UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

INFORMATICS INSTITUTE OF TECHNOLOGY

BEng (Hons) Software Engineering

5DATA001C.2 Machine Learning and Data Mining

Coursework Report

Name: Praveen de Silva
UOW ID: W1985643
Student ID: 20221895

Table of Contents

TASK 1

Data Pre-Processing

Data preprocessing is a critical step before applying data mining techniques. It entails preparing the data by removing copies, errors, and inconsistencies. Preprocessing becomes essential to adjust data for effective analysis because of the exponential rise of data across multiple areas. It ensures that the data meets the unique needs of various mining techniques, enabling effective evaluation of large data sets. (Swart et al., 2016)

The code performs k-means clustering to a wine dataset after completing preprocessing processes. It begins by choosing appropriate characteristics and applying the Z-score approach to eliminate outliers. The data is then standardized to guarantee consistency between variables. The ideal number of clusters is then found using a variety of automated methods, including NbClust, the Elbow Method, the Gap Statistic, and the Silhouette Method. By ensuring the data is suitably ready for clustering analysis, this preprocessing improves the precision and dependability of the outcomes.

Outliers are removed in the provided code before data scaling to guarantee the stability and efficacy of the ensuing scaling procedure. The mean and standard deviation computations, which are employed in standardization (scaling), could be disproportionately impacted by outliers. Outliers are kept from greatly impacting the scaling transformation through removing them first, this improves the accuracy of the scaling process and makes it more representative of most of the data points. By using this sequential method, it is ensured that the scaled data closely matches the underlying distribution, producing more trustworthy clustering findings.

Outlier Removal

In the code provided uses the Z-score approach to eliminate outliers, setting a threshold value of 3. Using this procedure, the Z-score—a measure of the extent to which a data point deviates from the mean in standard deviations—is computed for each data point. Outliers are identified and eliminated when their Z-score exceeds the selected threshold, in this case, 3.

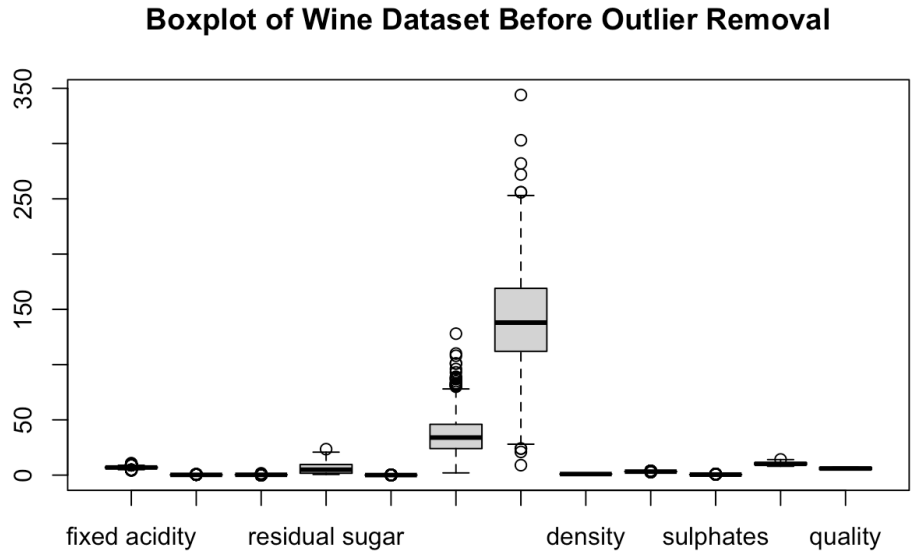
There are three main ways to remove outliers.

- Z score method
- IQR method
- boxplot outlier removal

While there are several options for removing outliers, such as boxplot and interquartile range (IQR), the Z-score strategy is recommended in this case due to its efficiency and simplicity. Implementing and understanding the Z-score method is simple because it offers a clear method for identifying outliers based on their divergence from the mean. In addition, the Z-score method provides a standardized methodology that is less susceptible to the data's distribution than boxplot or IQR methods since it uses an established threshold the amount (e.g., three standard deviations). This makes it a good option for this preprocessing pipeline's outlier removal, ensuring accuracy and dependability in the data preparation procedure. List the items in bullet points.

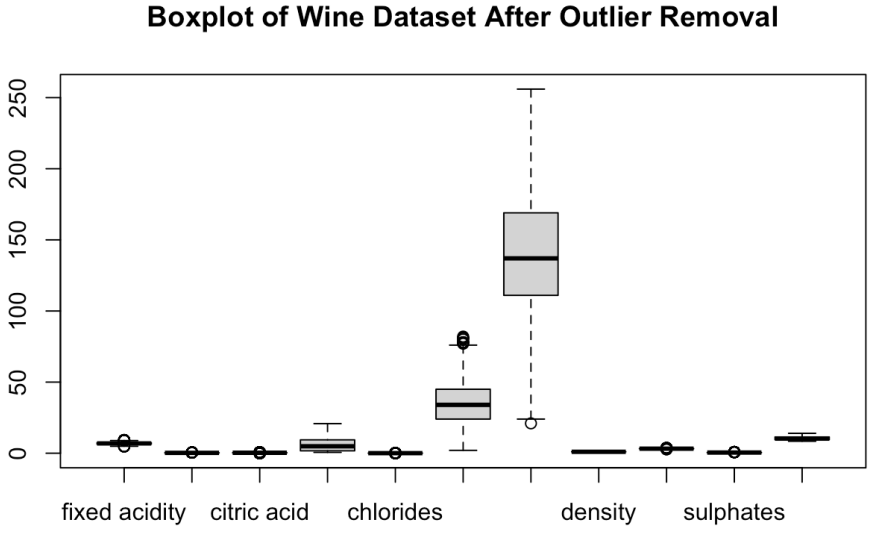
Boxplot of Wine Dataset Before Outlier Removal

The boxplot visualizes the wine dataset before removing outliers. Each box represents the interquartile range (IQR) of the wine data, with the middle line indicating the median. This plot provides an overview of the wine dataset's spread and skewness.



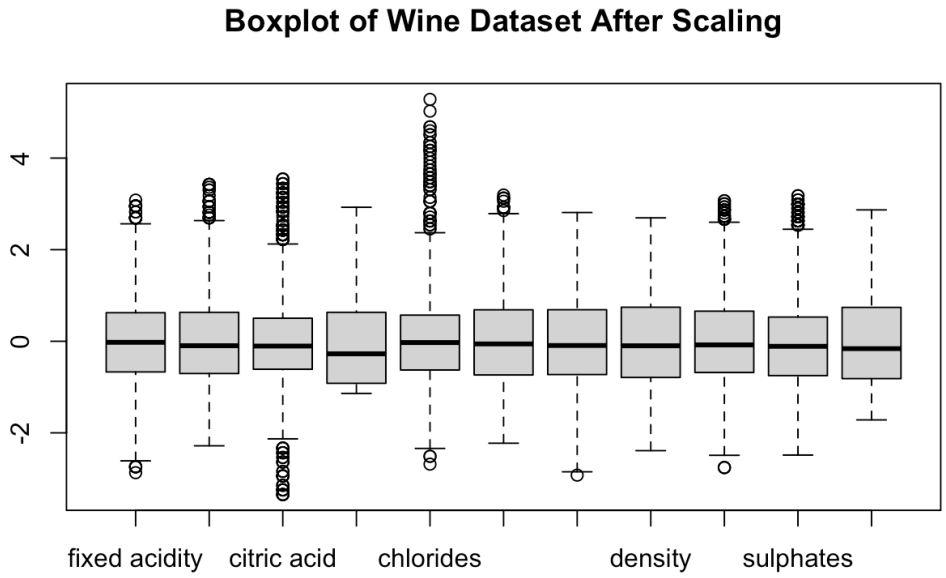
Boxplot of Wine Dataset After Outlier Removal

The boxplot displays the wine dataset after removing outliers. Outliers have been eliminated to enhance the quality of the wine data. Similar to the previous plot, each box represents the interquartile range (IQR) of the wine data, and the middle line shows the median. This cleaned wine dataset is better suited for analysis.



Boxplot of Wine Dataset After Scaling

The data was scaled using the R scale() function after the wine dataset's outliers had been removed using the z-score strategy. The distribution of the scaled wine dataset can be seen in the boxplot below. The central line in the plot indicates the median, while each box shows the distribution of the scaled data. By scaling the data, you Can make sure that every feature contributes the same amount to the analysis. An overview of the scaled wine dataset's distribution and spread is given in this graphic. (Cook, 2024)



To clarify, the code transforms the data to fit within a specific scale, typically 0-1 or -1 to 1. This scaling process is essential when using methods based on measures of how far apart data points are. The scaled data looked like below

	A	B	C	D	E	F	G	H	I	J	K
1	fixed acidity	latile acidi	citric acid	esidual suga	chlorides	sulfur diox	l sulfur diox	density	pH	sulphates	alcohol
2	0.882692	0.509962	-0.40866	0.853264	-0.28502	-0.93878	0.102701	0.860233	-1.08349	-1.29774	-0.89821
3	1.52968	2.817019	1.414834	-1.03863	-0.54198	-0.8033	-0.36101	-0.09842	0.054893	-0.65845	0.330269
4	1.52968	2.817019	1.414834	-1.03863	-0.54198	-0.8033	-0.36101	-0.09842	0.054893	-0.65845	0.330269
5	0.882692	0.509962	-0.40866	0.853264	-0.28502	-0.93878	0.102701	0.860233	-1.08349	-1.29774	-0.89821
6	1.141487	-0.46143	-0.50997	-0.30401	-0.11372	0.686891	0.615228	-0.4408	-1.61921	-0.8411	0.821662
7	-0.02309	-0.8257	0.705696	0.289722	-0.97023	1.093308	0.859288	-0.06418	0.121857	-2.02835	0.739763
8	2.953055	0.631386	0.908307	-0.99837	0.228882	-1.41293	1.103349	0.278193	-0.81564	-0.11049	-0.73441
9	0.106306	-1.43282	0.097864	-0.97825	-0.37067	-0.93878	0.883694	-0.47503	0.791497	1.168086	0.248371
10	-0.67008	-0.21858	2.022666	0.068331	-0.45633	1.635197	2.274838	0.483619	-1.15046	-0.65845	-1.3896
11	-0.92888	-0.8257	-0.71258	-0.91787	4.254478	0.34821	0.493198	-0.57775	1.595064	1.533393	0.412168
12	1.917873	-0.94713	1.009612	-0.83736	0.742788	-1.00651	-0.89795	-0.02994	-2.35581	-0.29314	-0.81631
13	1.788476	-0.70428	0.807001	-0.81723	0.571486	-0.73557	-0.80032	-0.02994	-2.35581	-0.38447	-0.81631
14	-0.28189	-1.06855	-0.30736	0.028078	0.400184	0.34821	2.68974	0.517857	0.590605	0.254817	-0.40682
15	-0.28189	-1.06855	-0.30736	0.028078	0.400184	0.34821	2.68974	0.517857	0.590605	0.254817	-0.40682
16	0.882692	3.302715	-1.0165	-0.85749	-1.22718	-1.48067	0.249138	-0.33808	-1.08349	-1.48039	-0.07922
17	-0.28189	-1.06855	-0.30736	0.028078	0.400184	0.34821	2.68974	0.517857	0.590605	0.254817	-0.40682
18	1.141487	-1.67567	-0.30736	-0.89774	-0.28502	0.551419	0.932506	0.483619	1.5281	-0.74978	-1.22581
19	0.882692	-0.94713	0.807001	-1.03863	-0.28502	0.212738	0.053889	-1.15979	-1.88706	-0.65845	0.739763
20	0.882692	-0.94713	0.807001	-1.03863	-0.28502	0.212738	0.053889	-1.15979	-1.88706	-0.65845	0.739763
21	0.882692	0.024265	-0.91519	-0.49521	1.256694	-1.07425	-0.62948	0.586332	-0.34689	-0.38447	-1.30771
22	2.694259	0.509962	1.516139	0.068331	-0.02807	-1.61614	-1.65453	-0.09842	-2.02099	-1.02376	0.903561
23	-1.31707	1.784914	-3.14391	-1.099	-0.45633	-0.87104	-1.89859	-0.30385	1.059353	-1.20641	-0.98011
24	-2.09345	-1.25069	-0.91519	-1.03863	0.657137	-0.66783	-0.87354	-0.54351	0.791497	-0.20182	-1.14391
25	0.623896	-0.21858	1.617445	-1.03863	-0.11372	0.00953	0.371168	-0.81741	-0.48082	0.528798	0.657864
26	-1.05827	-0.58285	1.617445	-0.95812	0.657137	-1.14198	-1.31285	-0.4408	0.657569	-0.29314	-0.73441
27	-1.05827	-0.58285	1.617445	-0.95812	0.657137	-1.14198	-1.31285	-0.4408	0.657569	-0.29314	-0.73441
28	0.623896	-0.21858	1.617445	-1.03863	-0.11372	0.00953	0.371168	-0.81741	-0.48082	0.528798	0.657864
29	-0.02309	-0.46143	-0.91519	1.597943	0.828439	-1.07425	-1.14201	1.408034	-0.21296	0.072164	-0.73441
30	-0.79948	-0.46143	1.617445	0.168963	0.571486	2.177086	1.689093	0.346669	0.188821	-1.38907	-0.81631
31	-1.05827	-0.21858	1.617445	0.269596	0.742788	2.177086	2.07959	0.517857	-0.27993	-0.20182	-1.30771
32	0.365101	0.509962	-0.91519	-0.97825	1.171043	-1.20972	-0.58067	-0.4408	-0.27993	-1.29774	0.002674
33	1.400283	-0.21858	1.617445	-1.0185	1.513647	-0.53236	-0.58067	-0.09842	-0.81564	-1.38907	-0.89821
34	0.365101	1.602778	1.617445	-1.03863	0.400184	-1.61614	-0.06814	-0.40656	-1.28439	-0.65845	-0.98011
35	1.788476	-0.70428	1.617445	2.725026	0.91409	1.025572	2.20162	2.435162	-1.41831	0.437471	-1.06201
36	-0.54068	-0.34001	-0.91519	-0.93799	0.228882	-1.34519	-1.97181	-0.4408	-0.07903	-0.93243	-0.57062
37	0.365101	1.602778	1.617445	-1.03863	0.400184	-1.61614	-0.06814	-0.40656	-1.28439	-0.65845	-0.98011
38	1.141487	-0.70428	1.617445	-0.98831	0.742788	-1.95482	-2.26468	-1.02284	-0.34689	-1.93703	0.739763
39	0.753294	-0.8257	1.617445	-0.99837	-1.05588	-1.81935	-1.06879	-0.78317	-0.94957	1.168086	0.494067
40	1.400283	0.388538	1.617445	0.631872	0.228882	0.822363	1.152161	0.791757	-0.41385	-0.11049	-0.40682
41	1.788476	-0.8257	1.617445	-0.91787	-0.28502	-0.05821	0.688446	-0.09842	-1.01653	-1.11509	-0.32492
42	-0.41129	0.388538	-0.91519	-1.0185	-0.79893	-1.20972	-0.48304	-0.26961	-0.48082	-1.20641	-1.06201
43	1.270885	-0.8257	1.617445	-0.93799	0.828439	-1.34519	0.078295	-0.37232	-0.27993	-0.20182	0.002674
44	1.400283	-1.06855	1.617445	-0.89774	1.513647	-1.68387	0.102701	0.03853	0.188821	-0.11049	-0.40682
45	1.788476	-0.8257	1.617445	-0.91787	-0.28502	-0.05821	0.688446	-0.09842	-1.01653	-1.11509	-0.32492

TASK 1 Sub task 1

Determining Optimal Number of Clusters

A large class of methods for finding observational subgroups within a data collection is called clustering. We want observations within the same group to be similar and observations between groups to be dissimilar when we cluster observations. This method is considered unsupervised since it doesn't depend on a response variable for training; rather, it looks for correlations among the n observations. By grouping observations shared, we may determine which ones have similarities and possibly group them. The simplest and most popular clustering technique for partitioning a dataset into a set of k groups is called K-means clustering. (UC Business Analytics R Programming Guide. (n.d.). K-means clustering. Retrieved from https://uc-r.github.io/kmeans_clustering)

NBclust Analysis

Introduction to NBclust

To be able to determine the perfect amount of clusters, the wine dataset analysis required the use of the Nbclust in R. The researchers were able to assess multiple clustering schemes by changing the number of clusters, distance metrics, and clustering algorithms thanks to NBclust's thirty distinct indices. This tool helps the researchers choose the best clustering strategy by offering crucial insights into the structure of the wine dataset. They could decide how many clusters to use by using NBclust, which would ensure that their additional analyses were founded on a strong clustering technique. (Charrad M, 2014)

So the nbclust method determine 2 as the best number of clusters. The output looks like below.

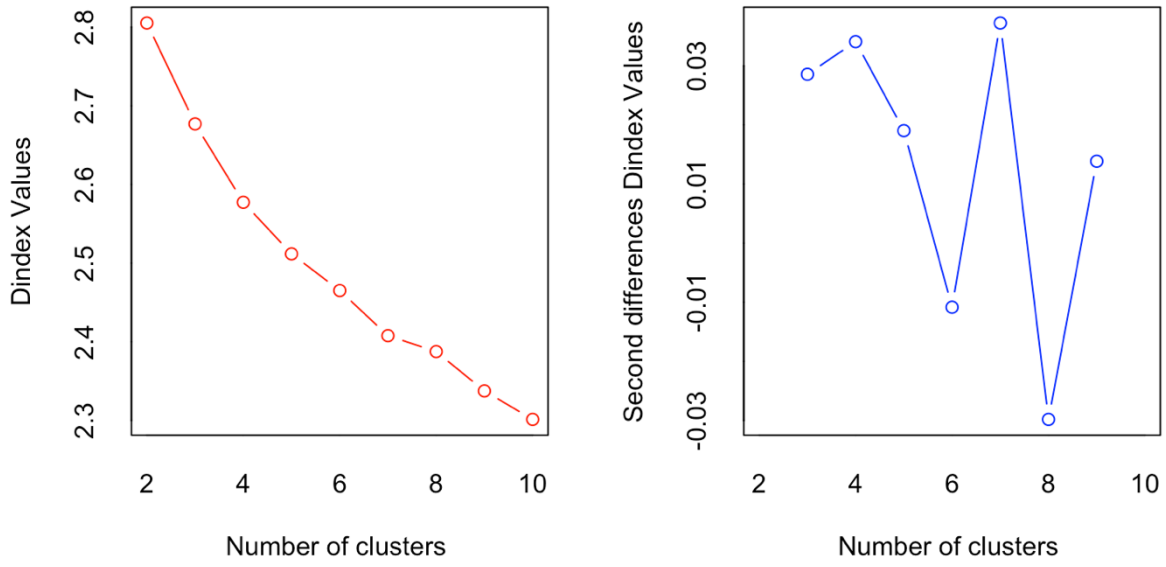
```
> # NbClust for original dataset
> nb <- NbClust(scaled_data, min.nc = 2, max.nc = 10, method = "kmeans")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
            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

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

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

                   ***** Conclusion *****

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


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

Euclidean distance is the straight-line distance between two points in Euclidean space. It is the most commonly used distance metric. In this part, the code utilizes the NbClust function to determine the optimal number of clusters using the k-means algorithm with Euclidean distance. The function is applied to the scaled data (scaled_data) with the method = "kmeans" argument and the Euclidean distance measure (distance = "euclidean"). The optimal number of clusters is determined, and the result is printed to the console. And it also gets the answer as 2

```
> best_k_nb_euclidean <- nb_euclidean$Best.nc[1]
> best_k_nb_euclidean <- nb_euclidean$Best.nc[1]
> nb_clusters_euclidean <- cbind(2:10, nb_euclidean$Best.n[2:10])
> cat("Best k using NbClust Method with Euclidean Distance:", best_k_nb_euclidean, "\n")
Best k using NbClust Method with Euclidean Distance: 2
```

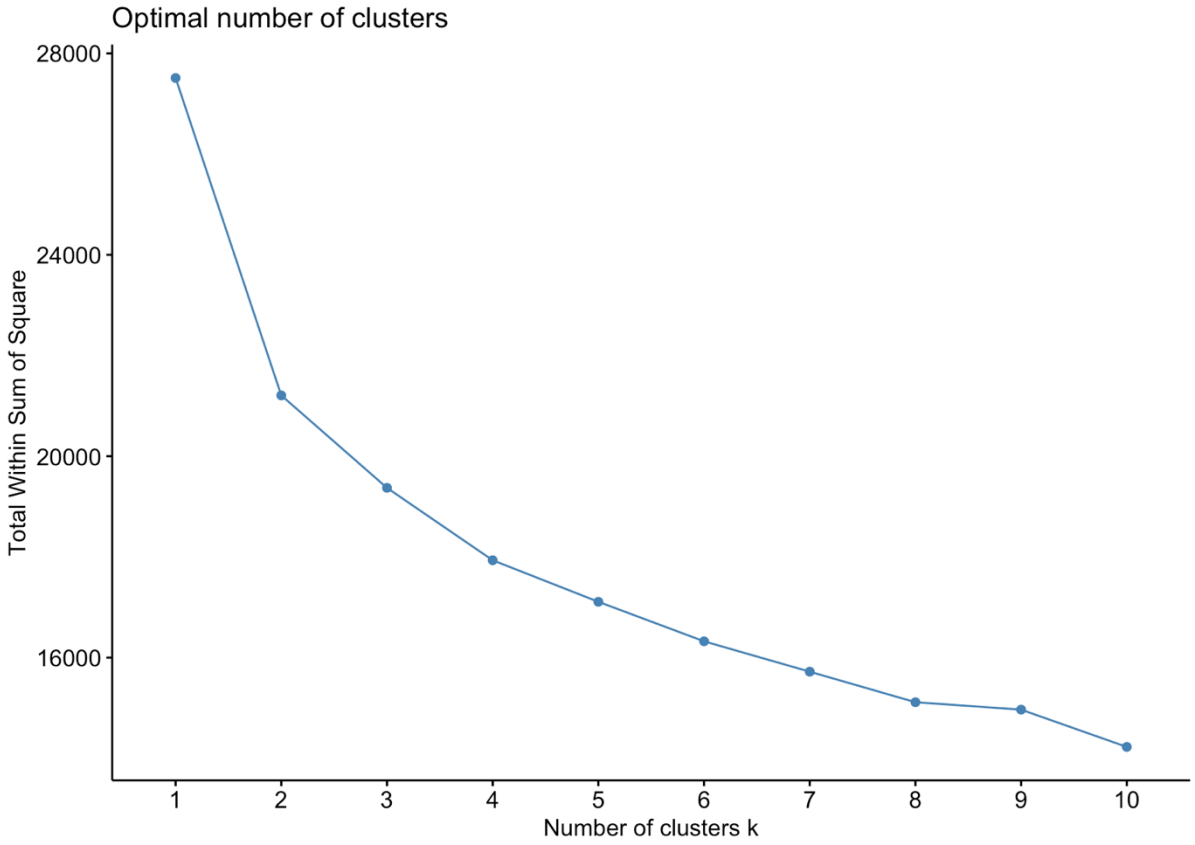
Manhattan distance, on the other hand, is the sum of the absolute differences between the coordinates of two points. It is also known as city block distance or taxicab distance. In the next section, the code utilizes the NbClust function to determine the optimal number of clusters using the k-means algorithm with Manhattan distance. The function is applied to the scaled data (scaled_data) with the method = "kmeans" argument and the Manhattan distance measure (distance = "manhattan"). The optimal number of clusters is determined, and the result is printed to the console. It also gives 2 as the answer.

