Informatics Institute of Technology

BEng (Hons) in Software Engineering

Machine Learning and Data Mining

Module Leader: Mr. Nipuna Senanayake

5DATA001C.2

Coursework: Report

Aaysha Fazal Mohamed

Uow Number – w1956175

IIT Number – 20221493

Group – C

1st Objective - Partitioning Clustering Part

Pre-processing tasks

Dataset

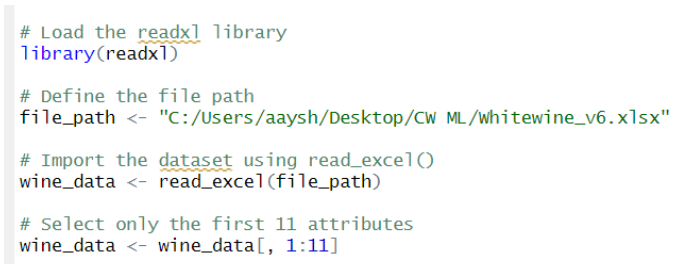
This project aims to cluster white wine varieties based on their chemical properties, excluding the sensory-based quality rankings provided by tasters. The dataset, named "whitewine_v6.xls," comprises 2700 wine varieties produced in a specific area of Portugal. The data include 12 attributes describing various chemical properties of the wines, with the output variable being the quality ranking from 1 to 10 based on sensory data.

Description of attributes (Used in the program):

1. Fixed acidity: Represents the non-volatile acids present in wine, contributing to its overall acidity level.
2. Volatile acidity: Indicates the amount of acetic acid in wine, affecting its taste and aroma.
3. Citric acid: Found in small quantities, adds freshness and flavor to wines.
4. Residual sugar: Quantifies the remaining sugar after fermentation, influencing the sweetness of the wine.
5. Chlorides: Measures the salt content in the wine.
6. Free sulfur dioxide: Helps prevent microbial growth and oxidation of wine.
7. Total sulfur dioxide: Represents the total amount of free and bound forms of sulfur dioxide.
8. Density: Reflects the density of wine, influenced by alcohol and sugar content.
9. pH: Describes the acidity or basicity of wine on a scale from 0 to 14.
10. Sulphates: Wine additive contributing to sulfur dioxide levels, acting as an antimicrobial and antioxidant.
11. Alcohol: Percentage of alcohol content in the wine.

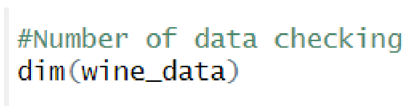
Output variable: Quality: Represents the sensory evaluation score of the wine, ranging from 0 to 10.

The chemical properties of white wine varieties are used for clustering analysis, with the aim of grouping them based on their attributes, excluding sensory data. The quality attribute, an ordinal variable ranked from 1 (worst) to 10 (best), is determined by the median rank from three independent tasters for each wine variety. However, this quality information cannot be used in the clustering process, as it is an unsupervised scheme.

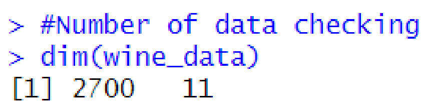


```
# Load the readxl library
library(readxl)

# Define the file path
file_path <- "C:/Users/aaysh/Desktop/CW ML/Whitewine_v6.xlsx"

# Import the dataset using read_excel()
wine_data <- read_excel(file_path)

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

This line of code imports the data from the Excel file into R and stores it in a data frame called **wine_data**, preparing it for further analysis and manipulation.



