results
exchangeResults_4_3_3 <- compute(exchangeModel_4_3_3,
splitRatesNormalised_4_test[1:4])

#Corellation between prediction and actual results
cor(exchangeResults_4_3_3$net.result, splitRatesNormalised_4_test$output)

#Find Min and Max Value
testRates_4_min <- min(splitRates_4_train$output)
testRates_4_max <- max(splitRates_4_train$output)

exchangePrediction_4_3_3 <- unnormalise(exchangeResults_4_3_3$net.result,
testRates_4_min, testRates_4_max)
exchangePrediction_4_3_3

#RMSE
rmse_4_3_3 <- rmse(splitRates_4_test$output, exchangePrediction_4_3_3)
rmse_4_3_3

#MAE
mae_4_3_3 <- mae(splitRates_4_test$output, exchangePrediction_4_3_3)
mae_4_3_3

#MAPE
mape_4_3_3 <- mape(splitRates_4_test$output, exchangePrediction_4_3_3)
mape_4_3_3

#- i=5

#Create the NN model
set.seed(100)
exchangeModel_5_3_3 <- neuralnet(output ~ input1 + input2 + input3 +
input4 + input5,
                                  data = splitRatesNormalised_5_train,
                                  hidden = c(3, 3))

#Plot the NN model
plot(exchangeModel_5_3_3)

#Obtain the NN model results
exchangeResults_5_3_3 <- compute(exchangeModel_5_3_3,
splitRatesNormalised_5_test[1:5])
```

```
#Corellation between prediction and actual results
cor(exchangeResults_5_3_3$net.result, splitRatesNormalised_5_test$output)

#Find Min and Max Value
testRates_5_min <- min(splitRates_5_train$output)
testRates_5_max <- max(splitRates_5_train$output)

exchangePrediction_5_3_3 <- unnormalise(exchangeResults_5_3_3$net.result,
testRates_5_min, testRates_5_max)
exchangePrediction_5_3_3

#RMSE
rmse_5_3_3 <- rmse(splitRates_5_test$output, exchangePrediction_5_3_3)
rmse_5_3_3

#MAE
mae_5_3_3 <- mae(splitRates_5_test$output, exchangePrediction_5_3_3)
mae_5_3_3

#MAPE
mape_5_3_3 <- mape(splitRates_5_test$output, exchangePrediction_5_3_3)
mape_5_3_3


#--- hn(l1) = 5 | hn(l2) = 3

#- i=2

#Create the NN model
set.seed(100)
exchangeModel_2_5_3 <- neuralnet(output ~ input1 + input2,
                                 data = splitRatesNormalised_2_train,
                                 hidden = c(5, 3))

#Plot the NN model
plot(exchangeModel_2_5_3)

#Obtain the NN model results
exchangeResults_2_5_3 <- compute(exchangeModel_2_5_3,
splitRatesNormalised_2_test[1:2])

#Corellation between prediction and actual results
cor(exchangeResults_2_5_3$net.result, splitRatesNormalised_2_test$output)

#Find Min and Max Value
testRates_2_min <- min(splitRates_2_train$output)
testRates_2_max <- max(splitRates_2_train$output)

exchangePrediction_2_5_3 <- unnormalise(exchangeResults_2_5_3$net.result,
testRates_2_min, testRates_2_max)
exchangePrediction_2_5_3

#RMSE
rmse_2_5_3 <- rmse(splitRates_2_test$output, exchangePrediction_2_5_3)
rmse_2_5_3

#MAE
mae_2_5_3 <- mae(splitRates_2_test$output, exchangePrediction_2_5_3)
mae_2_5_3
```

```
#MAPE
mape_2_5_3 <- mape(splitRates_2_test$output, exchangePrediction_2_5_3)
mape_2_5_3

#- i=3

#Create the NN model
set.seed(100)
exchangeModel_3_5_3 <- neuralnet(output ~ input1 + input2 + input3,
                                data = splitRatesNormalised_3_train,
                                hidden = c(5, 3))

#Plot the NN model
plot(exchangeModel_3_5_3)

#Obtain the NN model results
exchangeResults_3_5_3 <- compute(exchangeModel_3_5_3,
splitRatesNormalised_3_test[1:3])

#Corellation between prediction and actual results
cor(exchangeResults_3_5_3$net.result, splitRatesNormalised_3_test$output)

#Find Min and Max Value
testRates_3_min <- min(splitRates_3_train$output)
testRates_3_max <- max(splitRates_3_train$output)

exchangePrediction_3_5_3 <- unnormalise(exchangeResults_3_5_3$net.result,
testRates_3_min, testRates_3_max)
exchangePrediction_3_5_3

#RMSE
rmse_3_5_3 <- rmse(splitRates_3_test$output, exchangePrediction_3_5_3)
rmse_3_5_3