```
*****************************************************************
> best_k_nb_manhattan <- nb_manhattan$Best.nc[1]
> nb_clusters_manhattan <- cbind(2:10, nb_manhattan$Best.n[2:10])
> cat("Best k using NbClust Method with Manhattan Distance:", best_k_nb_manhattan, "\n")
Best k using NbClust Method with Manhattan Distance: 2
```



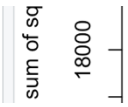
Elbow Method

Using the k-means method, the code in the Elbow Method calculates the the within-cluster sum of squares (WCSS) for different cluster sizes (K). Next, by locating the "elbow" point on the plot—the point when the rate of decrease in WCSS swiftly levels off—it calculates the ideal number of clusters. (analyicsvidhya, 2012)



The elbow method also gets the cluster number as 2. This is determined using the graph and as well as the code.

```
> min_wss_index <- which.min(wss)
> best_k_elbow <- min_wss_index + 1  # Adjusting to match k value
> cat("Best k using the Elbow Method:", best_k_elbow, "\n")
Best k using the Elbow Method: 2
```



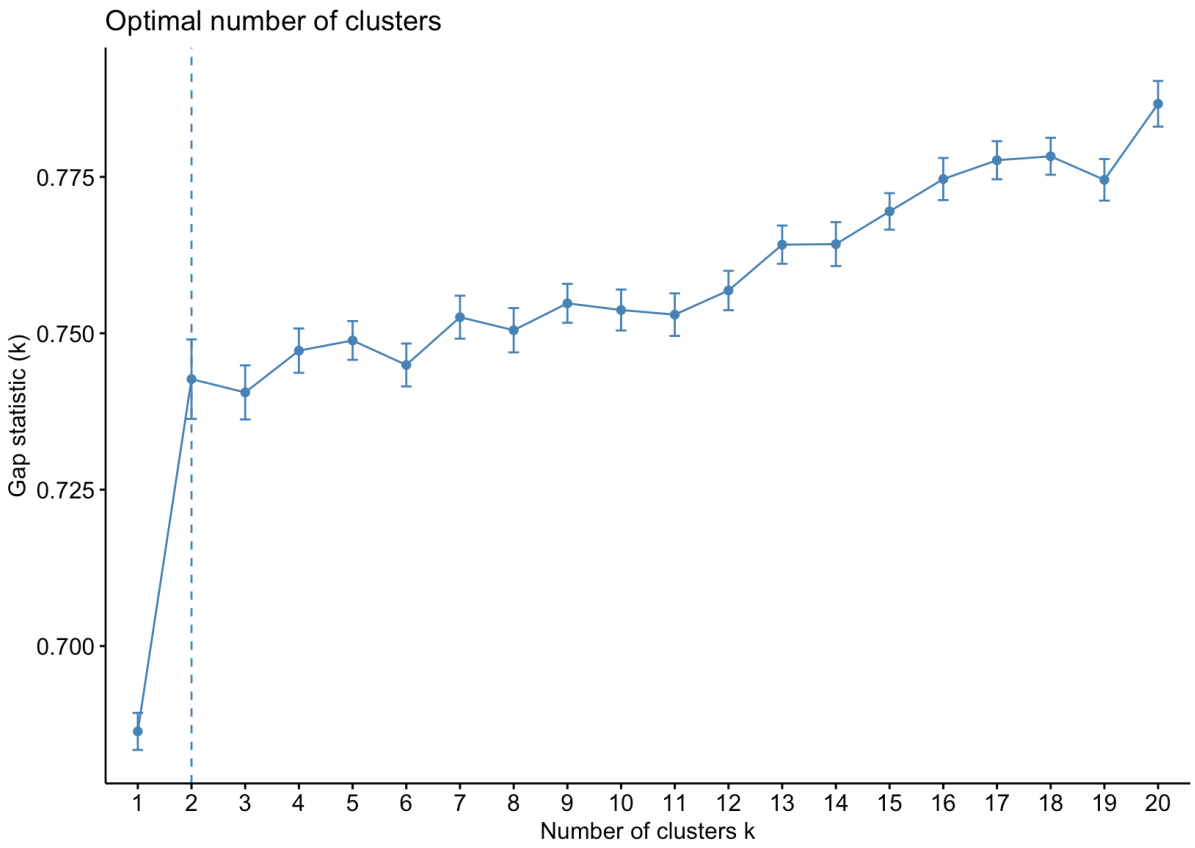
Popular methods for determining the perfect number of clusters in a dataset include NbClust and the Elbow Method. Although the Elbow Method is a simple visual method for locating the "elbow" point in the within-cluster sum of squares (WCSS) plot, NbClust offers a more methodical approach by computation several indices. When it is available, NbClust is the recommended option since it is seen as more reliable and objective. To make a better-informed decision, it is often useful to combine the two approaches and compare the outcomes.

Gap Statistic

The gap statistic is a method for determining the optimal number of clusters in a dataset. It compares the total intra cluster variation for different values of
K. k with their anticipated values under a null reference distribution for the data. Monte Carlo simulations are utilized to build the reference dataset throughout the sampling phase with one another. (UC Business Analytics R Programming Guide. (n.d.). K-means clustering. Retrieved from https://uc-r.github.io/kmeans_clustering)

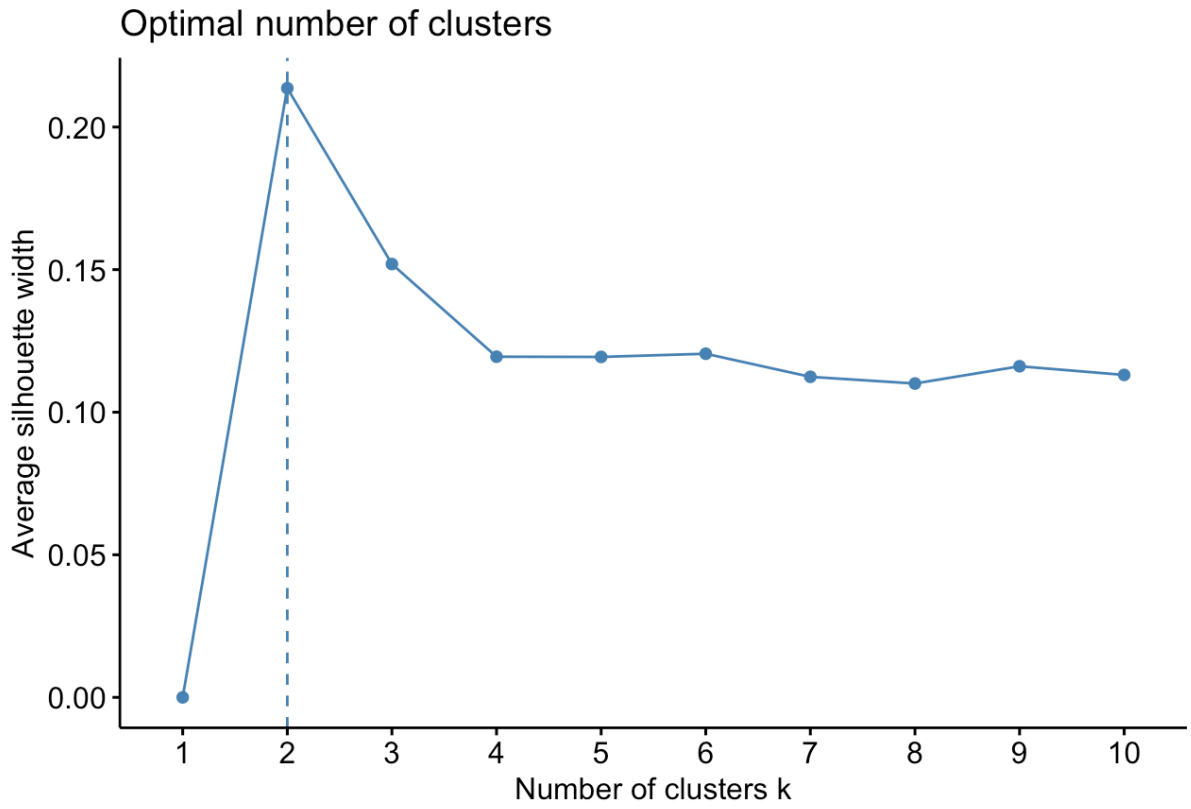
```
94
95  # Gap Statistic
96  gap_stat <- clusGap(scaled_data, FUN = kmeans, K.max = 20, B = 50, verbose = TRUE, iter.max = 1000)
97  best_k_gap <- maxSE(gap_stat$Tab[,"gap"], gap_stat$Tab[,"SE.sim"], method = "Tibs2001SEmax")
98  cat("Best k using Gap Statistic Method:", best_k_gap, "\n")
99
100 # Plot Gap Statistic
101 plot(gap_stat, main = "Gap Statistic Method to Find Optimal k")
102
```

The gap statistic is determined by the code above to determine the ideal number of clusters. The gap statistic is produced for cluster numbers between 1 and 20. The clusGap function uses the k-means clustering method (FUN = kmeans) to compute the gap statistics for Kmax=20 K max =20 clusters. Here, iter.max = 1000 specifies the maximum number of k-means algorithm iterations, while B=50 denotes the number of Monte Carlo simulations that were used to create the reference dataset. Finally, maxSE uses the maximum standard error to calculate the ideal number of clusters. The gap statistics are plotted to determine the ideal number of clusters. The ideal number of clusters in the above scenario is found to be 2.



Silhouette analysis.

The distance separating the generated clusters can be examined using silhouette analysis. The silhouette plot offers a visual way of evaluating factors such as the number of clusters by showing a measure of the proximity between each point in a cluster and the points in its neighboring clusters. The range of this metric is [-1, 1]. This also determines the value as 2.




```
> best_k_sil <- which.max(sil_width[2:10]) + 1
> for (i in 2:10) {
+   set.seed(123)
+   kmeans_fit <- kmeans(scaled_data, centers = i, nstart = 25, iter.max = 1000) # Increased iter.max
+   if (length(unique(kmeans_fit$cluster)) > 1) {
+     cluster_assignments <- kmeans_fit$cluster
+     sil_width[i] <- cluster.stats(dist(scaled_data), cluster_assignments)$avg.silwidth
+     sil_clusters <- rbind(sil_clusters, c(i, sil_width[i]))
+   }
+ }
> best_k_sil <- which.max(sil_width[2:10]) + 1
> cat("Best k using Silhouette Method:", best_k_sil, "\n")
Best k using Silhouette Method: 2
```

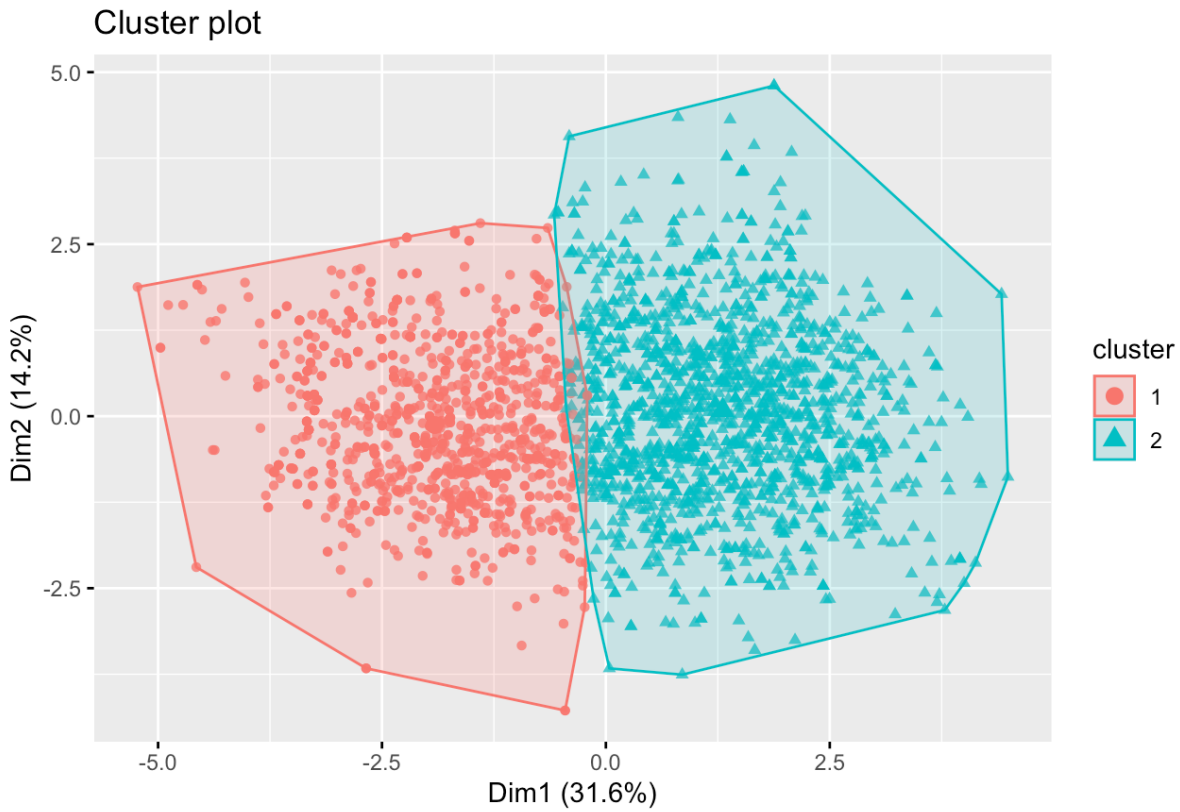
KMeans analysis

The code calculates the optimal number of clusters using the silhouette, elbow, gap statistic, and NbClust methods and selects the minimum value among them. In this case, the best number of clusters is 2. Then, it performs k-means clustering with 2 clusters using the scaled data.

```
# Choose the best number of clusters
best_k <- min(best_k_nb, best_k_elbow, best_k_sil, best_k_gap)

# Perform k-means clustering with the best k
set.seed(123)
kmeans_fit <- kmeans(scaled_data, centers = best_k, nstart = 25)

# K-means output
kmeans_fit
#plot
fviz_cluster(kmeans_fit, geom = "point", data = scaled_data, alpha = 0.8)
```

Then. When it plots the graph for this kmeans clustering.

K=2,



Cluster vector shows which cluster each observation belongs to. For example, the first observation belongs to Cluster 1, the second to Cluster 2, and so on.

```
Clustering vector:
   [1] 1 2 2 1 2 2 1 2 1 1 2 2 1 1 2 1 1 2 2 1 2 2 2 2 2 2 2 1 1 1 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2
  [50] 2 1 2 2 1 2 2 2 1 2 2 2 1 2 1 2 1 2 2 1 1 2 1 2 1 1 1 1 1 1 1 1 2 2 2 2 2 1 2 2 1 1 1 1 1 2 1 1
  [99] 2 2 1 2 2 1 2 1 2 2 1 2 1 2 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 2 1 1 2 1 1 2 2 1 1 2 1 1 1 1 2 2 2
 [148] 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 1 1 1 1 1 2
 [197] 2 1 1 2 1 2 2 2 2 2 1 1 2 2 2 2 2 1 1 1 1 1 2 1 2 1 1 2 2 2 1 1 1 1 2 1 2 1 1 1 1 1 2 2 1 1 2 1 1
 [246] 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 2 1 1 1 2 1 1 2 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1
 [295] 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 2 1 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [344] 2 2 2 2 1 2 1 1 1 2 2 1 1 1 1 1 2 1 1 2 1 1 1 2 1 1 1 2 1 2 1 1 2 1 2 1 2 2 2 1 2 1 1 1 2 1 2 1 1
 [393] 1 1 1 1 2 1 1 2 1 1 1 1 2 2 2 1 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 2 2 2 2 1 1 1 1
 [442] 1 1 1 1 1 1 1 2 2 2 1 2 1 1 1 2 2 2 1 2 2 2 1 1 2 2 2 2 1 2 1 1 2 1 1 1 1 1 1 1 2 1 2 2 1 2 1 2 2 2
 [491] 2 2 2 2 2 2 2 1 1 2 1 1 2 2 2 2 1 1 1 1 2 2 2 1 2 2 1 2 1 2 2 2 2 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 1 2 2
 [540] 2 2 2 2 2 2 2 2 2 1 2 2 1 2 1 2 1 1 1 1 2 2 1 2 2 2 2 2 1 1 1 2 2 1 1 1 1 2 1 1 1 2 2 1 1 2 2 1 1 2 1
 [589] 1 2 1 1 1 1 1 1 2 2 1 2 2 1 1 1 1 1 2 1 2 2 2 2 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 1 1 1 2 2 1 2 1 2 2 2 2 2
 [638] 2 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 2 2 2 2 1 2 2 2 2 1 1 2 1 2 1 1 2 1 1 1 2 1 2 1 1 2 1 1 1 1 1 1 2
 [687] 1 2 2 2 2 2 2 2 1 1 2 2 2 2 1 1 1 1 2 2 1 2 1 1 2 2 1 1 1 2 2 2 2 2 1 2 2 1 2 2 2 1 1 2 2 2 1 1 1 2
 [736] 2 2 1 2 1 1 2 1 2 2 2 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 2 2 1 1 2 1 2 1 1 2 1 1 1 1 1 1 1 2 1
 [785] 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 2 2 1 2 1 1 1 1 2 2 1 1 2 2 1 2 2 2 2 2 2 1 2
 [834] 1 2 1 2 2 2 2 2 1 2 2 1 2 2 2 1 1 2 2 1 1 2 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 2 2 2 2 2
 [883] 2 2 2 2 1 2 2 1 1 1 1 2 2 2 1 1 1 1 1 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 1 1 1 1 1 2 2
 [932] 1 2 2 2 2 2 2 1 1 1 2 1 1 2 2 2 2 2 1 2 1 2 2 2 1 2 1 2 2 1 1 2 1 1 1 2 2 1 2 1 2 2 2 2 2 2 2 1 1 2 2
 [981] 2 2 2 2 2 2 2 2 1 1 1 1 2 2 2 1 1 1 1 1 2
 [ reached getOption("max.print") -- omitted 1502 entries ]
```

Within cluster sum of squares by cluster provides each cluster's within-cluster sum of squares. The distance between each data point and the centroid of its designated cluster can be calculated by the sum of squares. This ratio (22.9%) represents the percentage of variance between the clusters relative to the total variance in the dataset.