```
#Number of data checking
dim(wine_data)
```



```
> #Number of data checking
> dim(wine_data)
[1] 2700   11
```

The `wine_data <- wine_data[, 1:11]` line in R keeps only the first 11 columns of `wine_data`, focusing on key attributes. Running `dim(wine_data) ` after this will show the dataset now has 11 columns but retains its original row count, offering a snapshot of its new, streamlined structure. For example, if it started with 2700 rows, it now displays as 2700 rows by 11 columns.

Detection/Removal Outliers

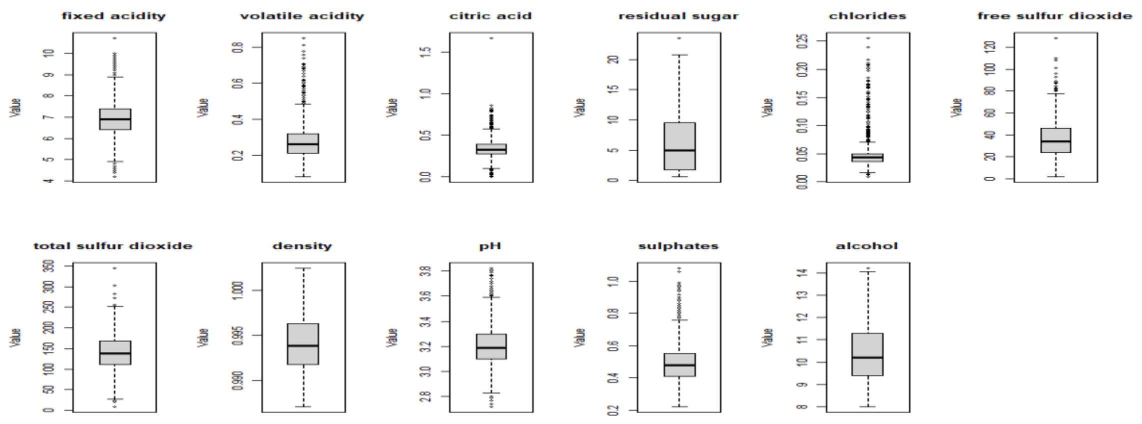
The first step of the machine learning project involves gathering data, which has already been provided for this project. The second step is data pre-processing, which involves performing several tasks before conducting k-means. These tasks include scaling, as well as detecting and removing outliers.

To ensure the reliability and accuracy of our dataset for further analysis, we perform outlier detection and removal. Outliers, as extreme and atypical observations, can distort visualizations, skew statistical analyses, and negatively impact predictive models. By systematically identifying and eliminating outliers from each attribute, we aim to enhance data representation accuracy, improve statistical analysis robustness, and facilitate more effective modelling. This process helps us gain more accurate insights into the underlying patterns and relationships within the data, ultimately leading to better decision-making and more reliable conclusions in our analyses.

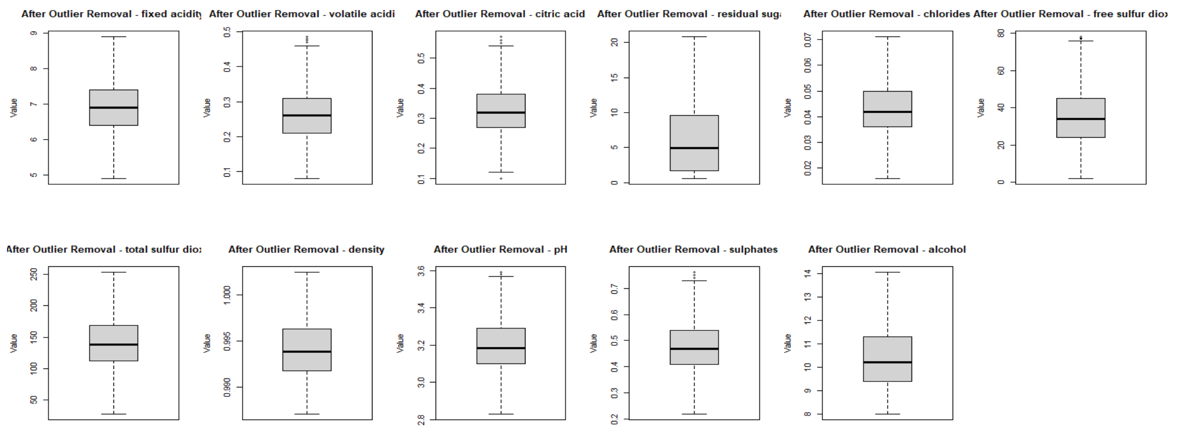
This code segment conducts outlier removal and visualization for a dataset named `wine_data`, focusing on white wine varieties and their chemical properties. It first generates boxplots for each attribute to visualize the data distribution before outlier removal. Then, it defines a function to remove outliers using the interquartile range method. Outliers are removed from each attribute, resulting in a cleaned dataset named `clean_data`. The code then plots boxplots again to visualize the cleaned data distribution. Finally, the cleaned data is consolidated into a data frame named `clean_df`. Overall, this code provides a clear before-and-after comparison of data distribution, aiding in understanding the impact of outlier removal on the dataset's characteristics.

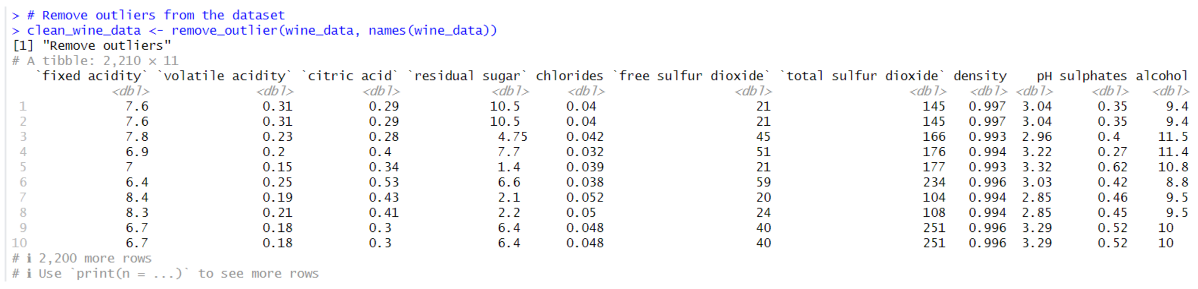
```r
# Set the layout to arrange the plots in two rows
par(mfrow = c(2, 6))  # Change the layout depending on the number of attributes