#MAE
mae_3_5_3 <- mae(splitRates_3_test$output, exchangePrediction_3_5_3)
mae_3_5_3

#MAPE
mape_3_5_3 <- mape(splitRates_3_test$output, exchangePrediction_3_5_3)
mape_3_5_3

#- i=4

#Create the NN model
set.seed(100)
exchangeModel_4_5_3 <- neuralnet(output ~ input1 + input2 + input3 +
input4,
                                data = splitRatesNormalised_4_train,
                                hidden = c(5, 3))

#Plot the NN model
plot(exchangeModel_4_5_3)

#Obtain the NN model results
exchangeResults_4_5_3 <- compute(exchangeModel_4_5_3,
splitRatesNormalised_4_test[1:4])

#Corellation between prediction and actual results
```

```
cor(exchangeResults_4_5_3$net.result, splitRatesNormalised_4_test$output)

#Find Min and Max Value
testRates_4_min <- min(splitRates_4_train$output)
testRates_4_max <- max(splitRates_4_train$output)

exchangePrediction_4_5_3 <- unnormalise(exchangeResults_4_5_3$net.result,
testRates_4_min, testRates_4_max)
exchangePrediction_4_5_3

#RMSE
rmse_4_5_3 <- rmse(splitRates_4_test$output, exchangePrediction_4_5_3)
rmse_4_5_3

#MAE
mae_4_5_3 <- mae(splitRates_4_test$output, exchangePrediction_4_5_3)
mae_4_5_3

#MAPE
mape_4_5_3 <- mape(splitRates_4_test$output, exchangePrediction_4_5_3)
mape_4_5_3

par(mfrow=c(1,1))
plot(splitRates_4_test$output, exchangePrediction_4_5_3
,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN', pch=18,col='red', bty='n')

plot(splitRates_4_test$output, type = "l", col="black", main = "Real vs
Predicted")
lines(exchangePrediction_4_5_3, lty=2, lwd=2, col="red")

#- i=5

#Create the NN model
set.seed(100)
exchangeModel_5_5_3 <- neuralnet(output ~ input1 + input2 + input3 +
input4 + input5,
                                 data = splitRatesNormalised_5_train,
                                 hidden = c(5, 3))

#Plot the NN model
plot(exchangeModel_5_5_3)

#Obtain the NN model results
exchangeResults_5_5_3 <- compute(exchangeModel_5_5_3,
splitRatesNormalised_5_test[1:5])

#Corellation between prediction and actual results
cor(exchangeResults_5_5_3$net.result, splitRatesNormalised_5_test$output)

#Find Min and Max Value
testRates_5_min <- min(splitRates_5_train$output)
testRates_5_max <- max(splitRates_5_train$output)

exchangePrediction_5_5_3 <- unnormalise(exchangeResults_5_5_3$net.result,
testRates_5_min, testRates_5_max)
exchangePrediction_5_5_3

#RMSE
```

```
rmse_5_5_3 <- rmse(splitRates_5_test$output, exchangePrediction_5_5_3)
rmse_5_5_3

#MAE
mae_5_5_3 <- mae(splitRates_5_test$output, exchangePrediction_5_5_3)
mae_5_5_3

#MAPE
mape_5_5_3 <- mape(splitRates_5_test$output, exchangePrediction_5_5_3)
mape_5_5_3

#--- hn(l1) = 5 | hn(l2) = 5

#- i=2

#Create the NN model
set.seed(100)
exchangeModel_2_5_5 <- neuralnet(output ~ input1 + input2,
                                  data = splitRatesNormalised_2_train,
                                  hidden = c(5, 5))

#Plot the NN model
plot(exchangeModel_2_5_5)

#Obtain the NN model results
exchangeResults_2_5_5 <- compute(exchangeModel_2_5_5,
splitRatesNormalised_2_test[1:2])

#Corellation between prediction and actual results
cor(exchangeResults_2_5_5$net.result, splitRatesNormalised_2_test$output)

#Find Min and Max Value
testRates_2_min <- min(splitRates_2_train$output)
testRates_2_max <- max(splitRates_2_train$output)

exchangePrediction_2_5_5 <- unnormalise(exchangeResults_2_5_5$net.result,
testRates_2_min, testRates_2_max)
exchangePrediction_2_5_5

#RMSE
rmse_2_5_5 <- rmse(splitRates_2_test$output, exchangePrediction_2_5_5)
rmse_2_5_5

#MAE
mae_2_5_5 <- mae(splitRates_2_test$output, exchangePrediction_2_5_5)
mae_2_5_5

#MAPE
mape_2_5_5 <- mape(splitRates_2_test$output, exchangePrediction_2_5_5)
mape_2_5_5