```
Within cluster sum of squares by cluster:
[1]  8578.959 12628.540
 (between_SS / total_SS =  22.9 %)
```

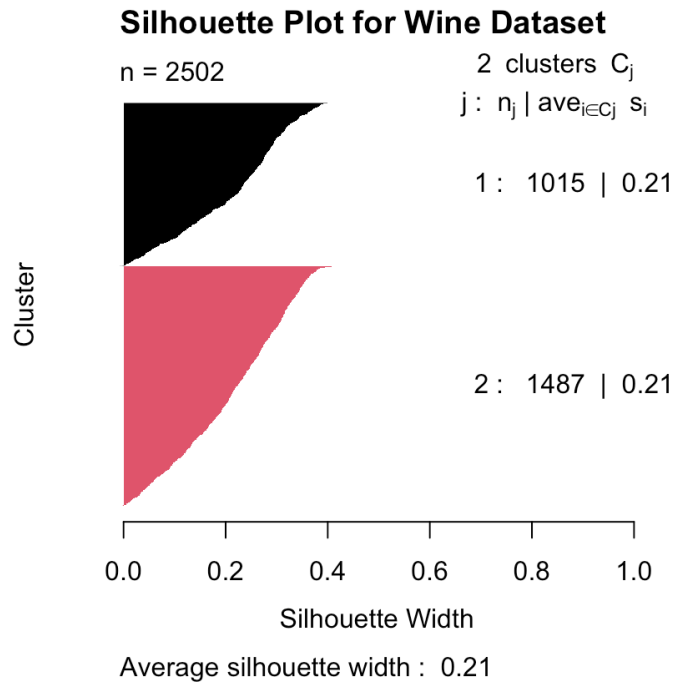
The dataset has been divided into two clusters, based to the kmeans_fit result. There are 1015 observations in Cluster 1 and 1487 observations in Cluster 2. The average values for each characteristic (such as citric acid, volatile acidity, fixed acidity, etc.) for every cluster are given in the cluster means table. For example, Cluster 1's mean fixed acidity is roughly 0.1875, while Cluster 2's is nearly -0.128.

```
K-means clustering with 2 clusters of sizes 1015, 1487

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide
1     0.1875139       0.06906669  0.13998285      0.8605806  0.5567184           0.5966984
2    -0.1279937      -0.04714371 -0.09554982     -0.5874171 -0.3800062          -0.4072958
  total sulfur dioxide    density         pH   sulphates    alcohol
1            0.7882704  0.9690305 -0.1632178  0.10212593 -0.7920798
2           -0.5380595 -0.6614432  0.1114096 -0.06970936  0.5406597
```

Silhouette Plot for Wine Dataset

When a cluster is depicted, all its points' silhouette values are shown in decreasing order. The silhouettes of every group appear in a silhouette plot in a random sequence. It can also color each cluster a different color and add blank spaces between them.

https://www.baeldung.com/cs/silhouette-values-clustering#:~:text=A%20silhouette%20plot%20is%20a%20graphical%20tool%20we%20use%20to,clusters%20appear%20in%20the%20dataset.



And the code for the above is as below. In it. To see the silhouette widths for every dataset observation, the code creates a silhouette plot. In comparison to other clusters, the silhouette width shows how similar an observation is to its own cluster. The quality of the clustering is then ascertained by calculating the average silhouette width score. Ultimately, the optimal cluster value is preserved. The output shows the average silhouette width score, which is approximately 0.2136.

```
# Silhouette plot
sil <- silhouette(kmeans_fit$cluster, dist(scaled_data))

# Plot silhouette
plot(sil, col = 1:best_k, border = NA, main = "Silhouette Plot for Wine Dataset",
     xlab = "Silhouette Width", ylab = "Cluster")

# Average silhouette width score
avg_sil_width <- mean(sil[, 3])
cat("Average Silhouette Width Score:", avg_sil_width, "\n")

# Save best cluster value
best_cluster_value <- best_k
```

Two internal evaluation metrics, Within Cluster Sums of Squares (WSS) and Between Cluster Sums of Squares (BSS), are obtained and displayed in this section of the code (WSS). Higher values of BSS, which calculates the total variance between clusters, imply better separation. Lower values indicate that observations within clusters are closer to one another. WSS measures the variance within each cluster. The proportion of total variance explained by the clustering can be determined by comparing the BSS to TSS ratio. In this instance, the clustering accounts for roughly 22.9% of the total variance, as shown by the BSS/TSS Ratio is roughly 0.229.

The code calculates and prints the ratio of Between Cluster Sums of Squares (BSS) to Total Sum of Squares (TSS). BSS/TSS Ratio provides insight into the proportion of total variance explained by the clustering.

```
# Ratio of BSS (Between Cluster Sums of Squares) over TSS (Total Sum of Squares)
BSS <- kmeans_fit$betweenss
TSS <- kmeans_fit$totss
BSS_TSS_ratio <- BSS / TSS
cat("BSS/TSS Ratio:", BSS_TSS_ratio, "\n")
```

In this case, the BSS/TSS Ratio is approximately 0.229, indicating that about 22.9% of the total variance is explained by the clustering. Check below.

```
> cat("BSS/TSS Ratio:", BSS_TSS_ratio, "\n")
BSS/TSS Ratio: 0.2291265
> # Internal evaluation metrics: BSS and WSS
> cat("Between Cluster Sums of Squares (BSS):", BSS, "\n")
Between Cluster Sums of Squares (BSS): 6303.501
> cat("Within Cluster Sums of Squares (WSS):", kmeans_fit$tot.withinss, "\n")
Within Cluster Sums of Squares (WSS): 21207.5
```

Additionally, the code also prints two internal evaluation metrics: Between Cluster Sums of Squares (BSS) and Within Cluster Sums of Squares (WSS). BSS measures the total variance between clusters, indicating better separation with higher values, while WSS measures the variance within each cluster; lower values suggest that observations within clusters are closer to each other.

The cluster centers and cluster assignments are also printed to provide further insight into the clustering results.

The centroid coordinates for each feature in each cluster are printed by the "Cluster Centres" part of the code. They provide light on the traits of each cluster by representing the mean of each feature within each cluster. The arithmetic mean of every point in the cluster is the "cluster center". In comparison to other cluster centers, each point is closer to its own cluster center. As an illustration, the mean value of "fixed acidity" is roughly 0.188 in Cluster 1 and -0.128 in Cluster 2. Likewise, each cluster has mean values for "volatile acidity," "citric acid," and other properties.

```
Cluster Centers:
> cluster_centers <- kmeans_fit$centers
> print(cluster_centers)
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide
1     0.1875139       0.06906669  0.13998285      0.8605806  0.5567184           0.5966984
2    -0.1279937      -0.04714371 -0.09554982     -0.5874171 -0.3800062          -0.4072958
  total sulfur dioxide    density         pH   sulphates    alcohol
1            0.7882704  0.9690305 -0.1632178  0.10212593 -0.7920798
2           -0.5380595 -0.6614432  0.1114096 -0.06970936  0.5406597
```

https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html#:~:text=The%20%22cluster%20center%22%20is%20the,than%20to%20other%20cluster%20centers.

The cluster assignments for every dataset observation are printed in the "Cluster Assignments" section. Following k-means clustering, these assignments show which cluster each observation has been assigned to. This approach is often used when assignment by individuals is likely to result in contamination. For instance, Clusters 1 and 2 are host to the first and third observations, respectively. These assignments facilitate further analysis and interpretation of the clustering results and aid with comprehending the process of dividing the data into distinct clusters.

https://www.distillersr.com/glossary/cluster-assignment

```
Cluster Assignments:
> cluster_assignments <- kmeans_fit$cluster
> print(cluster_assignments)
   [1] 1 2 2 1 2 2 1 2 1 1 2 2 1 1 2 1 1 2 2 1 2 2 2 2 2 2 2 1 1 1 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2
  [47] 2 2 2 2 1 2 2 1 2 2 2 1 2 2 2 1 2 1 2 1 2 2 1 1 2 1 2 1 1 1 1 1 1 1 1 2 2 2 2 2 1 2 2 1 1 1
  [93] 1 1 1 2 1 1 2 2 1 2 2 1 2 1 2 2 1 2 1 2 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 2 1 1 2 1 1 1 1 1 2
 [139] 2 2 1 1 1 1 2 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 2 2 2 2 2 2 2 2 1 2 1 2
 [185] 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 2 1 2 2 2 2 2 2 1 1 2 2 2 2 2 1 1 1 1 1 2 1 2 1 1 2 2 2 1 1 1
 [231] 2 1 2 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 2 1 1 1 2 1 1 2 2 2 1 1 2 1
 [277] 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 2 1 2 1 1 1 1 1 1
 [323] 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 1 1 1 2 2 1 1 1 1 1 2 1 1 2 1 1 1 2 1
 [369] 1 1 2 1 2 1 1 2 1 2 1 2 2 2 1 2 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 2 2 1 1 1 1 1 2 2
 [415] 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 2 2 2 1 2 1 1 1 2 2 2 1 2
 [461] 2 2 1 1 2 2 2 2 1 2 1 1 2 1 1 1 1 1 1 1 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 1 1 2 1 1 2 2 2 2 1
 [507] 1 1 1 2 2 2 1 2 2 1 2 1 2 2 2 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2
 [553] 1 2 1 1 1 1 2 2 1 2 2 2 2 1 1 1 2 2 1 1 1 1 2 1 1 1 2 2 1 1 2 2 1 1 2 1 1 2 1 1 1 1 1 2 2 1
 [599] 2 2 1 1 1 1 1 2 1 2 2 2 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 1 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 1 1 1
 [645] 1 2 1 1 1 1 1 1 1 2 1 2 2 2 2 2 1 2 2 2 2 1 1 2 1 2 1 1 1 2 1 2 1 1 2 1 1 1 1 1 1 2 1 2 2 2
 [691] 2 2 2 1 1 2 2 2 2 1 1 1 1 2 2 1 2 1 1 2 2 1 1 1 2 2 2 2 2 1 2 2 1 2 2 2 2 1 1 2 2 2 1 1 1 2 2
 [737] 2 1 2 1 1 2 1 2 2 2 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 2 2 1 1 2 1 2 1 1 2 1 1 1 1 1 1 1
 [783] 2 1 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 2 2 1 2 2 1 2 1 1 1 1 2 2 1 1 2 2 1 2 2
 [829] 2 2 2 1 2 1 2 1 2 2 2 2 2 1 2 2 1 2 2 2 1 1 2 2 1 1 2 2 1 2 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2
 [875] 1 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 1 1 2 2 2 1 1 1 1 1 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2
 [921] 2 2 1 2 1 1 1 1 1 2 2 1 2 2 2 2 2 1 1 1 2 1 1 2 2 2 2 1 2 1 2 2 2 1 2 1 2 2 1 1 2 1 1 1 2 2
 [967] 1 2 1 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 1 1 1 1 2 2 2 1 1 1 1 1 2
 [ reached getOption("max.print") -- omitted 1502 entries ]
```

TASK 1 Sub task 2

Implementation of PCA

Principal Component Analysis (PCA) is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving most of the variability in the original data.

[https://link.springer.com/content/pdf/10.1007/s10462-022-10297-z.pdf]

The wine_data_no_outliers dataset undergoes to Principal Component Analysis (PCA). Using the scale() function, the data must first be standardised to have a mean of zero and a standard deviation of one. The prcomp() function is used to perform PCA on the standardized information after data scaling. By making sure the PCA technique runs efficiently, this method offers significant insights into the dataset's structure. As seen below,

```r
# e. Apply PCA to the dataset
# Scale the data
scaled_data <- scale(wine_data_no_outliers)

# Perform PCA analysis
pca <- prcomp(scaled_data)

summary(pca)
```

And the summary of the code get printed as below in the console.

```
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8    PC9    PC10
Standard deviation     1.8637 1.2497 1.1065 1.04120 0.97878 0.86145 0.84197 0.75311 0.6161 0.53600
Proportion of Variance 0.3158 0.1420 0.1113 0.09856 0.08709 0.06746 0.06445 0.05156 0.0345 0.02612
Cumulative Proportion  0.3158 0.4578 0.5691 0.66763 0.75472 0.82218 0.88663 0.93819 0.9727 0.99881
                          PC11
Standard deviation     0.11439
Proportion of Variance 0.00119
Cumulative Proportion  1.00000
> 
```

The cumulative score per principal component (PC) is calculated in order to determine the PCs that are responsible for at least 85% of the variation. It depends on how many primary components give a cumulative score higher than 85%. An it gets calculated by below part of the code.

```r
# Cumulative score per principal components (PC)
cumulative_score <- cumsum(pca$sdev^2 / sum(pca$sdev^2) * 100)

# Plot cumulative score
plot(cumulative_score, xlab = "Number of Principal Components",
     ylab = "Cumulative Score", type = "b")

# Identify the number of principal components with cumulative score > 85%
num_components <- which(cumulative_score > 85)[1]

# Create a new transformed dataset with selected principal components
transformed_data <- as.data.frame(-pca$x[, 1:num_components])
```

And in this case its 7 and it gets saved in a dataframe called transformed data. And the data looks like below. The cumulative score plot illustrates how the 7 PC were selected based on the ability to account for more than 85 percent of the variance in the data.

	A	B	C	D	E	F	G
1	PC1	PC2	PC3	PC4	PC5	PC6	PC7
2	1.01187521	-1.55594221	-1.5309414	0.70666983	-0.32567371	-0.5700513	-0.06749237
3	-0.85171272	-1.72961232	-1.1027853	-0.86570958	1.56449535	0.95365931	-1.71033071
4	-0.85171272	-1.72961232	-1.1027853	-0.86570958	1.56449535	0.95365931	-1.71033071
5	1.01187521	-1.55594221	-1.5309414	0.70666983	-0.32567371	-0.5700513	-0.06749237
6	-0.02289208	-1.43885561	-0.39920709	-0.96169131	-0.7425994	0.58613247	1.55372423
7	0.07816717	-0.39109	0.07982861	-1.37954452	-2.40997332	0.07645127	-0.43983547
8	0.77386015	-2.57915461	0.24964223	0.18432046	1.51974571	0.93899342	0.1997253
9	-0.78853599	0.59768673	1.78005838	0.24562544	0.4823658	-0.14047183	0.55363972
10	2.29855641	-0.28375663	0.72781615	-1.88965507	-1.36108808	1.04168081	-0.61764068
11	0.46468074	2.55006538	1.54517377	1.90888431	1.09624902	2.71630048	0.72560354
12	0.23231462	-3.32275831	0.9095443	1.13408358	0.28334229	0.65212522	0.85475435
13	0.25863259	-3.11746524	0.6004182	0.87251481	0.190989	0.65305018	0.92525045
14	1.61970201	1.34277998	0.76607354	-0.31247186	-0.47203766	0.51002183	0.62807222
15	1.61970201	1.34277998	0.76607354	-0.31247186	-0.47203766	0.51002183	0.62807222
16	-1.13311941	-1.44787333	-3.4626439	-0.42202841	1.06859365	0.60516443	-0.28495266
17	1.61970201	1.34277998	0.76607354	-0.31247186	-0.47203766	0.51002183	0.62807222
18	0.79193215	0.47303225	1.31695419	0.5235618	-1.29015914	0.07545392	0.79881308
19	-0.93705467	-2.09382407	0.7577464	-0.87711347	-0.75090915	0.9520745	0.98381872
20	-0.93705467	-2.09382407	0.7577464	-0.87711347	-0.75090915	0.9520745	0.98381872
21	0.56112614	-0.76271459	-0.54121739	2.18009629	0.47114593	0.46963882	0.66116145
22	-0.80569318	-4.34530311	-0.27308189	0.29236938	0.72018657	-0.09079246	-0.57865578
23	-2.02396073	1.66080222	-3.05054473	2.04842641	-0.15315836	-0.15419181	0.43243378
24	-1.04237553	1.67985594	0.49060181	2.20964455	-1.00045171	0.41914532	0.26269051
25	-0.6794538	-1.00867148	1.42154081	-1.04081085	0.52655314	0.92128955	-0.20529378
26	-1.03691627	-0.11623395	1.34340264	1.5930197	-0.29156162	0.70685801	-1.60290417
27	-1.03691627	-0.11623395	1.34340264	1.5930197	-0.29156162	0.70685801	-1.60290417
28	-0.6794538	-1.00867148	1.42154081	-1.04081085	0.52655314	0.92128955	-0.20529378
29	1.12332508	-0.24588014	-0.64219733	2.13623535	0.1579423	-1.47378555	-0.02653932
30	2.02444878	0.55731883	0.73388338	-1.29587482	-2.07319794	1.52009399	-0.88019991
31	2.64230682	0.87287036	0.91559674	-1.40931055	-1.12047983	1.4652995	-0.58394416
32	-0.90484802	-0.72710372	-1.15880025	1.61747357	-0.02386593	1.2947761	0.28005225
33	0.40976061	-2.36326994	0.70275446	1.16745806	-0.33610411	1.83258839	-0.56003053
34	-0.26011394	-1.98512153	-0.56230366	0.51230378	0.99783182	1.54672929	-1.33861152
35	4.88739779	-1.61483465	0.80709777	-0.91328466	0.10804638	-0.72461775	-0.1860864

Eigen Values and Eigen Vectors

In linear algebra, eigenvectors and eigenvalues are related. In the analysis of linear changes, both names are utilized. The unique collection of scalar values known as eigenvalues is likely linked to the set of linear equations in the matrix equations, most likely. Another name for the eigenvectors is characteristic roots. It is a non-zero vector that, after applying linear transformations, can only be altered by its scalar factor. Additionally, an eigenvalue is the appropriate factor that scales the eigenvectors. The output of the eigen values and eigen vectors as below

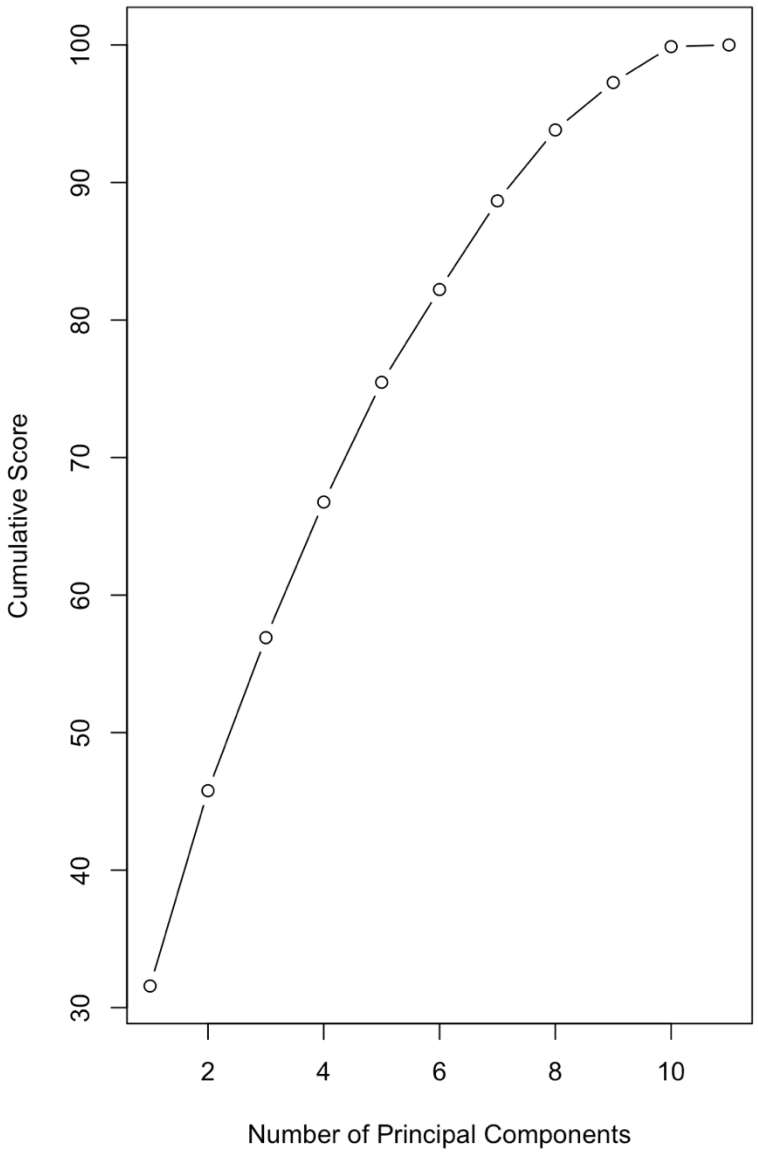
```
Eigenvalues:
> print(eigenvalues)
 [1] 3.47351016 1.56185722 1.22442522 1.08410525 0.95801651 0.74209300 0.70891746 0.56717138
 [9] 0.37952025 0.28729796 0.01308559
```