# Create boxplots for each attribute before removing outliers
for (col in names(wine_data)) {
  boxplot(wine_data[[col]], main = paste("Before Outlier Removal -", col), ylab = "Value")
}

# Define a function to remove outliers from each attribute
remove_outliers <- function(x) {
  qnt <- quantile(x, probs=c(.25, .75), na.rm = TRUE)
  H <- 1.5 * IQR(x, na.rm = TRUE)
  x[x < (qnt[1] - H) | x > (qnt[2] + H)] <- NA
  x
}

# Remove outliers from each attribute separately
clean_data <- sapply(wine_data, remove_outliers, simplify = FALSE)

# Create a new window to display boxplots after outlier removal
dev.new()

# Set the layout to arrange the plots in two rows
par(mfrow = c(2, 6))  # Change the layout depending on the number of attributes

# Create boxplots for each attribute after removing outliers
for (i in seq_along(clean_data)) {
  boxplot(clean_data[[i]], main = paste("After Outlier Removal -", names(clean_data)[i]), ylab = "Value")
}

# Combine the cleaned data into a data frame
clean_df <- as.data.frame(clean_data)
```

By executing a specific set of codes, it is possible to exclude the outliers from our dataset.

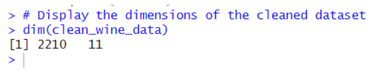


This exclusion results in a new dataset, renamed "**clean_data**". After removing the outliers, a boxplot of the dataset is presented below for further analysis.