#- i=3

#Create the NN model
set.seed(100)
exchangeModel_3_5_5 <- neuralnet(output ~ input1 + input2 + input3,
                                  data = splitRatesNormalised_3_train,
                                  hidden = c(5, 5))

#Plot the NN model
```

```
plot(exchangeModel_3_5_5)

#Obtain the NN model results
exchangeResults_3_5_5 <- compute(exchangeModel_3_5_5,
splitRatesNormalised_3_test[1:3])

#Corellation between prediction and actual results
cor(exchangeResults_3_5_5$net.result, splitRatesNormalised_3_test$output)

#Find Min and Max Value
testRates_3_min <- min(splitRates_3_train$output)
testRates_3_max <- max(splitRates_3_train$output)

exchangePrediction_3_5_5 <- unnormalise(exchangeResults_3_5_5$net.result,
testRates_3_min, testRates_3_max)
exchangePrediction_3_5_5

#RMSE
rmse_3_5_5 <- rmse(splitRates_3_test$output, exchangePrediction_3_5_5)
rmse_3_5_5

#MAE
mae_3_5_5 <- mae(splitRates_3_test$output, exchangePrediction_3_5_5)
mae_3_5_5

#MAPE
mape_3_5_5 <- mape(splitRates_3_test$output, exchangePrediction_3_5_5)
mape_3_5_5

#- i=4

#Create the NN model
set.seed(100)
exchangeModel_4_5_5 <- neuralnet(output ~ input1 + input2 + input3 +
input4,
                                  data = splitRatesNormalised_4_train,
                                  hidden = c(5, 5))

#Plot the NN model
plot(exchangeModel_4_5_5)

#Obtain the NN model results
exchangeResults_4_5_5 <- compute(exchangeModel_4_5_5,
splitRatesNormalised_4_test[1:4])

#Corellation between prediction and actual results
cor(exchangeResults_4_5_5$net.result, splitRatesNormalised_4_test$output)

#Find Min and Max Value
testRates_4_min <- min(splitRates_4_train$output)
testRates_4_max <- max(splitRates_4_train$output)

exchangePrediction_4_5_5 <- unnormalise(exchangeResults_4_5_5$net.result,
testRates_4_min, testRates_4_max)
exchangePrediction_4_5_5

#RMSE
rmse_4_5_5 <- rmse(splitRates_4_test$output, exchangePrediction_4_5_5)
rmse_4_5_5
```

```
#MAE
mae_4_5_5 <- mae(splitRates_4_test$output, exchangePrediction_4_5_5)
mae_4_5_5

#MAPE
mape_4_5_5 <- mape(splitRates_4_test$output, exchangePrediction_4_5_5)
mape_4_5_5

#- i=5

#Create the NN model
set.seed(100)
exchangeModel_5_5_5 <- neuralnet(output ~ input1 + input2 + input3 +
input4 + input5,
                                 data = splitRatesNormalised_5_train,
                                 hidden = c(5, 5))

#Plot the NN model
plot(exchangeModel_5_5_5)

#Obtain the NN model results
exchangeResults_5_5_5 <- compute(exchangeModel_5_5_5,
splitRatesNormalised_5_test[1:5])

#Corellation between prediction and actual results
cor(exchangeResults_5_5_5$net.result, splitRatesNormalised_5_test$output)

#Find Min and Max Value
testRates_5_min <- min(splitRates_5_train$output)
testRates_5_max <- max(splitRates_5_train$output)

exchangePrediction_5_5_5 <- unnormalise(exchangeResults_5_5_5$net.result,
testRates_5_min, testRates_5_max)
exchangePrediction_5_5_5

#RMSE
rmse_5_5_5 <- rmse(splitRates_5_test$output, exchangePrediction_5_5_5)
rmse_5_5_5

#MAE
mae_5_5_5 <- mae(splitRates_5_test$output, exchangePrediction_5_5_5)
mae_5_5_5

#MAPE
mape_5_5_5 <- mape(splitRates_5_test$output, exchangePrediction_5_5_5)
mape_5_5_5

#-------------- OG -------------------------------------------------------
------

#The below splits the data into input matrix
splitRates <- as.data.frame(splitExchangeRates(rates, 5))

#The below normalises the data
splitRatesNormalised <- as.data.frame(lapply(splitRates, normalise))

#Below we split the data into training and test data at about 80% and 20%
respectively
splitRates_train <- splitRates[1:395,] #~80%
splitRates_test <- splitRates[396:494,] #~20%
```

```
splitRatesNormalised_train <- splitRatesNormalised[1:395,] #~80%
splitRatesNormalised_test <- splitRatesNormalised[396:494,] #~20%