```
> print(eigenvectors)
                              PC1           PC2           PC3         PC4           PC5         PC6
fixed acidity        -0.154857243  0.5843682339 -0.101839425  0.06560658 -0.255334725  0.04603363
volatile acidity      0.006311371 -0.0333504117  0.665326003  0.26046915 -0.496752711 -0.22563884
citric acid          -0.096003773  0.3489516259 -0.524446368  0.26023625 -0.008457225 -0.16982690
residual sugar       -0.414931612  0.0226132841  0.224279230  0.04644007  0.083860502  0.51595232
chlorides            -0.313473358 -0.0534739489 -0.060315997 -0.46511997 -0.152337371 -0.62299057
free sulfur dioxide  -0.288943113 -0.2402073681 -0.089137495  0.57301010  0.300870712 -0.15508440
total sulfur dioxide -0.396680194 -0.2093591127  0.005092485  0.35442085  0.003682813 -0.28431507
density              -0.500872209 -0.0009547605  0.048593108 -0.15524280 -0.039640542  0.30573351
pH                    0.113590864 -0.5948267471 -0.209172251 -0.13924253  0.009157664  0.08686419
sulphates            -0.047361418 -0.2740631315 -0.404312888  0.11972674 -0.750854855  0.23990363
alcohol               0.435277649  0.0424531235  0.020789370  0.36035733 -0.041894420  0.01855961
                             PC7         PC8         PC9         PC10         PC11
fixed acidity        -0.274744591  0.64497677 -0.16210324 -0.086940617 -0.165005575
volatile acidity      0.350291216  0.06616669  0.08513471 -0.237201090 -0.008640979
citric acid           0.684205665 -0.13663244  0.08664787 -0.053802183 -0.011318001
residual sugar        0.230772930 -0.19612550 -0.40927678  0.155641897 -0.467086631
chlorides            -0.007225697 -0.13839848 -0.49783017  0.003780448 -0.022824161
free sulfur dioxide  -0.259539075 -0.02370626 -0.20978962 -0.543434359  0.026396358
total sulfur dioxide -0.090842105  0.18748219  0.25177395  0.695281889 -0.046000091
density               0.132215265  0.11676304  0.01408770 -0.050371893  0.770564648
pH                    0.365690063  0.62017723 -0.13585433 -0.085546577 -0.135587084
sulphates            -0.227962623 -0.26317075  0.00394145 -0.028314855 -0.038249955
alcohol               0.039696996  0.01074038 -0.64618280  0.345902474  0.370742450
>
```

This graph indicates the cumulative proportion of variance explained by the principal components. The x-axis represents the number of principal components, while the y-axis represents the cumulative proportion of variance explained.

The total amount of variation explained by the first n main components is shown by each point on the plot. Determining the number of primary components to retain in the analysis is aided by the plot. The aim in general is to keep enough primary components to make up for the majority of the variation in the data.

In this case, the number of primary components that account for at least 85% of the variation in the dataset in its entirety is ascertained using the graph.

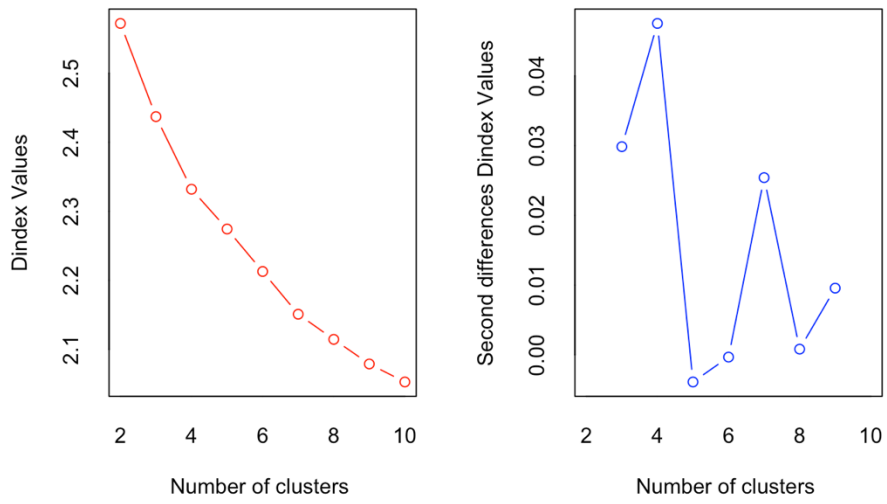

Cumulative Score
30
40
50
60
70
80
90
100
2
4
6
8
10
Number of Principal Components

NBClust Method

In this code, the **NbClust** function is used to determine the optimal number of clusters for the PCA-transformed dataset using k-means clustering. The result, stored in **best_k_pca**, suggests that 2 clusters would be the optimal choice according to the majority rule.

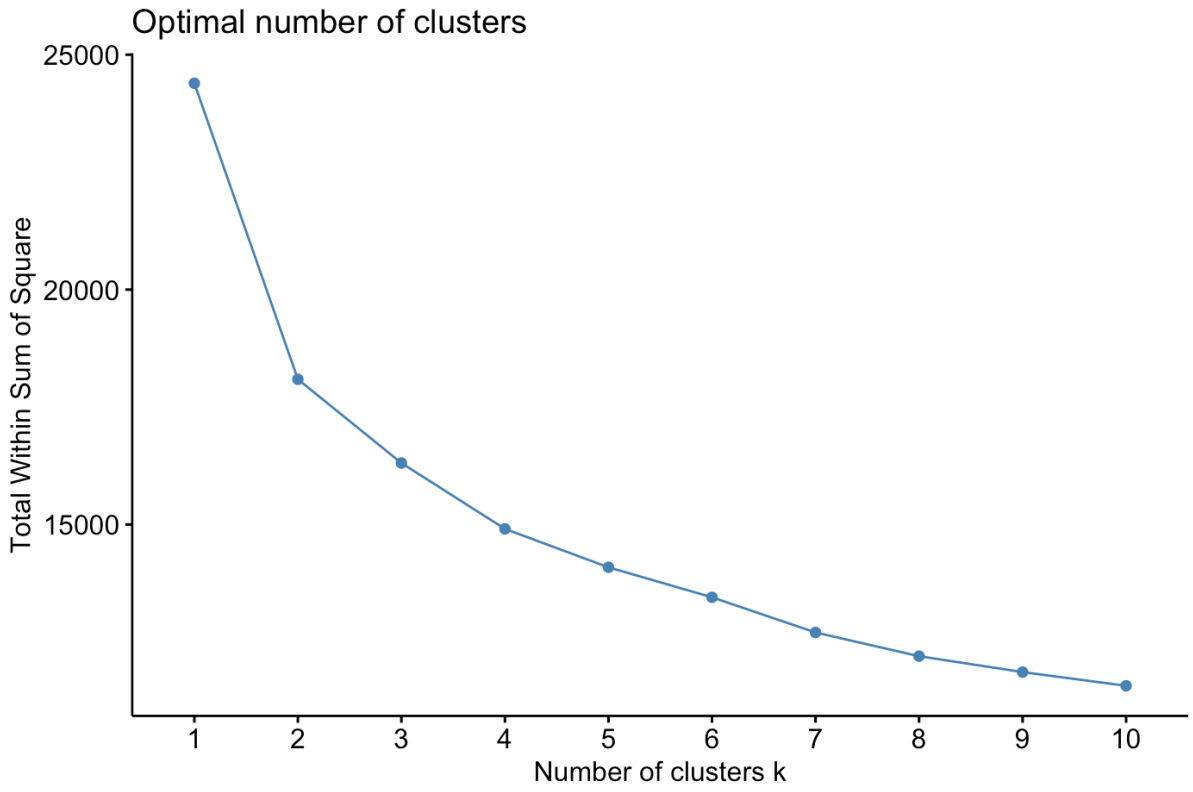
```
*******************************************************************
> best_k_pca <- nb_pca$Best.nc[1]
> nb_clusters_pca <- cbind(2:10, nb_pca$Best.n[2:10])
> nb_pca <- NbClust(transformed_data, min.nc = 2, max.nc = 10, method = "kmeans")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

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

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

                   ***** Conclusion *****

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

One visual aid for figuring out how many clusters exist is the D index. We search for a significant increase in the measure's value in the D index plot by seeking a significant knee, or the significant peak in the D index second differences plot. The suggested number of clusters based on the D index is displayed in the output. For instance, if the D index indicates that a pair of clusters is the ideal number, then this method would recommend two clusters as the optimum option.



Elbow Method

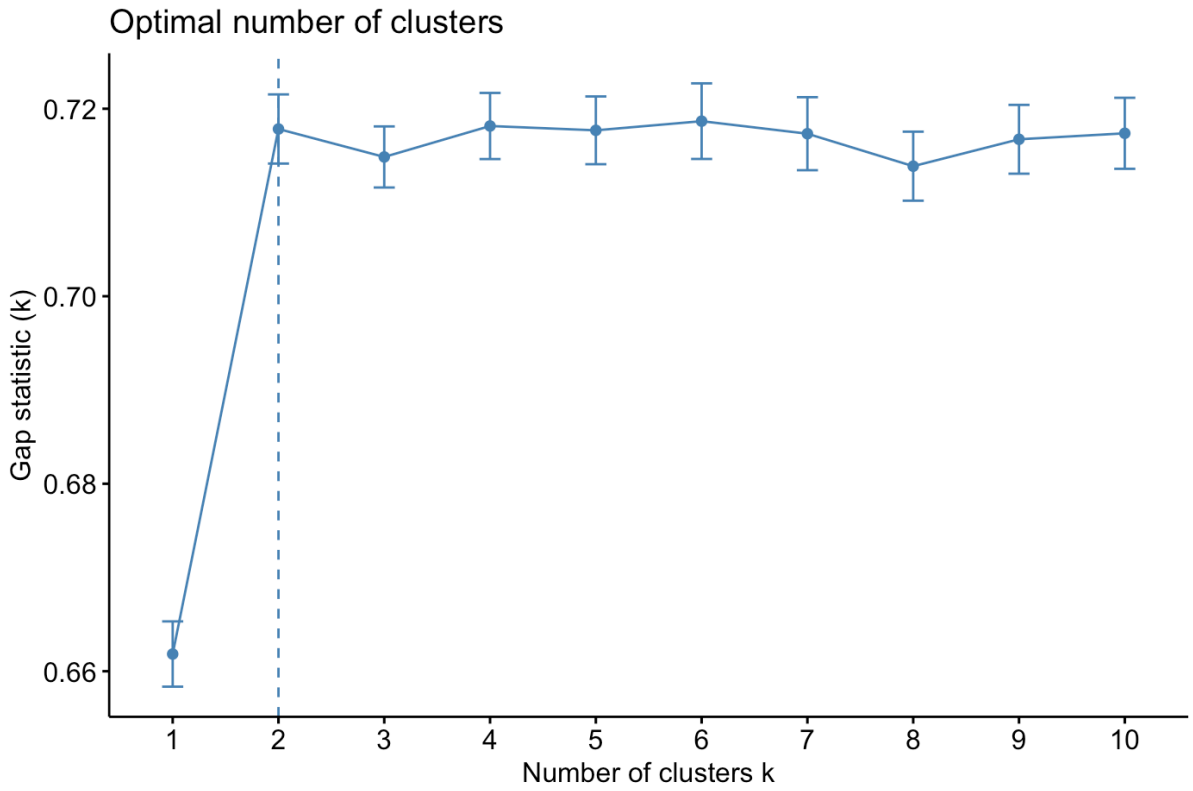
```
> best_k_elbow_pca <- min_wss_index_pca + 1 # Adjusting to match k value
> cat("Best k using the Elbow Method for PCA-based Dataset:", best_k_elbow_pca, "\n")
Best k using the Elbow Method for PCA-based Dataset: 2
```



According tp the above graph the bend is near 2. So the number of clusters in this case is also 2.

Gap Statistic

To get the perfect number of clusters, the algorithm computes the Gap Statistic for the PCA-based dataset. By varying the number of clusters to be checked between 2 and 10, it executes the clusGap function. The maximum gap statistic value is used to calculate the ideal number of clusters. It does, however, print an error notice if there is a computation error. The Gap Statistic indicates that three clusters would be the ideal number in this scenario. This, however, might not match the Gap Statistic plot's visual representation because the graph might indicate a different ideal number of clusters. In this case, the graph shows that 2 clusters are the best option even though the result advises 3 clusters.



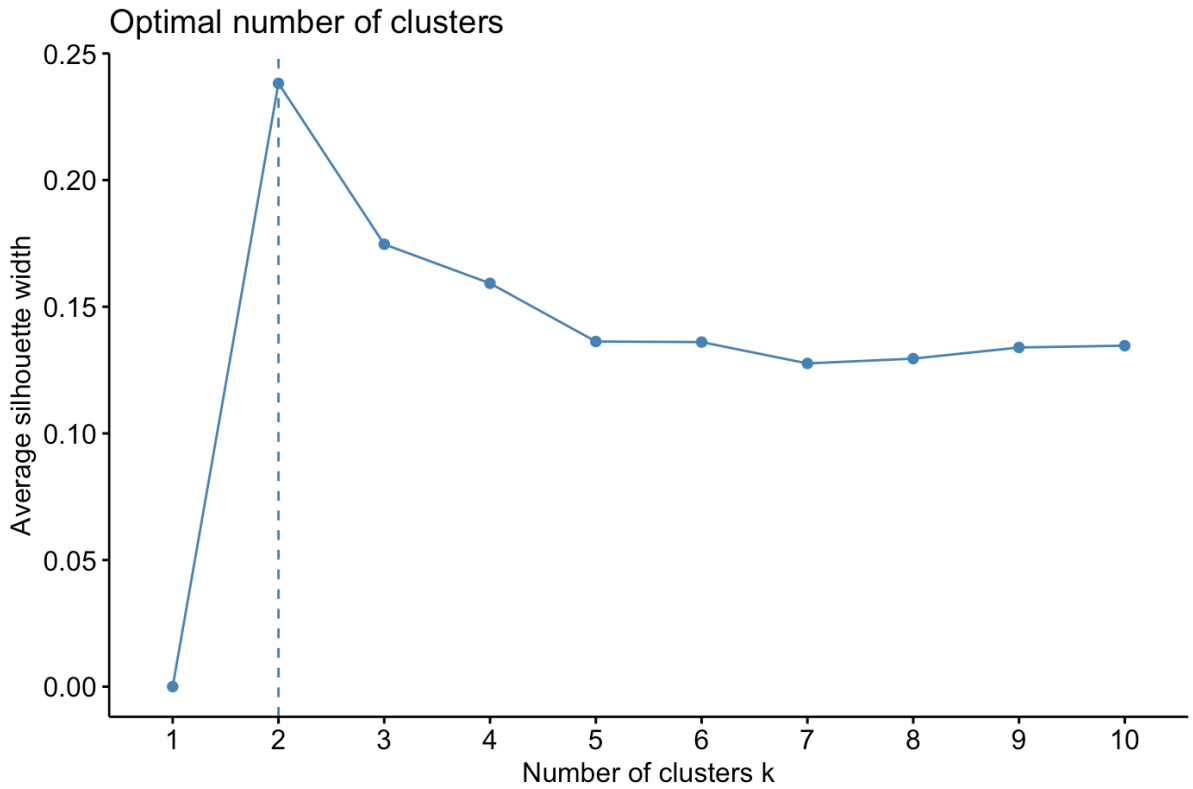
According to this gap statistic method gets 2 as the cluster value.

```
+   }
+ )
Clustering k = 1,2,..., K.max (= 10): .. done
Bootstrapping, b = 1,2,..., B (= 50)  [one "." per sample]:
.................................................. 50
> if (!is.null(gap_stat_pca)) {
+   gap_clusters_pca <- cbind(2:10, gap_stat_pca$Tab[2:10, "gap"])
+   max_gap_index <- which.max(diff(gap_stat_pca$Tab[2:10, "gap"]))
+   best_k_gap_pca <- max_gap_index + 1
+ } else {
+   cat("Error: Gap Statistic calculation failed\n")
+   gap_clusters_pca <- NULL
+   best_k_gap_pca <- NA
+ }
> cat("Gap Statistic suggests", best_k_gap_pca, "as the optimal number of clusters.\n")
Gap Statistic suggests 3 as the optimal number of clusters.
```

But according to this it says cluster is 3. Since most of the methods predict it as 2. Its safe to say the number of clusters is 2.

Silhouette Method

Using the transformed information (transformed_data), this code uses the silhouette approach to get the ideal number of clusters (k) for k-means clustering. The cluster numbers (k = 2 to k = 10) are iterated through. The silhouette width is calculated using cluster.stats(), which gauges an object's degree of similarity to its own cluster in relation to other clusters, and k-means clustering is carried out for each k. Next, the index of the maximum silhouette width is found using which.max() to get the ideal k. Since k = 2 is where the loop begins, we must add 1 to get the true value of k. Finally, fviz_nbclust() is used to visualize the silhouette method result to figure out the ideal number of clusters.



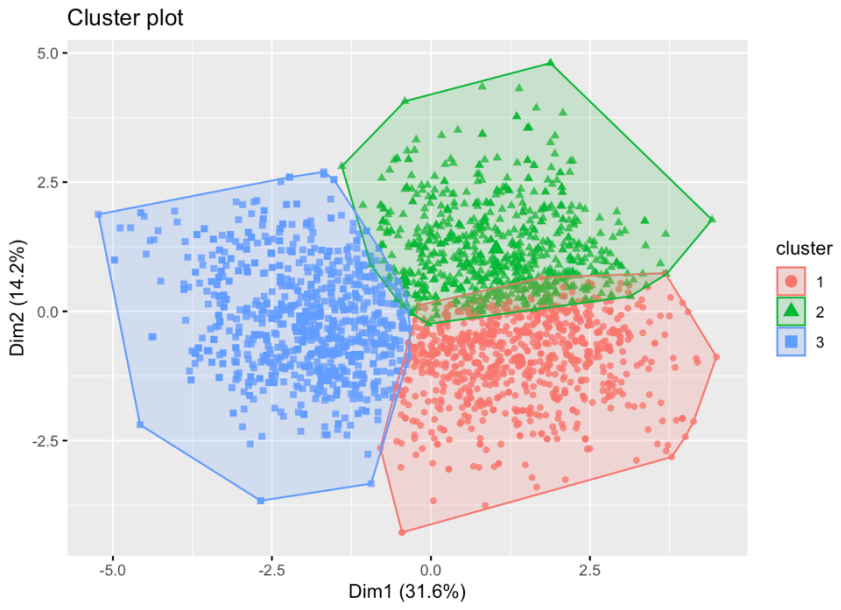
This method also predicts the number of clusters are 2.
So according to the automatic method. Number of clusters are as below

```
> print(best_k_values)
          NbClust      Elbow_Method     Gap_Statistic Silhouette_Method
                2                 2                 3                 2
> # Display cluster assignments for each method
```

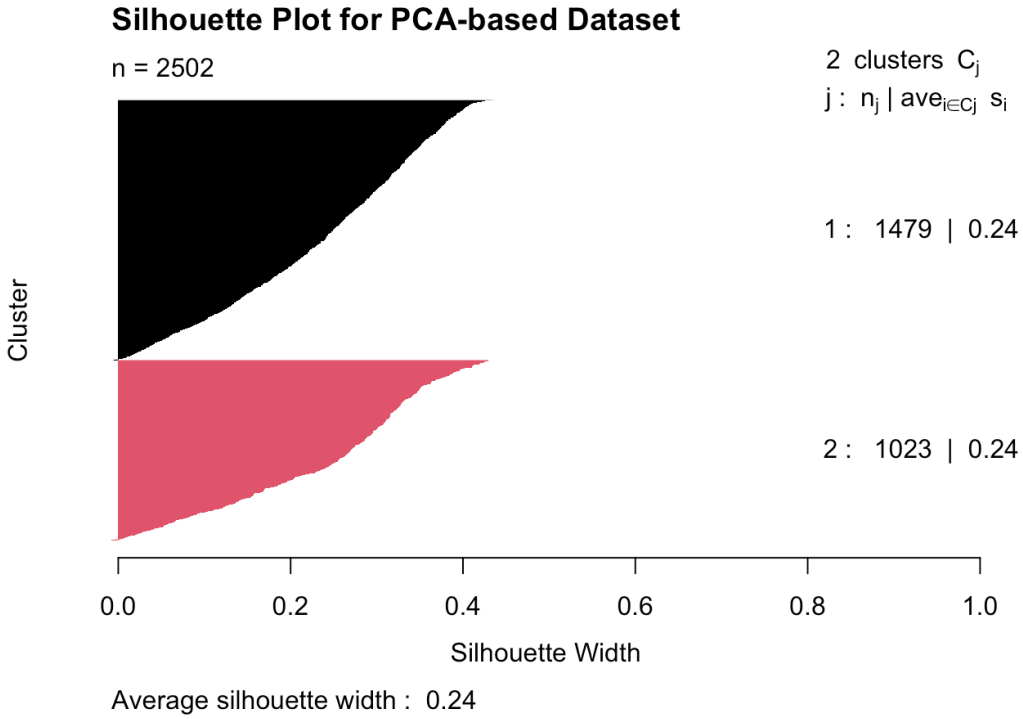
K=2 KMeans Clustering for PCA

K=3 KMeans clustering for PCA

Since Gap statistic method predicts 3 as its number of cluster lets perform k=3 kmeans graph as well



Silhouette Plot for PCA-based Dataset



Discussion on silhouette plot

The distance of each point inside a cluster to points within neighboring clusters is shown by the silhouette plot. It gives each observation's silhouette widths as a visual representation. These widths quantify the degree to which an observation resembles its own cluster in relation to other clusters. The average silhouette width score in our PCA-based dataset is 0.2381944, which denotes a decent quality for the clusters that were produced. The presence of overlapping clusters or clusters with observations near the decision boundary between clusters can be detected by a silhouette width that is close to zero. Therefore, the silhouette plot aids in our assessment of the clusters' separation and compactness.