```
> # Remove outliers from the dataset
> clean_wine_data <- remove_outlier(wine_data, names(wine_data))
[1] "Remove outliers"
# A tibble: 2,210 × 11
   `fixed acidity` `volatile acidity` `citric acid` `residual sugar` chlorides `free sulfur dioxide` `total sulfur dioxide` density    pH sulphates alcohol
             <dbl>              <dbl>         <dbl>            <dbl>     <dbl>                 <dbl>                  <dbl>   <dbl> <dbl>     <dbl>   <dbl>
 1             7.6               0.31          0.29            10.5      0.04                     21                    145   0.997  3.04      0.35     9.4
 2             7.6               0.31          0.29            10.5      0.04                     21                    145   0.997  3.04      0.35     9.4
 3             7.8               0.23          0.28             4.75     0.042                    45                    166   0.993  2.96      0.4     11.5
 4             6.9               0.2           0.4              7.7      0.032                    51                    176   0.994  3.22      0.27    11.4
 5             7                 0.15          0.34             1.4      0.039                    21                    177   0.993  3.32      0.62    10.8
 6             6.4               0.25          0.53             6.6      0.038                    59                    234   0.996  3.03      0.42     8.8
 7             8.4               0.19          0.43             2.1      0.052                    20                    104   0.994  2.85      0.46     9.5
 8             8.3               0.21          0.41             2.2      0.05                     24                    108   0.994  2.85      0.45     9.5
 9             6.7               0.18          0.3              6.4      0.048                    40                    251   0.996  3.29      0.52    10
10             6.7               0.18          0.3              6.4      0.048                    40                    251   0.996  3.29      0.52    10
# ℹ 2,200 more rows
# ℹ Use `print(n = ...)` to see more rows
```



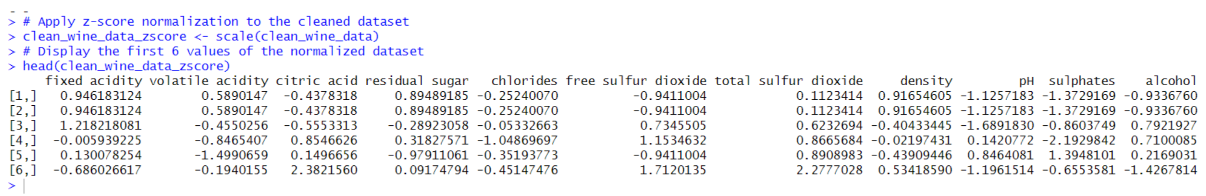
```
> # Display the dimensions of the cleaned dataset
> dim(clean_wine_data)
[1] 2210   11
> |
```

Data Normalization

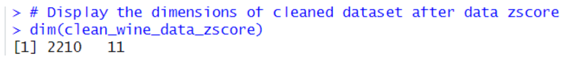
Then, the z-score method is used to do the scaling function. One of the most popular and efficient methods for scaling data is the z-score scaling approach, which is frequently regarded as the best method for several reasons.

One factor is the z-score scaling method's foundation in the data's mean and standard deviation. These common statistical techniques can help make the data easier to understand and compare across many datasets. The z-score scaling method also has the benefit of being resistant to outliers. Other scaling techniques, such as min-max scaling, which is dependent on the data range, might be significantly impacted by outliers.

The z-score method, on the other hand, is based on the mean and standard deviation, which are less susceptible to outliers and can aid in normalising the data even when there are outliers. Since the z-score scaling approach is also a linear transformation, the data distribution remains unaffected. This is crucial because it maintains the connections between the data points and can support preserving the data's statistical features.