####### MAKE INTO FUNCTION???? #########

#Create the NN model
set.seed(100)
exchangeModel <- neuralnet(output ~ input1 + input2 + input3 + input4 +
input5,
                           data = splitRatesNormalised_train,
                           hidden = 5)
#Plot the NN model
plot(exchangeModel)

#Obtain the NN model results
exchangeResults <- compute(exchangeModel, splitRatesNormalised_test[1:5])

#Corellation between prediction and actual results
cor(exchangeResults$net.result, splitRatesNormalised_test$output)

#Find actual predictions -------------

#Find non-normalised results -------------
head(exchangeResults$net.result)

#Find Min and Max Value
testRates_min <- min(splitRates_train$output)
testRates_max <- max(splitRates_train$output)

head(splitRates_train$output)

#Unnormalise
exchangePrediction <- unnormalise(exchangeResults$net.result,
testRates_min, testRates_max)
exchangePrediction

#Performance index -----------------
error <- (splitRates_test$output - exchangePrediction)

#RMSE
predictionRMSE <- rmse(error)
predictionRMSE


#------ Plots ----------------------------------------------------------
------

#Plot Results
par(mfrow=c(1,1))
plot(splitRates_test$output, exchangePrediction ,col='red',main='Real vs
predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN', pch=18,col='red', bty='n')

plot(splitRates_test$output, type = "l", col="black", main = "Real vs
Predicted")
lines(exchangePrediction, lty=2, lwd=2, col="red")
```

REFERENECES

Domingos, P. (2012). A Few Useful Things to Know About Machine Learning: Tapping into the “folk knowledge” needed to advance machine learning applications. *Communications of the ACM,* 55 (10), 78-87. Available from: https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf [Accessed 26 April 2021].

Jolliffe, I. T. and Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A,* 374 (2065). Available from https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202 [Accessed 26 April 2021].

Brownlee, J. (2020). Introduction to Dimensionality Reduction for Machine Learning. *Machine Learning Mastery*. Available from https://machinelearningmastery.com/dimensionality-reduction-for-machine-learning/ [Accessed 26 April 2021].

EduPristine (2015). Beyond the k-Means – Prepping the Data. *EduPristine*. Available from https://www.edupristine.com/blog/k-means-algorithm [Accessed 28 April 2021].

ektamaini (no date). Z score for Outlier Detection – Python. *GeeksforGeeks*. Available from https://www.geeksforgeeks.org/z-score-for-outlier-detection-python/ [Accessed 28 April 2021].

Parrinello, C. M. et al. (2016). Iterative outlier removal: A method for identifying outliers in laboratory recalibration studies. *Clin Chem*, 62 (7), 966-972. Available from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4927349/ [Accessed 28 April 2021].

Soetewey, A. (2020) The complete guide to clustering analysis: k-means and hierarchical clustering by hand and in R. *StatsandR*. Available from https://statsandr.com/blog/clustering-analysis-k-means-and-hierarchical-clustering-by-hand-and-in-r/#optimal-number-of-clusters [Accessed 1 May 2021].

Datanovia (2020) Determining the Optimal Number of Clusters: 3 Must Know Methods. *Datanovia.* Available from https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/#:~:text=The%20optimal%20number%20of%20clusters%20can%20be%20defined%20as%20follow,the%20number%20of%20clusters%20k. [Accessed 1 May 2021].

Crone, S. F. and Kourentzes, N. (2007). Input variable selection for time series prediction with neural networks- an evaluation of visual, autocorrelation and spectral analysis for varying seasonality. *The 1st European Symposium on Time Series Prediction.* Helsinky, Finland. January 2007. Available from https://www.researchgate.net/publication/237019865_Input_variable_selection_for_time_series_prediction_with_neural_networks-_an_evaluation_of_visual_autocorrelation_and_spectral_analysis_for_varying_seasonality [Accessed 10 May 2021].

Brownlee, J. (2017). Autoregression Models for Time Series Forecasting With Python. *Machine Learning Mastery*. Available form https://machinelearningmastery.com/autoregression-models-time-series-forecasting-python/ [Accessed 10 May 2021].

Jordan, J. (2018). Normalizing your data (specifically, input and batch normalization). *JeremyJordan.* Available from https://www.jeremyjordan.me/batch-normalization/#:~:text=By%20normalizing%20all%20of%20our,parameters%20for%20each%20input%20node.&text=Moreover%2C%20if%20your%20inputs%20and,for%20your%20neural%20network%20(ie. [Accessed 12 May 2021].

Vandeput, N. (2019). Forecast KPIs: RMSE, MAE, MAPE & Bias. *Towards Data Science.* Available from https://towardsdatascience.com/forecast-kpi-rmse-mae-mape-bias-cdc5703d242d [Accessed 12/05/2021].