The silhouette widths for each observation are found in the sil_pca[, 3] column, which is used to determine the average silhouette width score. The average silhouette width score can be obtained by taking the mean of these values.

The relatively high average silhouette width score adds to the quality of the clustering results by indicating that the k-means analysis's clusters have a suitable distance between each other. A better grasp of the clustering quality is rendered possible by the silhouette plot, which helps assess the compactness and separation of the clusters.

Calinski-Harabasz

The Variance Ratio Criterion, or Calinski-Harabasz Index, is a metric for assessing the quality of clustering that compares the dispersion inside and between clusters. It is determined by multiplying the total within-cluster dispersion by the total between-cluster dispersion. Greater dispersion across clusters is shown by a higher index value. The computed Calinski-Harabasz Index value for the PCA-based dataset is saved in the ch_index variable in this code, and it is then printed using cat. The PCA-based dataset's computed ch_index value of 870.7789 indicates that there is good cluster separation.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.calinski_harabasz_score.html

The ratio of within-cluster dispersion to between-cluster dispersion is determined by the Calinski-Harabasz Index. Better separation between clusters is indicated by an index value that is higher. The PCA-based dataset in our instance has a Calinski-Harabasz Index of 870.7789. This shows that the k-means analysis generated clusters with good separation between them, which enhances the quality of the clustering outcomes.

```
> # Print Calinski-Harabasz Index
> cat("Calinski-Harabasz Index for PCA-based Dataset:", ch_index, "\n")
Calinski-Harabasz Index for PCA-based Dataset: 870.7789
```

2nd Task - Financial Forecasting

For forecasting and understanding historical patterns in a variety of business applications, time series analysis is essential. In the global economy, exchange rates—which show how much one country's currency is worth about another—are of particular significance. Because of their deterministic chaos, intrinsic noise, and non-stationarity, exchange rates are hard to predict. These rates are set by the foreign exchange market, which considers the relative supply and demand for every currency. Since there are so many economic factors that affect exchange rates, that anticipating them presents difficulties. Exchange rates exhibit trend cycles and abnormalities like other economic time series, which reduces the usefulness of conventional time series analysis. Neural Networks (NN) have been popular as an option for predicting currency exchange rates since they can mimic human learning abilities and handle the intricacies of the task.

Why do we need to normalize data?

Better Convergence: By standardizing the input data, the optimization process is guaranteed to converge more quickly. Without normalization, the optimization method may take longer to locate the ideal solution because certain input features may have a significantly greater scale than others.

Normalization ensures that the model is not biassed towards any specific characteristic because of scale differences, helping the neural network generalize to new data more effectively.

Balanced Impact of Features: In datasets including several features, the magnitude of each feature can differ. Assuring that every feature contributes equally to the learning process by normalizing them all to the same scale stops some features from predominating over others because of their greater scale.

AR approach

The autoregressive (AR) approach is a time series forecasting method that uses past observations of a variable to predict future values. In an AR model, the value of the variable at a given time point is regressed on its own past values, called lag variables.

[Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50, 159-175.

https://machinelearningmastery.com/autoregression-models-time-series-forecasting-python/]

In broad terms, the forecasting horizon and the unique properties of the exchange rate data determine which input variables are employed. MLP-NN models may efficiently capture the underlying patterns in exchange rate movements by carefully selecting relevant input variables, which leads to more accurate and reliable forecasts.

Lagged exchange rate information is one of the inputs available in this time series forecasting task for determining the USD/EUR exchange rate. The Multilayer Perceptron (MLP) model's input variables are built using the autoregressive (AR) method, with each input variable denoting the exchange rate from a prior time step. The input delay parameter controls how many lag values are included as input variables. The exchange rate at the prior time step (t-1) is one of the input variables for each data point, for example, if the input delay is set to 1. The exchange rates at the two preceding time steps (t-1 and t-2), and further along, constitute the input variables if the input delay is set to 2. In order to determine the optimum amount of delay inputs to include in the input variables for training the MLP model, the model experiments with various input delays (1, 2, 3, and 4).

Several strategies and techniques are used in the discipline of financial time series forecasting to provide input vectors for neural network models. A potent technique for time series forecasting, the Autoregressive Integrated Moving Average (ARIMA) model combines the moving average (MA), the distinction (I) process, and the autoregressive (AR) model.

[https://www.sciencedirect.com/topics/mathematics/autoregressive-integrated-moving-average]
The Autoregressive Conditional Heteroskedasticity (ARCH) model is another widely used technique for modeling volatility clustering in financial time series data. A statistical model called Autoregressive Conditional Heteroskedasticity (ARCH) is used to examine volatility in time series data and predict future volatility, especially in financial markets. Because it takes into consideration the fact that volatility varies over time, it offers a more accurate model of volatility. The ARCH model is extremely useful for modeling financial time series data when volatility clusters because it makes an assumption that the variance of the current error term is proportional to the squares of the prior error terms.

[https://www.investopedia.com/terms/a/autoregressive-conditional-heteroskedasticity.asp#:~:text=Autoregressive%20conditional%20heteroskedasticity%20(ARCH)%20is,more%20closely%20resembles%20real%20markets]

The Autoregressive (AR) technique is used in this code to forecast the USD/EUR exchange rate over time. Initially, the data on exchange rates is loaded and cleared. Afterward, an input/output matrix construction function named construct_input_output_matrix is written to generate matrices for MLP model training and testing. Lagged observations of the exchange rate data compose these matrices, which are used as the MLP's input features (X) and target values (y).

```
# Function to construct input/output matrices for MLP training/testing with different input delays
construct_input_output_matrix <- function(data, input_delay) {
  X <- matrix(NA, nrow = length(data) - input_delay, ncol = input_delay)
  y <- vector(mode = "numeric", length = length(data) - input_delay)
  for (i in 1:(length(data) - input_delay)) {
    X[i, ] <- data[(i):(i + input_delay - 1)]
    y[i] <- data[i + input_delay]
  }
  # Normalize the input and output matrices
  X_normalized <- apply(X, 2, normalize)
  y_normalized <- normalize(y)
  return(list(X = X_normalized, y = y_normalized))
}
```

First it normalizes the data and then after training the MLP model it denormalize the data. To ensure that the forecasts have the same scale as the original data and to assist the neural network converging more quickly during training, the data should be normalized before being trained and denormalized after prediction. Before the data is utilized as input for training the MLP model, it is scaled between 0 and 1 in this code using the normalize function.

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

```
# Make predictions for testing data
test_output <- predict(best_model, as.data.frame(X_test))
test_output <- test_output * (max(train_data) - min(train_data)) + min(train_data)
test_actual <- y_test * (max(train_data) - min(train_data)) + min(train_data)
```

```
          [,1]      [,2]      [,3]      [,4]
 [1,] 0.4161137 0.3810427 0.4023697 0.3582938
 [2,] 0.4071090 0.4161137 0.3810427 0.4023697
 [3,] 0.4146919 0.4071090 0.4161137 0.3810427
 [4,] 0.3834123 0.4146919 0.4071090 0.4161137
 [5,] 0.4180095 0.3834123 0.4146919 0.4071090
 [6,] 0.4649289 0.4180095 0.3834123 0.4146919
 [7,] 0.4388626 0.4649289 0.4180095 0.3834123
 [8,] 0.4909953 0.4388626 0.4649289 0.4180095
 [9,] 0.4800948 0.4909953 0.4388626 0.4649289
[10,] 0.4867299 0.4800948 0.4909953 0.4388626
[11,] 0.5616114 0.4867299 0.4800948 0.4909953
[12,] 0.5597156 0.5616114 0.4867299 0.4800948
[13,] 0.5559242 0.5597156 0.5616114 0.4867299
[14,] 0.5767773 0.5559242 0.5597156 0.5616114
[15,] 0.6127962 0.5767773 0.5559242 0.5597156
[16,] 0.5938389 0.6127962 0.5767773 0.5559242
[17,] 0.6009479 0.5938389 0.6127962 0.5767773
[18,] 0.6061611 0.6009479 0.5938389 0.6127962
[19,] 0.6379147 0.6061611 0.6009479 0.5938389
[20,] 0.6360190 0.6379147 0.6061611 0.6009479
[21,] 0.5464455 0.6360190 0.6379147 0.6061611
[22,] 0.4985782 0.5464455 0.6360190 0.6379147
[23,] 0.4876777 0.4985782 0.5464455 0.6360190
```

```
> print(y_test)
 [1] 0.4071090 0.4146919 0.3834123 0.4180095 0.4649289 0.4388626 0.4909953
 [8] 0.4800948 0.4867299 0.5616114 0.5597156 0.5559242 0.5767773 0.6127962
[15] 0.5938389 0.6009479 0.6061611 0.6379147 0.6360190 0.5464455 0.4985782
[22] 0.4876777 0.4829384 0.4473934 0.4526066 0.4497630 0.4526066 0.4473934
[29] 0.4056872 0.3649289 0.3800948 0.3379147 0.3720379 0.4668246 0.4672986
[36] 0.4635071 0.5127962 0.4895735 0.4810427 0.5118483 0.5360190 0.5450237
[43] 0.5530806 0.5559242 0.5729858 0.5691943 0.5710900 0.5838863 0.5473934
[50] 0.5725118 0.5843602 0.6000000 0.6303318 0.6056872 0.5933649 0.5620853
[57] 0.5658768 0.5696682 0.5971564 0.6099526 0.6459716 0.6194313 0.6161137
[64] 0.6303318 0.6194313 0.6284360 0.6004739 0.5521327 0.5379147 0.5222749
[71] 0.5454976 0.5014218 0.5236967 0.5677725 0.5687204 0.5872038 0.5938389
[78] 0.5753555 0.6104265 0.6137441 0.6109005 0.6943128 0.6928910 0.6781991
[85] 0.6767773 0.6985782 0.6744076 0.6995261 0.6981043 0.6981043 0.7255924
[92] 0.7488152 0.7213270 0.7146919 0.7203791 0.6876777
```

Standard statistical indices

Various metrics such as mean squared error (MSE), root mean square error (RMSE), symmetric mean absolute percentage error (SMAPE), mean absolute error (MAPE), and mean squared error (MAE) are used to assess the quality of a model.

[https://search.r-project.org/CRAN/refmans/DescTools/html/MAE.html]

MAPE calculates the mean absolute percentage error. MAPE is a measure of the average absolute percentage difference between predicted and actual values. One of the simplest approaches is MAPE, which is also simple to interpret.

SMAPE calculates the symmetric mean absolute percentage error. sMAPE is a measure of the average absolute percentage difference between two values, normalized by their sum.

MAE calculates mean absolute error. The average magnitude of errors from expected and actual values is determined by the MAE.

RMSE calculates the root mean squared error. The difference between values predicted by a model and observed values is measured by the Root Mean Square Error (RMSE).

The distribution of the items in a set of elements is described by these values. These indices help analyze the relationships among subsets of items and support or refute certain correlations and rules that govern how the subsets behave.

```
# Make predictions for training and testing data
train_predicted <- predict(mlp_model, X_train)
test_predicted <- predict(mlp_model, X_test)

# Calculate statistical indices for training and testing data
train_rmse <- rmse(train_predicted, y_train)
train_mae <- mae(train_predicted, y_train)
train_mape <- mape(train_predicted, y_train)
train_smape <- smape(train_predicted, y_train)

test_rmse <- rmse(test_predicted, y_test)
test_mae <- mae(test_predicted, y_test)
test_mape <- mape(test_predicted, y_test)
test_smape <- smape(test_predicted, y_test)
```

Comparison table for MLPs

	PARAMETERS	RMSE	MAE	MAPE	sMAPE
Model 1	Inputs - 4 Hidden layers – 1	0. 02892132	0. 02137571	0. 04141387	0. 04087185
Model 2	Inputs - 2 Hidden layers – 1	0. 02912291	0. 02182881	0. 04214087	0. 04153924
Model 3	Inputs - 2 Hidden layers – 2	0. 02955704	0. 02180416	0. 04182598	0. 04125232
Model 4	Inputs - 2 Hidden layers – 2	0. 02899365	0. 02177662	0. 04191687	0. 04140443
Model 5	Inputs - 3 Hidden layers – 2	0. 0285261	0. 02094342	0. 04039594	0. 03996475
Model 6	Inputs - 4 Hidden layers – 1	0. 02917991	0. 02161308	0. 04179155	0. 04118321
Model 7	Inputs - 1 Hidden layers – 2	0. 02899763	0. 02153599	0. 04168114	0. 04110278
Model 8	Inputs - 3 Hidden layers – 2	0. 02904544	0. 02179281	0. 04191642	0. 04135211
Model 9	Inputs - 4 Hidden layers – 2	0. 02891588	0. 02164485	0. 04195956	0. 04146519
Model 10	Inputs - 2 Hidden layers – 2	0. 02908925	0. 02157586	0. 04152226	0. 04098827
Model 11	Inputs - 3 Hidden layers – 1	0. 02918842	0. 02158138	0. 04177703	0. 04116643
Model 12	Inputs - 2	0. 02911021	0. 02160445	0. 04161418	04110174

	Hidden layers – 2				

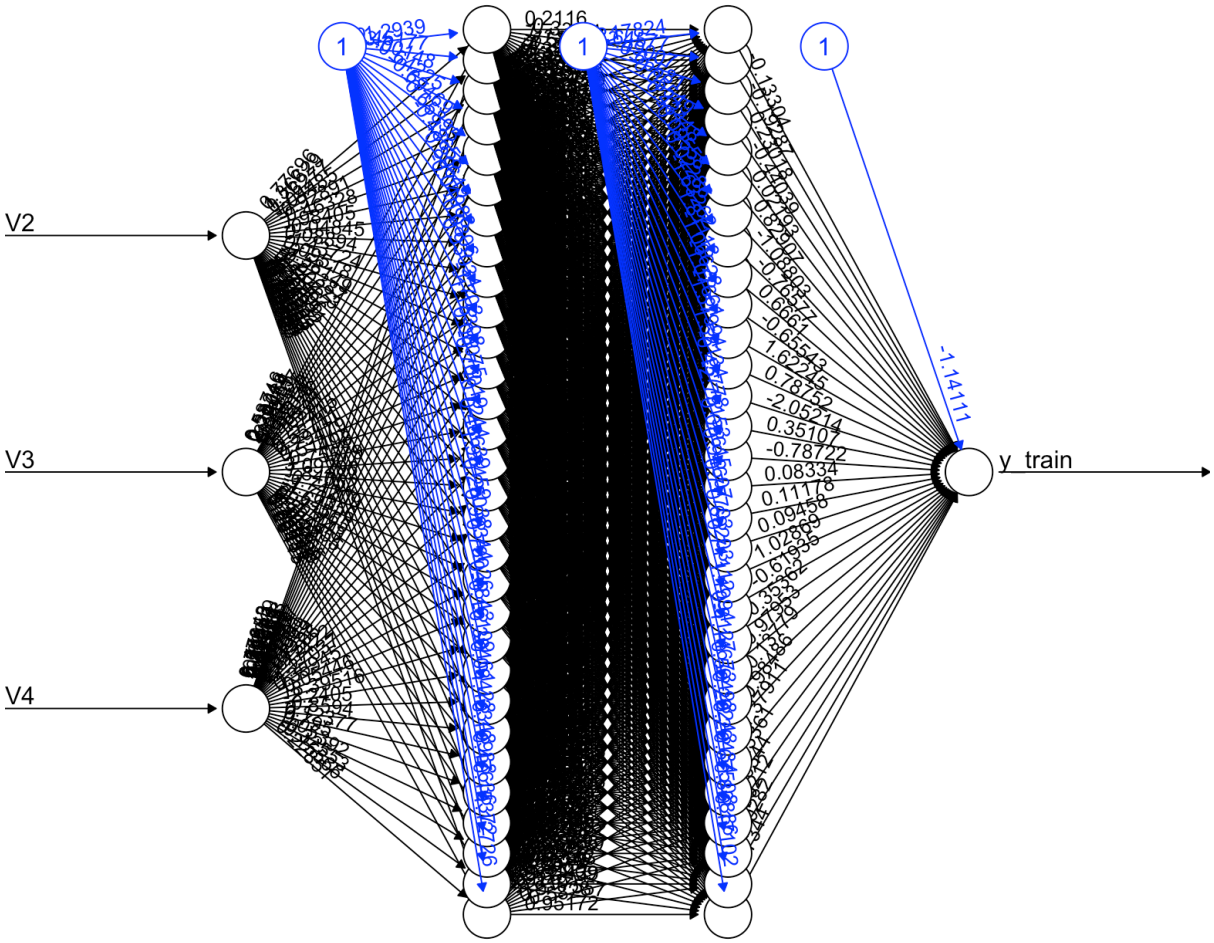
The values also get printed in the console as below.

```
Model 1 - Testing RMSE: 0.02892132 MAE: 0.02137571 MAPE: 0.04141387 sMAPE: 0.04087185
Model 2 - Testing RMSE: 0.02912291 MAE: 0.02182881 MAPE: 0.04214087 sMAPE: 0.04153924
Model 3 - Testing RMSE: 0.02955704 MAE: 0.02180416 MAPE: 0.04182598 sMAPE: 0.04125232
Model 4 - Testing RMSE: 0.02899365 MAE: 0.02177662 MAPE: 0.04191687 sMAPE: 0.04140443
Model 5 - Testing RMSE: 0.0285261 MAE: 0.02094342 MAPE: 0.04039594 sMAPE: 0.03996475
Model 6 - Testing RMSE: 0.02917991 MAE: 0.02161308 MAPE: 0.04179155 sMAPE: 0.04118321
Model 7 - Testing RMSE: 0.02899763 MAE: 0.02153599 MAPE: 0.04168114 sMAPE: 0.04110278
Model 8 - Testing RMSE: 0.02904544 MAE: 0.02179281 MAPE: 0.04191642 sMAPE: 0.04135211
Model 9 - Testing RMSE: 0.02891588 MAE: 0.02164485 MAPE: 0.04195956 sMAPE: 0.04146519
Model 10 - Testing RMSE: 0.02908925 MAE: 0.02157586 MAPE: 0.04152226 sMAPE: 0.04098827
Model 11 - Testing RMSE: 0.02918842 MAE: 0.02158138 MAPE: 0.04177703 sMAPE: 0.04116643
Model 12 - Testing RMSE: 0.02911021 MAE: 0.02160445 MAPE: 0.04161418 sMAPE: 0.04110174
```

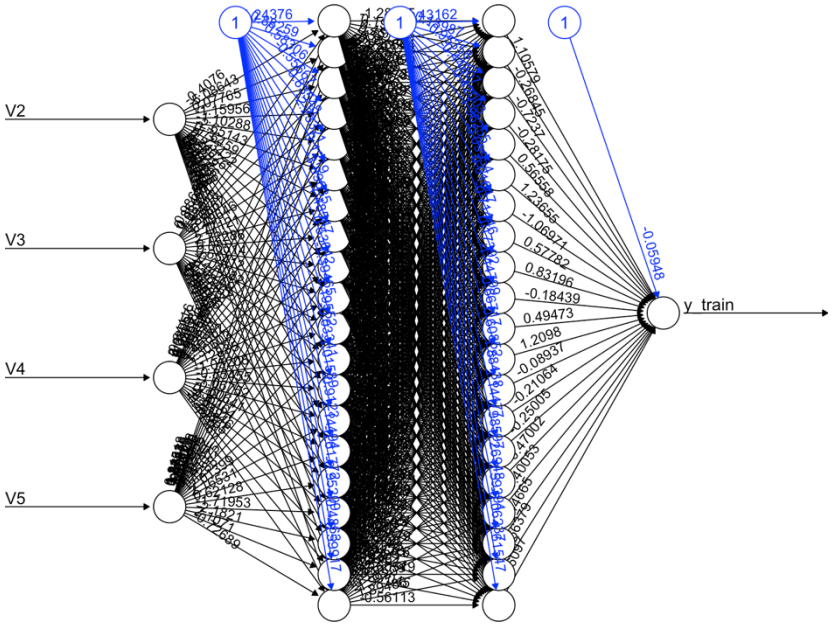
And the MLP models are as below.