```
> # Apply z-score normalization to the cleaned dataset
> clean_wine_data_zscore <- scale(clean_wine_data)
> # Display the first 6 values of the normalized dataset
> head(clean_wine_data_zscore)
     fixed acidity volatile acidity citric acid residual sugar   chlorides free sulfur dioxide total sulfur dioxide     density         pH  sulphates    alcohol
[1,]   0.946183124        0.5890147  -0.4378318     0.89489185 -0.25240070         -0.9411004            0.1123414  0.91654605 -1.1257183 -1.3729169 -0.9336760
[2,]   0.946183124        0.5890147  -0.4378318     0.89489185 -0.25240070         -0.9411004            0.1123414  0.91654605 -1.1257183 -1.3729169 -0.9336760
[3,]   1.218218081       -0.4550256  -0.5553313    -0.28923058 -0.05332663          0.7345505            0.6232694 -0.40433445 -1.6891830 -0.8603749  0.7921927
[4,]  -0.005939225       -0.8465407   0.8546626     0.31827571 -1.04869697          1.1534632            0.8665684 -0.02197431  0.1420772 -2.1929842  0.7100085
[5,]   0.130078254       -1.4990659   0.1496656    -0.97911061 -0.35193773         -0.9411004            0.8908983 -0.43909446  0.8464081  1.3948101  0.2169031
[6,]  -0.686026617       -0.1940155   2.3821560     0.09174794 -0.45147476          1.7120135            2.2777028  0.53418590 -1.1961514 -0.6553581 -1.4267814
> 
```

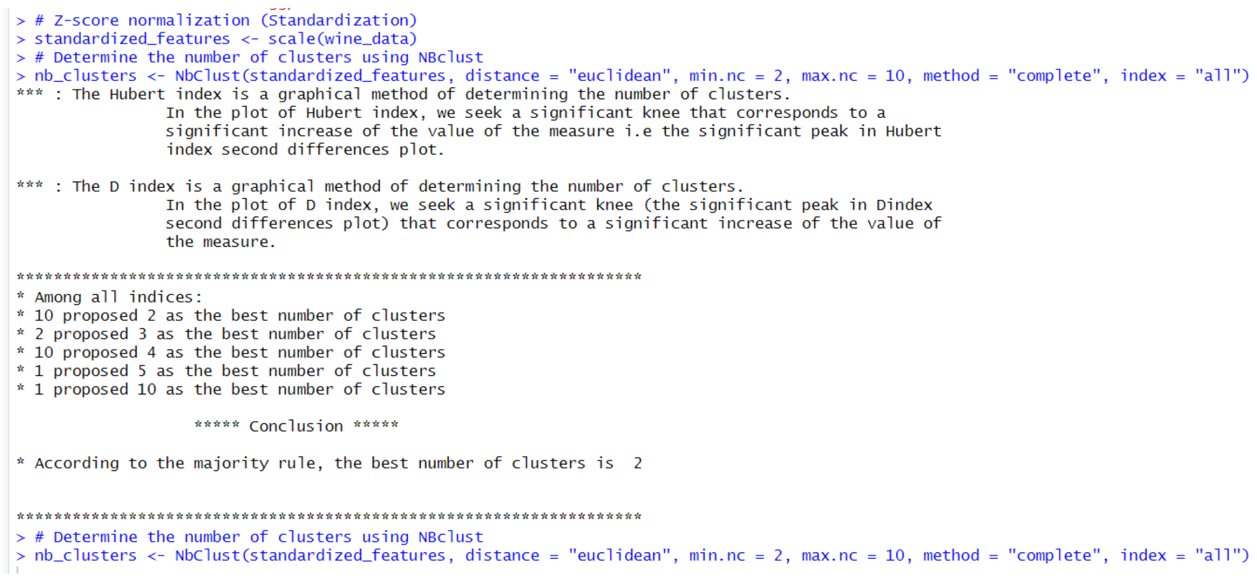


```
> # Display the dimensions of cleaned dataset after data zscore
> dim(clean_wine_data_zscore)
[1] 2210   11
```

Usage of Automation Tools to find the optimal number of clusters.

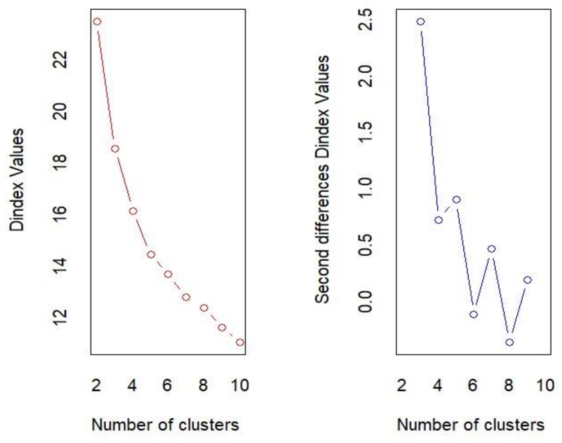
Four automated tools (NBclust, Elbow, Gap statistics, and Silhouette) are integrated into the code to determine the number of cluster centres.

1. NBclust tool.