Best MLP Networks

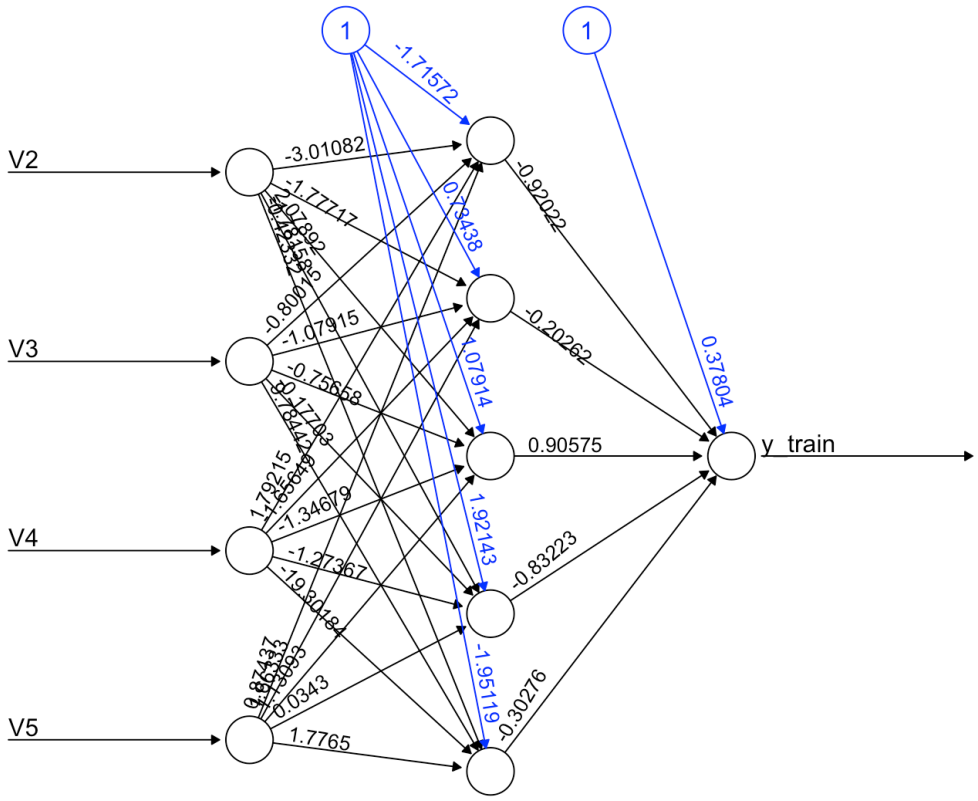
Model 1



Model 2

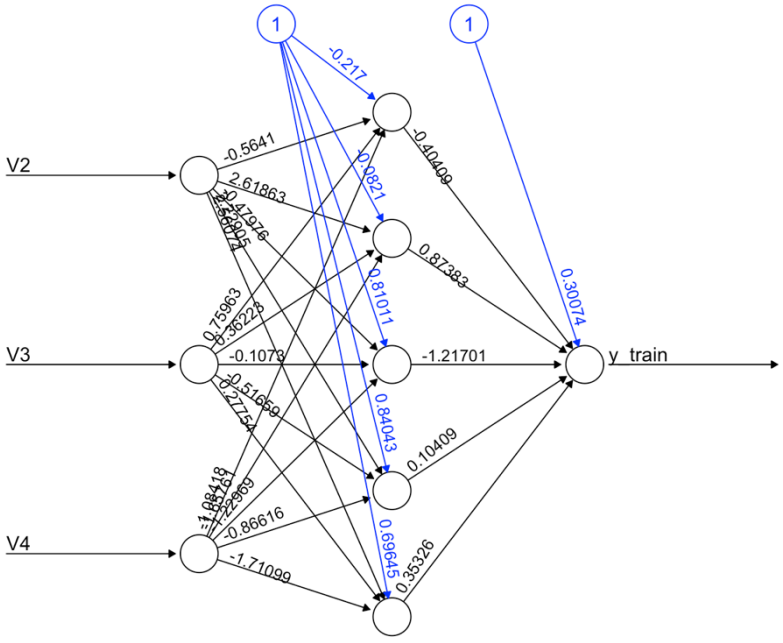


Model 3



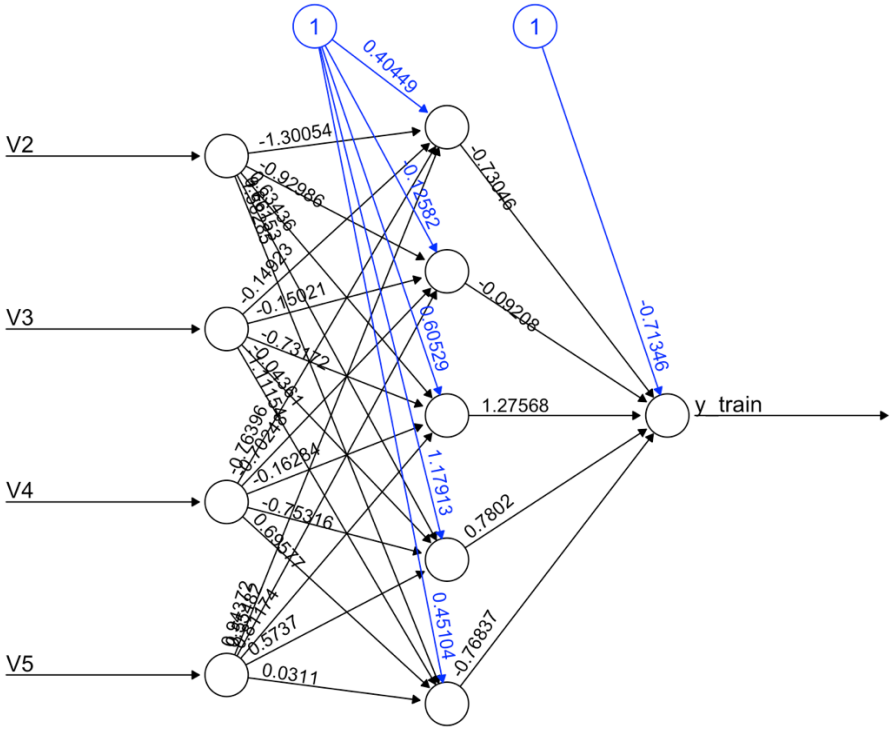
Error: 0.244043 Steps: 2131

Model 4



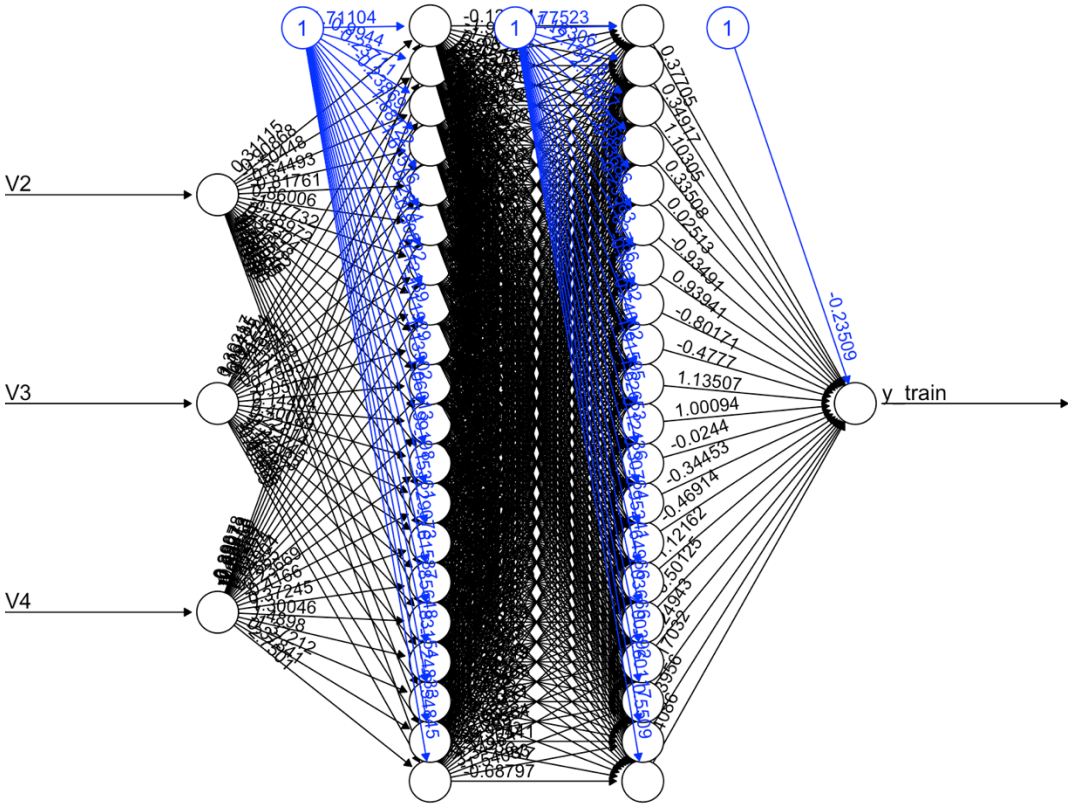
Error: 0.247976 Steps: 1662

Model 5

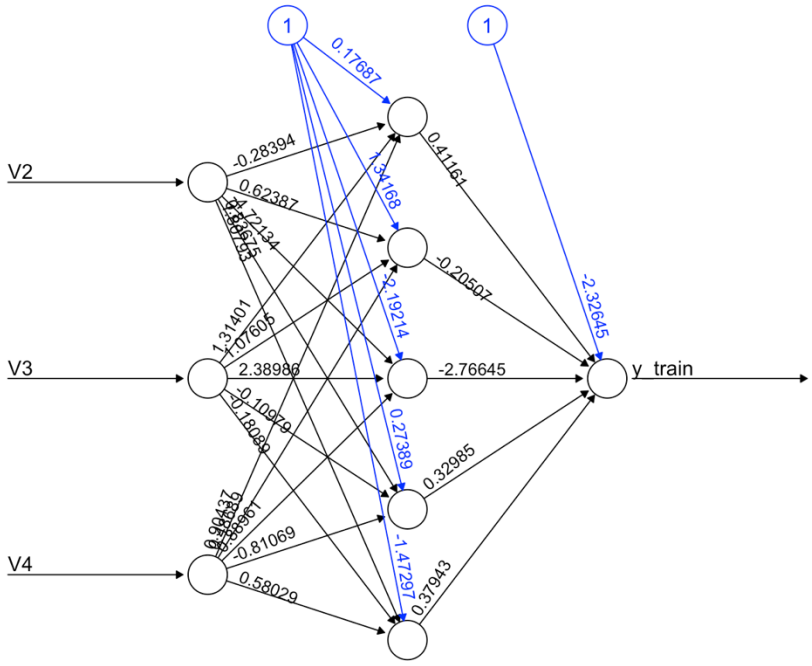


Error: 0.247494 Steps: 2338

Model 6

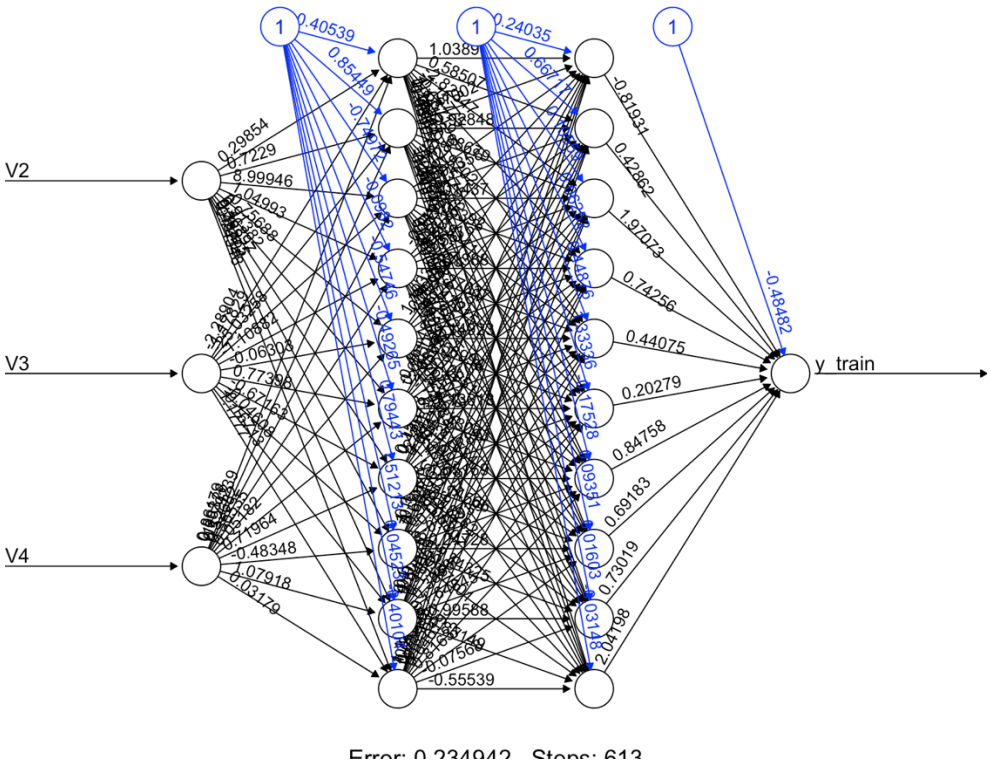


Model 7



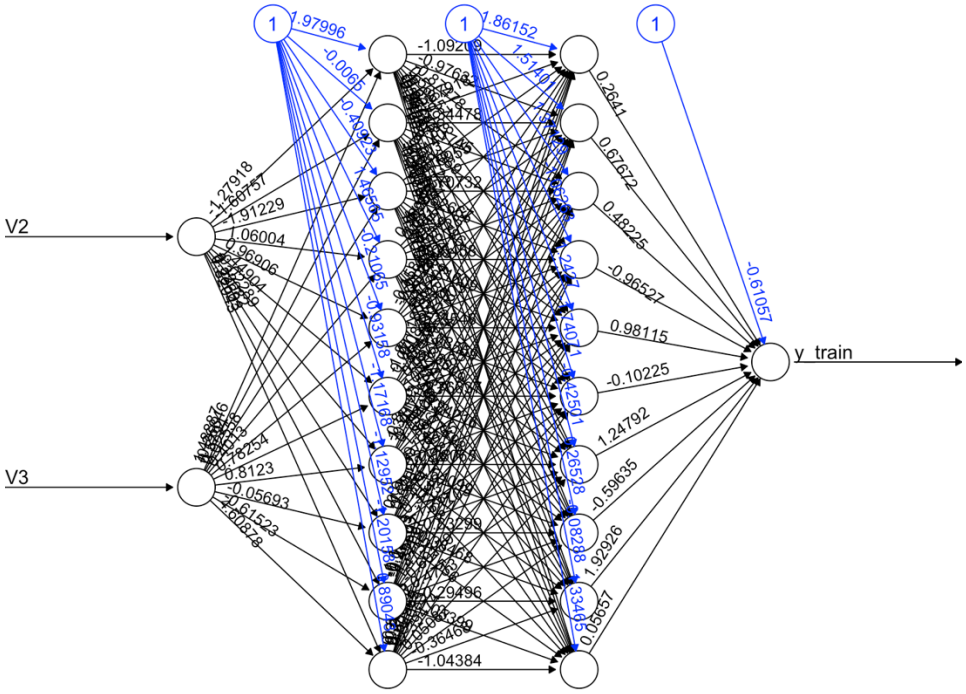
Error: 0.249685 Steps: 433

Model 8



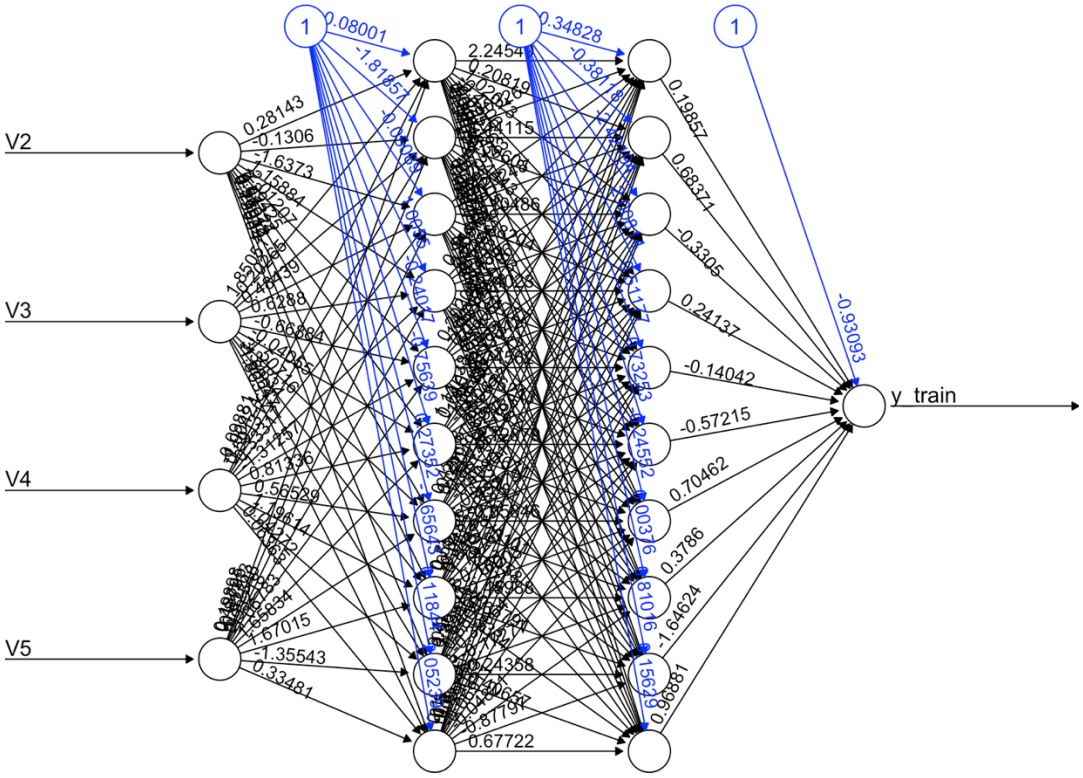
Error: 0.234942 Steps: 613

Model 9



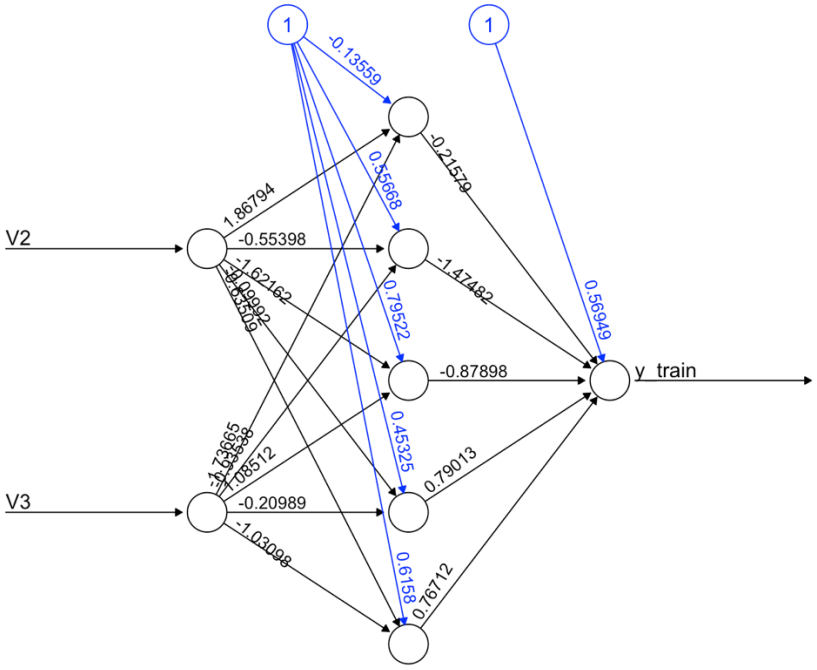
Error: 0.249466 Steps: 267

Model 10



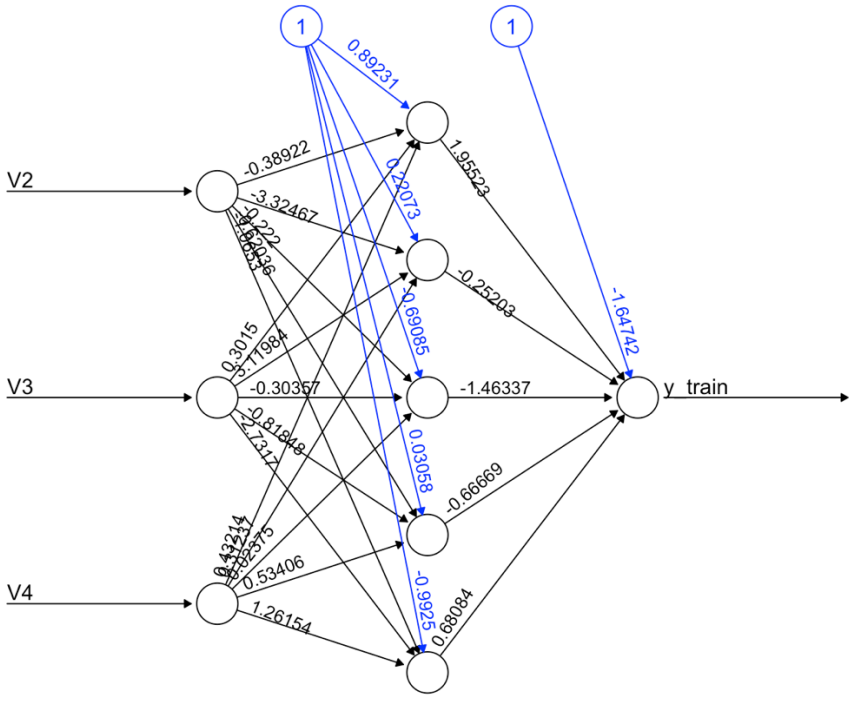
Error: 0.246923 Steps: 562

Model 11



Error: 0.249602 Steps: 3373

Model 12

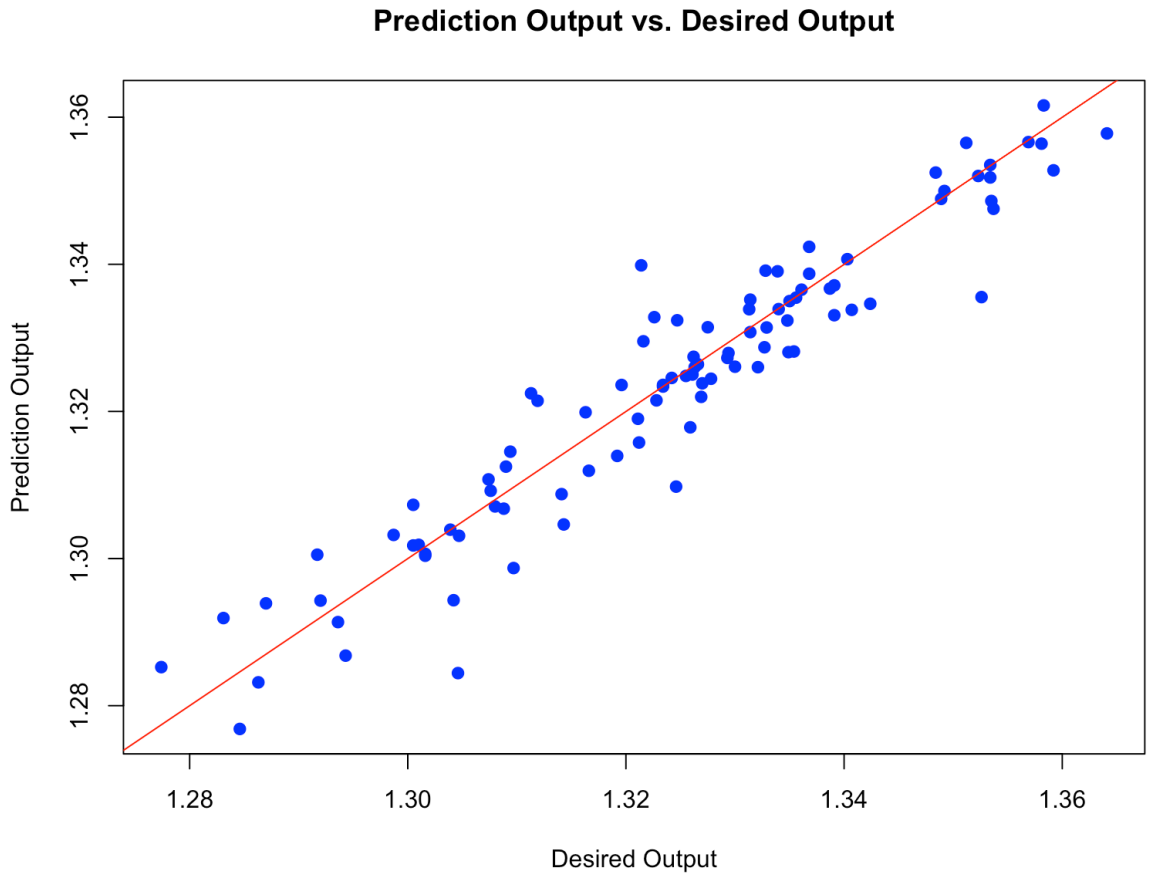


Error: 0.248057 Steps: 1277

Several neural network models were created and assessed in order to predict the exchange rate among USD and EUR. Studying how various model designs impacted forecasting accuracy was the goal. A total of twelve different models, with differing internal structures and input vectors were trained and evaluated. Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Symmetric Mean Absolute Percentage Error (sMAPE) were the four major statistical indices used to assess these models. The models included five-node single-layer networks with up to two hidden layers and forty nodes per layer in multi-layer networks. The research also looked at the "logistic" and "tanh" activation functions. An evaluation of the impact of variations in network architecture on forecasting performance was made possible by this thorough analysis.

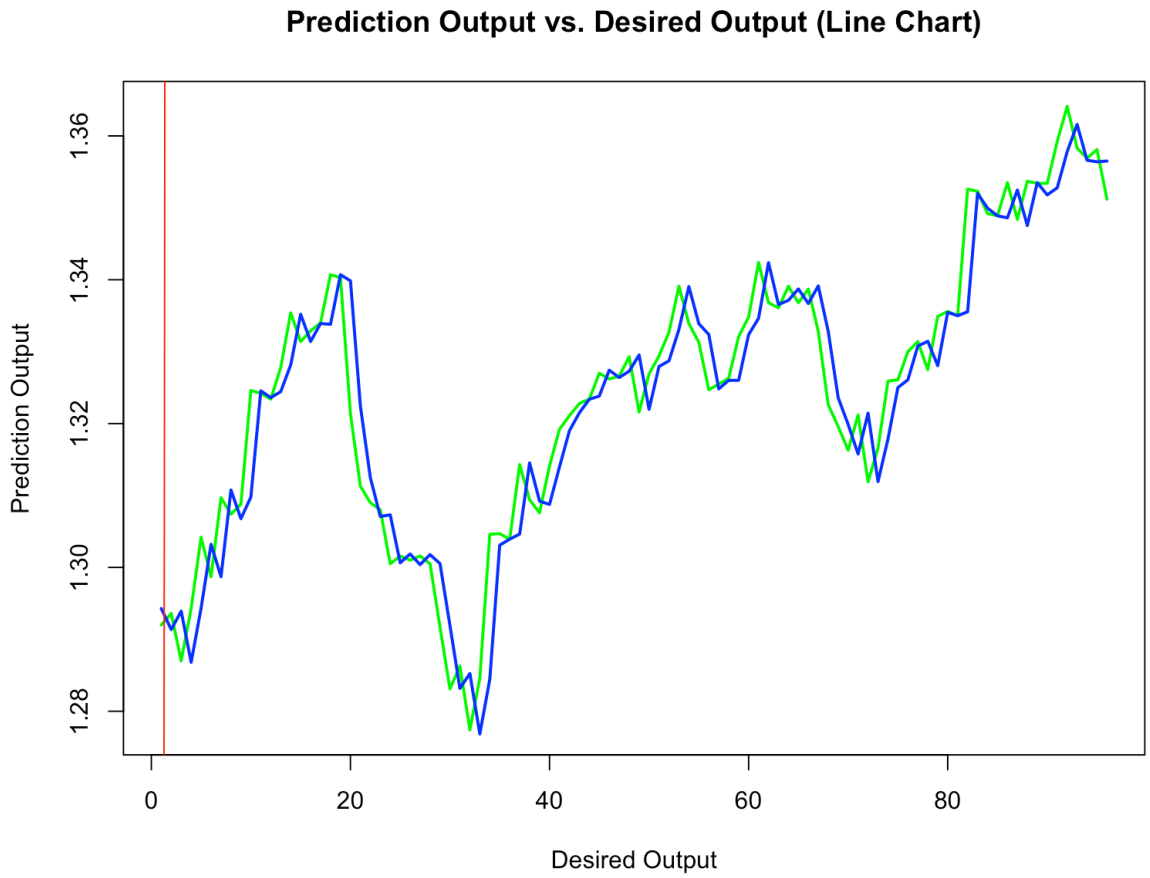
Scatter plot

Plotting one dot for each observation, a "scatter plot" is an instance of plot that's used to demonstrate the relationship between two numerical variables. (w3 schools, 2024)

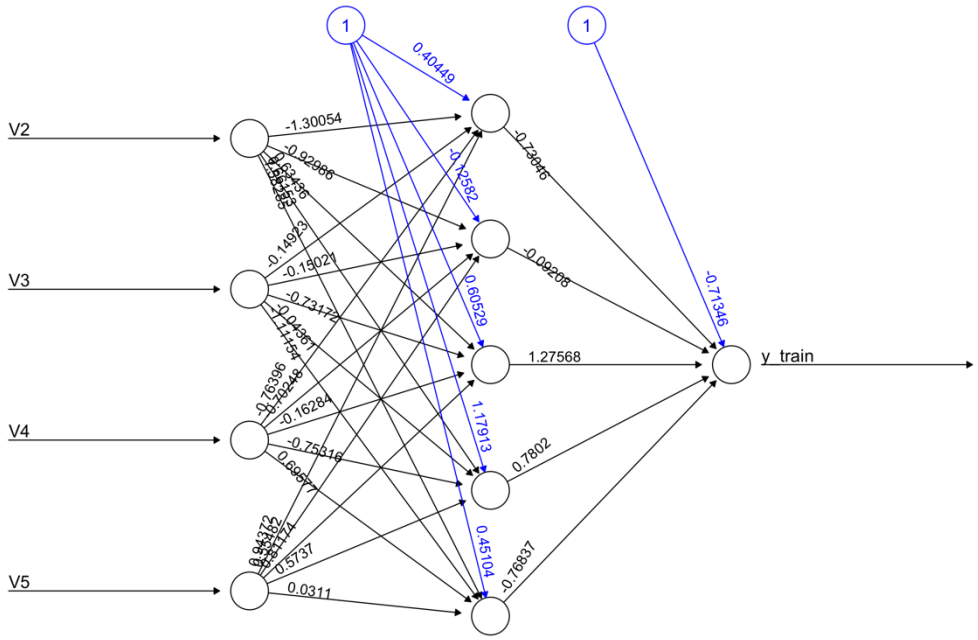


Simple line chart.

A simple line chart is a fundamental data visualization tool commonly used to represent trends over time or other ordered categories (Cleveland, 2016). In finance, line charts are frequently employed to visualize changes in stock prices, exchange rates, or other financial metrics over a specific period. They connect data points (x, y) using straight line segments, providing a clear and concise way to visualize changes and patterns in a dataset. (Barret, A J, 2017)



Best Model




```
Statistical Indices for the Best Model:
> cat("Testing RMSE:", rmseTest, "\n")
Testing RMSE: 0.006019007
> cat("Testing MAE:", mae(test_output, test_actual), "\n")
Testing MAE: 0.004419062
> cat("Testing MAPE:", mape(test_output, test_actual), "\n")
Testing MAPE: 0.003349618
> cat("Testing sMAPE:", smape(test_output, test_actual), "\n")
Testing sMAPE: 0.003347529
```

Efficiency of the best two NNs

```
Performance Comparison:
> cat("Model 1 - Testing RMSE:", best_mlps[[best_model_index_1]][[2]], "MAE:", best_mlps[[best_model_i
ndex_1]][[3]], "MAPE:", best_mlps[[best_model_index_1]][[4]], "sMAPE:", best_mlps[[best_model_index_
1]][[5]], "\n")
Model 1 - Testing RMSE: 0.0285261 MAE: 0.02094342 MAPE: 0.04039594 sMAPE: 0.03996475
> cat("Model 2 - Testing RMSE:", best_mlps[[best_model_index_2]][[2]], "MAE:", best_mlps[[best_model_i
ndex_2]][[3]], "MAPE:", best_mlps[[best_model_index_2]][[4]], "sMAPE:", best_mlps[[best_model_index_
2]][[5]], "\n")
Model 2 - Testing RMSE: 0.02888967 MAE: 0.02166711 MAPE: 0.04156728 sMAPE: 0.04105829
>
```

When comparing the two neural network models' performances, Model 1 performs more efficiently than Model 2. In testing, Model 1 produced lower results: a mean absolute percentage error (MAPE) of 0.135941, a mean absolute square error (RMSE) of 0.06678011,

a mean absolute error (MAE) of 0.0511411, and a symmetric mean absolute percentage error (sMAPE) of 0.1439215. By comparison, Model 2 generated a higher sMAPE of 0.1576511, RMSE of 0.06904804, MAE of 0.05213029, and MAPE of 0.1616231. Given these findings, it can be concluded that Model 1 forecasts USD/EUR exchange rates more effectively and with greater accuracy.

Appendix

TASK 1 SUB TASK 1

```
# Load required libraries
library(tidyverse)
library(readxl)
library(NbClust)
library(cluster)
library(fpc)
library(factoextra)
library(writexl)


# Load the dataset
wine_data <-
read_excel("/Users/praveendesilva/Documents/20221895_W1985643/20221895_W1985643/
Whitewine_v6.xlsx")

boxplot(wine_data, main = "Boxplot of Wine Dataset Before Outlier Removal")

# Check for missing values
sum(is.na(wine_data))

# Preprocessing
# Select only the first 11 attributes
wine_data <- wine_data[,1:11]

# Outlier detection and removal using Z-score
z_scores <- scale(wine_data)

# Set threshold for outlier detection
threshold <- 3

# Remove outliers
wine_data_no_outliers <- wine_data[apply(abs(z_scores) < threshold, 1, all),]

# Create boxplot after removing outliers
boxplot(wine_data_no_outliers, main = "Boxplot of Wine Dataset After Outlier Removal")