```
> # Z-score normalization (Standardization)
> standardized_features <- scale(wine_data)
> # Determine the number of clusters using NBclust
> nb_clusters <- NbClust(standardized_features, distance = "euclidean", min.nc = 2, max.nc = 10, method = "complete", index = "all")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

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

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


*******************************************************************
> # Determine the number of clusters using NBclust
> nb_clusters <- NbClust(standardized_features, distance = "euclidean", min.nc = 2, max.nc = 10, method = "complete", index = "all")
```

The plotted diagram:



Through the NBclust tool techniques, mostly 3 clusters are favourably predicted as the best number of clusters. As it is clearly printed above in the section where they are determining according to the majority rule.

NB Clust output:

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

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

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


*******************************************************************
> print(nb_clusters)
$All.index
      KL      CH Hartigan     CCC     Scott     Marriot   TrCovW    TraceW Friedman  Rubin Cindex     DB Silhouette   Duda
2  3.6572  12.3793 118.3443 -54.7743  153.9936 7.472929e+35 11479485 29553.40   0.0859 1.0046 0.3034 0.2841     0.6293 0.9580
3  0.2377  65.6090 560.3503 -68.1995 1861.4052 8.933806e+35 10926677 28311.55   1.2103 1.0487 0.3241 0.7130     0.4827 0.8262
4  5.2290 239.5215  35.2567 -44.9081 4172.9953 6.746759e+35  6067522 23441.22  32.6935 1.2665 0.3460 1.3591     0.1765 0.9665
5  9.7655 190.7338  98.6165 -55.9322 4469.3137 9.446098e+35  5905375 23138.62  33.0095 1.2831 0.3659 1.5983     0.1338 0.9421
6  0.1413 177.8278 132.4541 -60.0279 5352.6012 9.807048e+35  5449178 22321.81  33.7170 1.3300 0.3670 1.7234     0.1209 0.8744
7  2.0095 177.4856  38.7639 -59.7358 6641.8783 8.280412e+35  4918834 21275.76  38.1718 1.3954 0.3841 1.8880     0.1270 0.7780
8  0.6552 159.7987 100.6276 -65.7870 6907.5014 9.801903e+35  4723240 20973.86  39.0168 1.4155 0.3877 1.8455     0.1123 0.5263
9  1.0873 157.5704  13.7366 -65.5261 8443.5024 7.023442e+35  4433410 20218.10  41.0982 1.4684 0.3984 1.7967     0.0984 0.9914
10 1.2047 142.2509  15.6533 -73.6633 8559.4107 8.306558e+35  4384045 20115.42  41.3958 1.4759 0.4215 1.7564     0.0960 0.7154
  Pseudot2  Beale Ratkowsky     Ball Ptbiserial    Frey McClain   Dunn Hubert SDindex Dindex   SDbw
2  118.3004 0.3265    0.0230 14776.700     0.1431 46.1853  0.0003 0.5539  3e-04  0.7280 3.1459 0.4979
3  562.3675 1.5658    0.0672  9437.183     0.3988  8.5337  0.0094 0.1376  3e-04  1.1105 3.1047 0.7624
4   33.7028 0.2576    0.2038  5860.304     0.3698  0.7609  0.7068 0.0842  1e-04  1.5661 2.8004 0.7514
5  104.3281 0.4572    0.1883  4627.725     0.3715 -0.0097  0.7432 0.0894  1e-04  1.7107 2.7849 0.7511
6  132.4636 1.0685    0.1899  3720.302     0.4112 -0.1685  0.8797 0.0924  1e-04  1.8238 2.7442 0.8062
7   31.9562 2.1055    0.1896  3039.395     0.4467 -0.1633  0.9540 0.0992  1e-04  2.0880 2.6899 0.9018
8  108.8899 6.6452    0.1823  2621.732     0.4478 -0.3710  0.9563 0.1002  1e-04  2.1101 2.6748 0.8988
9   13.7473 0.0646    0.1787  2246.456     0.4504 -0.2289  0.9582 0.1031  1e-04  1.9627 2.6309 0.7761
10  19.4961 2.9031    0.1708  2011.542     0.4541 -0.1386  0.9638 0.1093  1e-04  1.9532 2.6256 0.7484

$All.CriticalValues
  CritValue_Duda CritValue_PseudoT2 Fvalue_Beale
2         0.9168          244.6022       0.9804
3         0.9167          242.7853       0.1015
4         0.9001          108.0296       0.9927
5         0.9103          167.3714       0.9297
6         0.8989          103.6721       0.3826
7         0.8191           24.7307       0.0175
8         0.8237           25.8957       0.0000
```

```
9        0.9091         158.2874      1.0000
10        0.7582          15.6239      0.0010

$Best.nc
                KL      CH Hartigan      CCC   Scott      Marriot  TrCovW   TraceW Friedman   Rubin Cindex     DB
Number_clusters 5.0000   4.0000   4.0000   4.0000    4.00 4.000000e+00      4    4.000   4.0000  4.0000 2.0000 2.0000
Value_Index    9.7655 239.5215 525.0937 -44.9081 2311.59 4.886387e+35 4859155 4567.742  31.4832 -0.2013 0.3034 0.2841
            Silhouette  Duda PseudoT2  Beale Ratkowsky     Ball PtBiserial  Frey McClain   Dunn Hubert SDindex Dindex
Number_clusters     2.0000 2.000   2.0000 2.0000    4.0000    3.000    10.0000 3.0000   2e+00 2.0000      0   2.000     0
Value_Index        0.6293 0.958 118.3004 0.3265    0.2038 5339.517     0.4541 8.5337   3e-04 0.5539      0   0.728     0
              SDbw
Number_clusters 2.0000
Value_Index    0.4979

$Best.partition
   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  [62] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [123] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [184] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [245] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [306] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [367] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [428] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [489] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [550] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [611] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [672] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [733] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [794] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [855] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [916] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [977] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [ reached getOption("max.print") -- omitted 1700 entries ]
```

Based on the majority rule applied to the output from NBclust, the best number of clusters is determined to be 2. This decision is made by considering the number of indices favouring each cluster number. In this case:

- 10 indices proposed 2 as the best number of clusters.

- 2 indices proposed 3 as the best number of clusters.

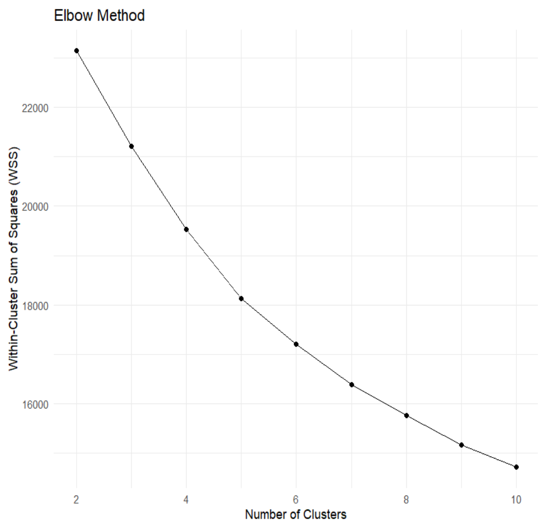
- 10 indices proposed 4 as the best number of clusters.

- 1 index proposed 5 as the best number of clusters.

- 1 index proposed 10 as the best number of clusters.