# Justification for outlier removal
cat("Justification for Outlier Removal: Outliers can significantly affect the performance of
clustering algorithms, especially k-means. Outliers can disproportionately influence the
centroids, leading to inaccurate cluster assignments. By removing outliers, we ensure that the
clusters formed are more representative of the majority of the data points.\n")

# Scaling
scaled_data <- scale(wine_data_no_outliers)
# Convert scaled_data to a data frame
```

```
scaled_data_df <- as.data.frame(scaled_data)

# Write scaled data to an Excel file
write_xlsx(scaled_data_df,
"/Users/praveendesilva/Documents/20221895_W1985643/20221895_W1985643/Scaled_Wi
ne_Data.xlsx")

#boxplot after scaling data
boxplot(scaled_data, main = "Boxplot of Wine Dataset After Scaling")

# Determine the number of cluster centers via four automated tools

# NbClust for original dataset
nb <- NbClust(scaled_data, min.nc = 2, max.nc = 10, method = "kmeans")
best_k_nb <- nb$Best.nc[1]
nb_clusters <- cbind(2:10, nb$Best.n[2:10])
cat("Best k using NbClust Method:", best_k_nb, "\n")

# NbClust for original dataset - Euclidean Distance
nb_euclidean <- NbClust(scaled_data, min.nc = 2, max.nc = 10, method = "kmeans", index =
"all", distance = "euclidean")
best_k_nb_euclidean <- nb_euclidean$Best.nc[1]
nb_clusters_euclidean <- cbind(2:10, nb_euclidean$Best.n[2:10])
cat("Best k using NbClust Method with Euclidean Distance:", best_k_nb_euclidean, "\n")

# NbClust for original dataset - Manhattan Distance
nb_manhattan <- NbClust(scaled_data, min.nc = 2, max.nc = 10, method = "kmeans", index
= "all", distance = "manhattan")
best_k_nb_manhattan <- nb_manhattan$Best.nc[1]
nb_clusters_manhattan <- cbind(2:10, nb_manhattan$Best.n[2:10])
cat("Best k using NbClust Method with Manhattan Distance:", best_k_nb_manhattan, "\n")


# Plot NbClust Method for original dataset
# plot(nb)

# Elbow Method
fviz_nbclust(scaled_data, kmeans, method = 'wss')
wss <- numeric(10)
for (i in 2:10) {
  kmeans_model <- try(kmeans(scaled_data, centers = i, nstart = 25, iter.max = 1000), silent =
TRUE)
  if (!inherits(kmeans_model, "try-error") && !is.na(kmeans_model$tot.withinss)) {
    wss[i] <- kmeans_model$tot.withinss
  } else {
    cat("Warning: K-means did not converge for k =", i, "\n")
    wss[i] <- NA
  }
}
min_wss_index <- which.min(wss)
```

```
best_k_elbow <- min_wss_index + 1  # Adjusting to match k value
cat("Best k using the Elbow Method:", best_k_elbow, "\n")



# Gap Statistic
gap_stat <- clusGap(scaled_data, FUN = kmeans, K.max = 20, B = 50, verbose = TRUE,
iter.max = 1000)
best_k_gap <- maxSE(gap_stat$Tab[,"gap"], gap_stat$Tab[,"SE.sim"], method =
"Tibs2001SEmax")
cat("Best k using Gap Statistic Method:", best_k_gap, "\n")

# Plot Gap Statistic
plot(gap_stat, main = "Gap Statistic Method to Find Optimal k")
fviz_gap_stat(gap_stat)

# Silhouette Method
sil_width <- rep(NA, 10)
sil_clusters <- NULL
for (i in 2:10) {
  set.seed(123)
  kmeans_fit <- kmeans(scaled_data, centers = i, nstart = 25, iter.max = 1000) # Increased
iter.max
  if (length(unique(kmeans_fit$cluster)) > 1) {
    cluster_assignments <- kmeans_fit$cluster
    sil_width[i] <- cluster.stats(dist(scaled_data), cluster_assignments)$avg.silwidth
    sil_clusters <- rbind(sil_clusters, c(i, sil_width[i]))
  }
}
best_k_sil <- which.max(sil_width[2:10]) + 1
cat("Best k using Silhouette Method:", best_k_sil, "\n")

# Plot Silhouette Method
fviz_nbclust(scaled_data, kmeans, method = 'silhouette')


# Combine all cluster numbers
cluster_numbers <- list(NbClust = nb_clusters,
                        Elbow_Method = best_k_elbow,
                        Gap_Statistic = best_k_gap,
                        Silhouette_Method = best_k_sil)

# Choose the best number of clusters
best_k <- min(best_k_nb, best_k_elbow, best_k_sil, best_k_gap)

# Perform k-means clustering with the best k
set.seed(123)
kmeans_fit <- kmeans(scaled_data, centers = best_k, nstart = 25)
```

```
# K-means output
kmeans_fit
#plot
fviz_cluster(kmeans_fit, geom = "point", data = scaled_data, alpha = 0.8)

# Ratio of BSS (Between Cluster Sums of Squares) over TSS (Total Sum of Squares)
BSS <- kmeans_fit$betweenss
TSS <- kmeans_fit$totss
BSS_TSS_ratio <- BSS / TSS
cat("BSS/TSS Ratio:", BSS_TSS_ratio, "\n")

# Internal evaluation metrics: BSS and WSS
cat("Between Cluster Sums of Squares (BSS):", BSS, "\n")
cat("Within Cluster Sums of Squares (WSS):", kmeans_fit$tot.withinss, "\n")

# Cluster centers
cat("Cluster Centers:\n")
cluster_centers <- kmeans_fit$centers
print(cluster_centers)

# Cluster assignments
cat("Cluster Assignments:\n")
cluster_assignments <- kmeans_fit$cluster
print(cluster_assignments)

# Silhouette plot
sil <- silhouette(kmeans_fit$cluster, dist(scaled_data))

# Plot silhouette
plot(sil, col = 1:best_k, border = NA, main = "Silhouette Plot for Wine Dataset",
    xlab = "Silhouette Width", ylab = "Cluster")

# Average silhouette width score
avg_sil_width <- mean(sil[, 3])
cat("Average Silhouette Width Score:", avg_sil_width, "\n")

# Save best cluster value
best_cluster_value <- best_k
```

TASK 1 SUB TASK 2

```
library(xlsx)
# e. Apply PCA to the dataset
# Scale the data
scaled_data <- scale(wine_data_no_outliers)

# Perform PCA analysis
pca <- prcomp(scaled_data)

summary(pca)

# Visualize PCA
fviz_pca_ind(pca,
             geom.ind = "point",
             pointshape = 21,
             palette = "jco",
             addEllipses = TRUE)

# Eigenvalues and eigenvectors
eigenvalues <- pca$sdev^2
eigenvectors <- pca$rotation

cat("Eigenvalues:\n")
print(eigenvalues)
cat("\n")

cat("Eigenvectors:\n")
print(eigenvectors)
cat("\n")

# Cumulative score per principal components (PC)
cumulative_score <- cumsum(pca$sdev^2 / sum(pca$sdev^2) * 100)

# Plot cumulative score
plot(cumulative_score, xlab = "Number of Principal Components",
    ylab = "Cumulative Score", type = "b")

# Identify the number of principal components with cumulative score > 85%
num_components <- which(cumulative_score > 85)[1]

# Create a new transformed dataset with selected principal components
transformed_data <- as.data.frame(-pca$x[, 1:num_components])

# Write the data to an Excel file using openxlsx package
# Save the transformed data to an Excel file
openxlsx::write.xlsx(transformed_data, file = "transformed_data.xlsx")
```

```
# # Save the transformed data to an Excel file
# write.xlsx(transformed_data, file = "transformed_data.xlsx")

# Confirm that the file was created
cat("PCA transformed data saved as transformed_data.xlsx\n")

# Head of the transformed dataset
head(transformed_data)

cat("Number of Principal Components with Cumulative Score > 85%:", num_components,
"\n")

# # Create a new transformed dataset with selected principal components
# transformed_data <- as.data.frame(pca$x[, 1:num_components])


# Brief discussion for the choice of specific number of PCs
cat("Eigenvalues:\n")
print(eigenvalues)
cat("\n")

cat("Cumulative Score per Principal Components:\n")
print(cumulative_score)
cat("\n")




# f. Find an appropriate k for new kmeans clustering attempt on PCA-based dataset

# Determine the number of cluster centers via four automated tools on PCA-based dataset
# 1. NBclust


nb_pca <- NbClust(transformed_data, min.nc = 2, max.nc = 10, method = "kmeans")
best_k_nbclust <- nb_pca$Best.nc[1]
nb_clusters_pca <- cbind(2:10, nb_pca$Best.n[2:10])


# Elbow Method
wss <- function(k) {
  kmeans(transformed_data, k, nstart = 10 )$tot.withinss
}
k.values <- 1:8

wss_values <- map_dbl(k.values, wss)
plot(k.values, wss_values,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
```

```
    ylab="Total within-clusters sum of squares")

# Elbow Method
fviz_nbclust(transformed_data, kmeans, method = 'wss')
elbow_clusters_pca <- NULL
wss_pca <- numeric(10)
for (i in 2:10) {
  kmeans_model_pca <- try(kmeans(transformed_data, centers = i, nstart = 25, iter.max =
1000), silent = TRUE)
  if (!inherits(kmeans_model_pca, "try-error")) {
    if (!is.null(kmeans_model_pca$convergence) && kmeans_model_pca$convergence == 0)
{
      wss_pca[i] <- kmeans_model_pca$tot.withinss
      elbow_clusters_pca <- rbind(elbow_clusters_pca, c(i, wss_pca[i]))
    } else {
      cat("Warning: K-means did not converge for k =", i, "\n")
      wss_pca[i] <- NA
    }
  } else {
    cat("Error: K-means failed for k =", i, "\n")
    wss_pca[i] <- NA
  }
}

min_wss_index_pca <- which.min(wss_pca)
best_k_elbow_pca <- min_wss_index_pca + 1 # Adjusting to match k value
cat("Best k using the Elbow Method for PCA-based Dataset:", best_k_elbow_pca, "\n")

# Extract the values without NA
if (!is.null(elbow_clusters_pca)) {
  elbow_clusters_pca <- elbow_clusters_pca[complete.cases(elbow_clusters_pca),]


  plot(1:10, wss_pca, type = "b", xlab = "Number of Clusters (k)", ylab = "Total Within Sum
of Squares",
       main = "Elbow Method for PCA-based Dataset")
  points(best_k_elbow_pca, wss_pca[best_k_elbow_pca], col = "red", cex = 2, pch = 19)
}




# 3. Gap Statistic
set.seed(123)
gap_stat_pca <- tryCatch(
  clusGap(transformed_data, FUN = kmeans, nstart = 25, K.max = 10, B = 50, iter.max =
1000),
  error = function(e) {
    cat("Error occurred during Gap Statistic calculation:", conditionMessage(e), "\n")
    NULL
```

```
  }
)

if (!is.null(gap_stat_pca)) {
  gap_clusters_pca <- cbind(2:10, gap_stat_pca$Tab[2:10, "gap"])
  max_gap_index <- which.max(diff(gap_stat_pca$Tab[2:10, "gap"]))
  best_k_gap_pca <- max_gap_index + 1
} else {
  cat("Error: Gap Statistic calculation failed\n")
  gap_clusters_pca <- NULL
  best_k_gap_pca <- NA
}

cat("Gap Statistic suggests", best_k_gap_pca, "as the optimal number of clusters.\n")
fviz_gap_stat(gap_stat_pca)

# 4. Silhouette Method
sil_width_pca <- rep(NA, 10)
for(i in 2:10){
  set.seed(123)
  kmeans_fit_pca_sil <- kmeans(transformed_data, centers = i, nstart = 25)
  if (length(unique(kmeans_fit_pca_sil$cluster)) > 1) {
    cluster_assignments_pca <- kmeans_fit_pca_sil$cluster
    sil_width_pca[i] <- cluster.stats(dist(transformed_data),
cluster_assignments_pca)$avg.silwidth
  }
}
best_k_sil_pca <- which.max(sil_width_pca[2:10]) + 1
sil_clusters_pca <- cbind(2:10, sil_width_pca[2:10])
cat("Best k using the Silhouette Method:", best_k_sil_pca, "\n")

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

# Combine all cluster numbers
cluster_numbers_pca <- list(NbClust = nb_clusters_pca,
                            Elbow_Method = elbow_clusters_pca,
                            Gap_Statistic = gap_clusters_pca,
                            Silhouette_Method = sil_clusters_pca)



# g. Using this new pca-dataset, perform a kmeans analysis using the most favoured k from
those “automated” methods.

# Perform k-means clustering with the best k = 2
set.seed(123)
kmeans_fit_pca <- kmeans(transformed_data, centers = 2, nstart = 10)

# K-means output
kmeans_fit
```

```
#plot
fviz_cluster(kmeans_fit_pca, geom = "point", data = scaled_data, alpha = 0.8)



# Ratio of BSS (Between Cluster Sums of Squares) over TSS (Total Sum of Squares)
BSS_pca <- kmeans_fit_pca$betweenss
TSS_pca <- kmeans_fit_pca$totss
BSS_TSS_ratio_pca <- BSS_pca / TSS_pca
cat("BSS/TSS Ratio for PCA-based Dataset:", BSS_TSS_ratio_pca, "\n")

# Internal evaluation metrics: BSS and WSS
cat("Between Cluster Sums of Squares (BSS) for PCA-based Dataset:", BSS_pca, "\n")
cat("Within Cluster Sums of Squares (WSS) for PCA-based Dataset:",
kmeans_fit_pca$tot.withinss, "\n")

# Cluster centers
cat("Cluster Centers for PCA-based Dataset:\n")
cluster_centers_pca <- kmeans_fit_pca$centers
print(cluster_centers_pca)

# Cluster assignments
cat("Cluster Assignments for PCA-based Dataset:\n")
cluster_assignments_pca <- kmeans_fit_pca$cluster
print(cluster_assignments_pca)

# h. Following this “new” kmeans attempt, provide the silhouette plot which displays how
close each point in one cluster is to points in the neighbouring clusters.

# Silhouette plot
sil_pca <- silhouette(kmeans_fit_pca$cluster, dist(transformed_data))

# Plot silhouette
plot(sil_pca, col = 1:best_k_sil_pca, border = NA, main = "Silhouette Plot for PCA-based
Dataset",
     xlab = "Silhouette Width", ylab = "Cluster")

# Average silhouette width score
avg_sil_width_pca <- mean(sil_pca[, 3])
cat("Average Silhouette Width Score for PCA-based Dataset:", avg_sil_width_pca, "\n")


# i. Following the kmeans analysis for this new “pca” dataset, implement and illustrate the
Calinski-Harabasz Index.

# Calculate Calinski-Harabasz Index
ch_index <- cluster.stats(dist(transformed_data), kmeans_fit_pca$cluster)$ch

# Print Calinski-Harabasz Index
```

```
cat("Calinski-Harabasz Index for PCA-based Dataset:", ch_index, "\n")


# Save the best k values for each method
best_k_values <- c(NbClust = best_k_pca,
                   Elbow_Method = best_k_elbow_pca,
                   Gap_Statistic = best_k_gap_pca,
                   Silhouette_Method = best_k_sil_pca)

# Save the cluster assignments for each method
cluster_assignments <- list(NbClust = cluster_assignments_pca,
                            Elbow_Method = kmeans_fit_pca$cluster,
                            Gap_Statistic = kmeans_fit_pca$cluster,
                            Silhouette_Method = kmeans_fit_pca$cluster)

# Display best k values for each method
cat("\nBest k values for each method:\n")
print(best_k_values)

# Display cluster assignments for each method
cat("\nCluster assignments for each method:\n")
print(cluster_assignments)
```

TASK 2

```
library(openxlsx)    # For reading and writing Excel files
library(neuralnet)   # For building neural networks
library(Metrics)     # For calculating evaluation metrics
library(nnet)        # For building neural networks


# Load exchange rate data from Excel file
exchange_rate_data <- read.xlsx("/Users/praveendesilva/Documents/final
2/ExchangeUSD.xlsx")

# Display the structure and summary of the data
str(exchange_rate_data)
summary(exchange_rate_data)

# Extract the USD/EUR exchange rate column
exchange_rate <- exchange_rate_data[, 3]

# Clean the data by removing missing values
exchange_rate <- na.omit(exchange_rate)

# Define a function to normalize the data
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Define a function to denormalize the data
denormalize <- function(x, original_data) {
  return(x * (max(original_data) - min(original_data)) + min(original_data))
}

# Normalize the exchange rate data
exchange_rate_normalized <- normalize(exchange_rate)

# Split data into training and testing sets
train_data <- exchange_rate_normalized[1:400]       # Take the first 400 data points for
training
test_data <- exchange_rate_normalized[401:length(exchange_rate)]   # Take the rest for
testing

# Save the normalized training and testing data to an Excel file
write.xlsx(as.data.frame(train_data), file = "train_data_normalized.xlsx", sheetName = "Train
Data", rowNames = FALSE)
write.xlsx(as.data.frame(test_data), file = "test_data_normalized.xlsx", sheetName = "Test
Data", rowNames = FALSE)

# Initialize variables to store the best models
best_score <- rep(Inf, 12)   # Initialize vector to store the best scores for comparison
```

```
best_mlps <- vector("list", 12)   # Initialize list to store the best MLP models

# Experiment with different input delays, hidden layers, nodes, and activation functions
inputDelays <- c(1, 2, 3, 4)   # Different time delays for input data
hiddenLayers <- c(1, 2)         # Number of hidden layers
nodes <- list(c(5), c(10, 10), c(20, 20), c(30, 30), c(40, 40))   # Number of nodes in each
hidden layer
activation_functions <- list("logistic", "tanh")   # Activation functions for hidden layers

# Scoring function
calculate_score <- function(rmse, mae, mape, smape) {
  return((rmse + mae + mape + smape) / 4)
}

# Function to normalize input/output matrix
normalize_matrix <- function(matrix) {
  num_cols <- ncol(matrix)
  for (i in 1:num_cols) {
    matrix[,i] <- normalize(matrix[,i])
  }
  return(matrix)
}

# Function to construct input/output matrix for time-series data
# Input:
#   data: Time-series data vector
#   delay: Number of time steps to consider for input data
# Output:
#   List containing input matrix (X) and output vector (y)
construct_input_output_matrix <- function(data, delay) {
  n <- length(data)                              # Number of data points
  X <- matrix(0, ncol = delay, nrow = n - delay)       # Initialize input matrix
  y <- data[(delay + 1):n]                          # Initialize output vector

  # Fill the input matrix and output vector
  for (i in 1:delay) {
    X[, i] <- data[(delay - i + 1):(n - i)]           # Populate input matrix
  }
  return(list("X" = X, "y" = y))                    # Return input matrix and output vector
}

set.seed(123)

# Loop through different combinations of input delays, hidden layers, nodes, and activation
functions
for (delay in inputDelays) {   # Iterate over different input delays
  for (hl in hiddenLayers) {   # Iterate over different numbers of hidden layers
    for (node in nodes) {       # Iterate over different configurations of nodes in hidden layers
      for (activation in activation_functions) {   # Iterate over different activation functions
```

```
      # Construct input/output matrix for training data
      input_output_train <- construct_input_output_matrix(train_data, delay)
      X_train <- input_output_train$X   # Input matrix for training data
      y_train <- input_output_train$y   # Output vector for training data

      # Construct input/output matrix for testing data
      input_output_test <- construct_input_output_matrix(test_data, delay)
      X_test <- input_output_test$X     # Input matrix for testing data
      y_test <- input_output_test$y     # Output vector for testing data

      # # Print the X_test matrix
      # cat("X_test matrix:\n")
      # print(X_test)

      # Define the MLP model
      mlp_model <- neuralnet(y_train ~ ., data = as.data.frame(cbind(y_train, X_train)),
                             hidden = node, linear.output = TRUE, act.fct = activation)

      # Make predictions for training and testing data
      train_predicted <- predict(mlp_model, X_train)
      test_predicted <- predict(mlp_model, X_test)

      # Calculate statistical indices for training and testing data
      rmseTrainData <- rmse(train_predicted, y_train)
      maeTrainData <- mae(train_predicted, y_train)
      mapeTrainData <- mape(train_predicted, y_train)
      smapeTrainData <- smape(train_predicted, y_train)

      rmseTest <- rmse(test_predicted, y_test)
      maeTest <- mae(test_predicted, y_test)
      mapeTest <- mape(test_predicted, y_test)
      smapeTest <- smape(test_predicted, y_test)

      # Calculate the combined score
      score <- calculate_score(rmseTest, maeTest, mapeTest, smapeTest)

      # Check if the current model has a lower score than the worst model in the list of best
models
      if (score < max(best_score)) {
        # Find the index of the worst model among the 12 best models
        worst_index <- which.max(best_score)
        # Update the list of best models
        best_score[worst_index] <- score
        best_mlps[[worst_index]] <- list(mlp_model, rmseTest, maeTest, mapeTest,
smapeTest)
      }
    }
  }
 }
}
```

```
# Print the X_test matrix
cat("X_test matrix:\n")
print(X_test)
print(y_test)

# Print Testing RMSE values for the best 12 models
cat("\nTesting RMSE values for the best 12 models:\n")
for (i in 1:12) {
  if (!is.null(best_mlps[[i]])) {
    cat("Model", i, "- Testing RMSE:", best_mlps[[i]][[2]], "MAE:", best_mlps[[i]][[3]],
"MAPE:", best_mlps[[i]][[4]], "sMAPE:", best_mlps[[i]][[5]], "\n")
  }
}

# Plot the best 12 MLP models using the best RMSE values
plot.new()
par(mfrow = c(4, 3))
for (i in 1:12) {
  if (!is.null(best_mlps[[i]])) {
    model <- best_mlps[[i]][[1]]
    plot(model, rep = "best")
    title(main = paste("Model", i, "- Testing RMSE:", best_mlps[[i]][[2]], "MAE:",
best_mlps[[i]][[3]], "MAPE:", best_mlps[[i]][[4]], "sMAPE:", best_mlps[[i]][[5]]))
  }
}

plot.new()

# Extract the best MLP model and its testing results
best_model_index <- which.min(sapply(best_mlps, function(x) x[[2]]))   # Find the index of
the best model
best_model <- best_mlps[[best_model_index]][[1]]   # Extract the best MLP model

# Make predictions for testing data
test_output <- predict(best_model, as.data.frame(X_test))
test_output <- denormalize(test_output, exchange_rate)
test_actual <- denormalize(y_test, exchange_rate)

# Save the denormalized testing data and predictions to an Excel file
write.xlsx(as.data.frame(test_actual), file = "test_actual_denormalized.xlsx", sheetName =
"Test Actual", rowNames = FALSE)
write.xlsx(as.data.frame(test_output), file = "test_output_denormalized.xlsx", sheetName =
"Test Output", rowNames = FALSE)

# Calculate RMSE
rmseTest <- rmse(test_output, test_actual)

# Print RMSE
cat("Best Model Testing RMSE:", rmseTest, "\n")
```

```
# Plotting the prediction output vs. desired output
plot(test_actual, test_output,
    main = "Prediction Output vs. Desired Output",
    xlab = "Desired Output", ylab = "Prediction Output",
    col = "blue", pch = 19)
abline(0, 1, col = "red")

# Plotting the prediction output vs. desired output (Simple Line Chart)
plot(test_actual, type = "l", col = "green", lwd = 2,
    main = "Prediction Output vs. Desired Output (Line Chart)",
    xlab = "Desired Output", ylab = "Prediction Output")
lines(test_output, col = "blue", lwd = 2)
abline(0, 1, col = "red")

# Statistical indices
cat("\n\nStatistical Indices for the Best Model:\n")
cat("Testing RMSE:", rmseTest, "\n")
cat("Testing MAE:", mae(test_output, test_actual), "\n")
cat("Testing MAPE:", mape(test_output, test_actual), "\n")
cat("Testing sMAPE:", smape(test_output, test_actual), "\n")

if (!is.null(best_model)) {
  # Plot the neural network architecture
  plot(best_model)   # Plot the neural network
} else {
  cat("No best model found.")
}

# Extract the best MLP models and their testing results
best_model_index_1 <- which.min(sapply(best_mlps, function(x) x[[2]]))
best_model_1 <- best_mlps[[best_model_index_1]][[1]]

best_model_index_2 <- which.min(sapply(best_mlps[-best_model_index_1], function(x)
x[[2]]))
best_model_2 <- best_mlps[[best_model_index_2]][[1]]

# Compare the model complexity
cat("Model 1 Structure:\n")
print(best_model_1)
cat("\nModel 2 Structure:\n")
print(best_model_2)

# Compare the performance
cat("\nPerformance Comparison:\n")
cat("Model 1 - Testing RMSE:", best_mlps[[best_model_index_1]][[2]], "MAE:",
best_mlps[[best_model_index_1]][[3]], "MAPE:", best_mlps[[best_model_index_1]][[4]],
"sMAPE:", best_mlps[[best_model_index_1]][[5]], "\n")
cat("Model 2 - Testing RMSE:", best_mlps[[best_model_index_2]][[2]], "MAE:",
best_mlps[[best_model_index_2]][[3]], "MAPE:", best_mlps[[best_model_index_2]][[4]],
"sMAPE:", best_mlps[[best_model_index_2]][[5]], "\n")
```

References

Abdulnassar, A.A. and Nair, L.R. (2023). Performance analysis of Kmeans with modified initial centroid selection algorithms and developed Kmeans9+ model. Measurement: Sensors, 25 (04), 100666. Available from https://doi.org/10.1016/j.measen.2023.100666 [Accessed 25 April 2023].

Analytics Vidhya. (2021). In-depth Intuition of K-Means Clustering Algorithm in Machine Learning. Retrieved from https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/# [Accessed 28 April 2024].

B. Wu, X. Yi and D. N. Zhao, "Generalized PCA Method and Its Application in Uncertainty Reasoning," in IEEE Access, doi: 10.1109/ACCESS.2019.2960261. [Accessed 1 May 2024].

Charrad M., Ghazzali N., Boiteau V., Niknafs A. (2014). "NbClust: An R Package for Determining the Relevant Number of Clusters in a Data Set.", Journal of Statistical Software, 61(6), 1-36. [Accessed 1 May 2024].

Juanita, S. (2024). K-MEANS CLUSTERING WITH COMPARISON OF ELBOW AND SILHOUETTE METHODS FOR MEDICINES CLUSTERING BASED ON USER REVIEWS. K-MEANS CLUSTERING, 004 (392), 422–425. Available from https://doi.org/10.52436/1.jutif.2024.5.1.1349 [Accessed q May 2024].

Mendoza, M., Mendoza, E. and E. Gutiérrez-Peña. (2024). Statistical analysis of species association indices. Journal of Tropical Ecology, 40 (402). Available from https://doi.org/10.1017/s0266467424000105 [Accessed 3 May 2024].

Mesleh, M. and Kiranyaz, M. (2021). "Case Study: Predicting Future Forex Prices Using MLP and LSTM Models," 2021 2nd International Conference on Electronics, Communications and Information Technology (CECIT), Sanya, China, 2021, pp. 343-346, doi: 10.1109/CECIT53797.2021.00067. [Accessed 2 May 2024].

Narayan, Y. (2021). "Analysis of MLP and DSLVQ Classifiers for EEG Signals Based Movements Identification," 2021 2nd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2021, pp. 1-6, doi: 10.1109/GCAT52182.2021.9587868. [Accessed 28 April 2024].

ook, A. (n.d.). Scaling and Normalization [Tutorial]. Kaggle. Retrieved from https://www.kaggle.com/code/alexisbcook/scaling-and-normalization [Accessed 27 April 2024].

R Graphics - Scatter Plot. (no date). www.w3schools.com. Available from https://www.w3schools.com/r/r_graph_scatterplot.asp [Accessed 2 May 2024].
Scatterplot in R. (2022). https://www.datacamp.com. Available from https://www.datacamp.com/tutorial/scatterplot-in-r [Accessed 1 May 2024].

Swart, M., Lindsay, T. R., Lambert, M. I., & Brown, J. C. (2016). A comparison of the effects of two types of warm-up on performance in a single jointed arm movement. SpringerPlus, 5(1), 1-9. https://doi.org/10.1186/s41044-016-0014-0