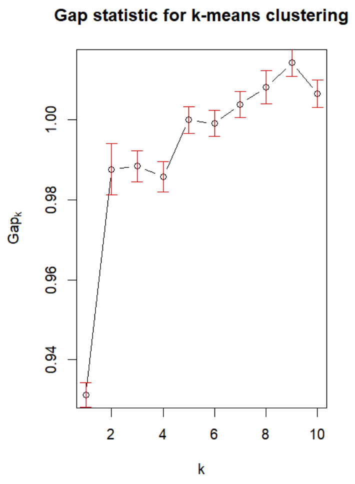
Since 10 indices favoured 2 clusters, the highest among all options, it is determined as the best number of clusters according to the majority rule. Therefore, you should proceed with 2 clusters based on this analysis.

2. Elbow Method.



The elbow method plot is a useful tool for determining the optimal number of clusters in K-means clustering . This plot shows the Within-Cluster Sum of Squares (WCSS) for different numbers of clusters. The WCSS is calculated as the sum of the squared distances between each point in a cluster and the centre of that cluster. The optimal number of clusters is usually identified by finding the "elbow" of the graph, where the curve starts to level off. In this particular case, the optimal number of clusters is 2.

3. Cluster Gap

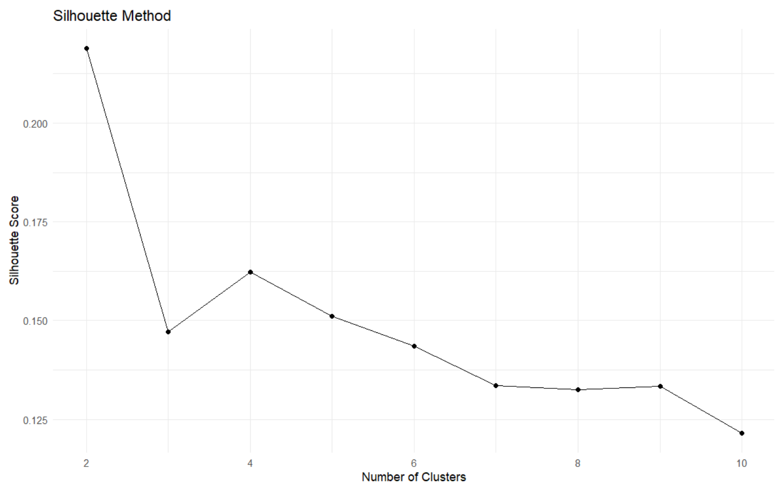


```
# Compute gap statistics with increased number of bootstrap replicates
> gap <- clusGap(standardized_features, FUN = function(x, k) kmeans(x, k, iter.max = 100), K.max = 10, B = 100)
Clustering k = 1,2,..., K.max (= 10): .. done
Bootstrapping, b = 1,2,..., B (= 100)  [one "." per sample]:
.................................................. 50
.................................................. 100
>
> # Plot the gap statistics
> plot(gap, main = "Gap statistic for k-means clustering")
>
> # Determine the optimal number of clusters using maxSE function
> best_clusters <- maxSE(gap$Tab[, "gap"], gap$Tab[, "SE.sim"], method = "Tibs2001SEmax")
>
> # Check contents of best_clusters list
> str(best_clusters)
 int 2
```

Based on the gap statistic analysis for k-means clustering, it is clear that the optimal number of clusters is 2.
This observation coincides with the peak in the gap statistic curve, which indicates the maximum separation between clusters.

As a result, we suggest dividing the data into two clusters based on this insight.
Further exploration of these clusters could reveal significant patterns within the dataset.

4. Silhouette Method



The image shows a line graph for determining the optimal number of clusters in unsupervised machine learning using silhouette scores, which range from -1 to 1 indicating how well data points fit within a cluster. The graph's x-axis represents the number of clusters, and the y-axis shows silhouette scores. A peak in the graph suggests the best cluster number. However, the graph lacks a clear maximum, implying either the data doesn't fit well into up to 10 clusters or the best cluster number might be around 4 or 5, though this is not strongly indicated.

K-means clustering

The next step involves performing a k-means clustering analysis using all the input variables from the dataset. To determine the most effective number of clusters (k), automated methods such as the Elbow Method, the Silhouette Method, and the Gap Statistic are employed. Each method evaluates the dataset differently to suggest the optimal k value that best represents the data's structure.

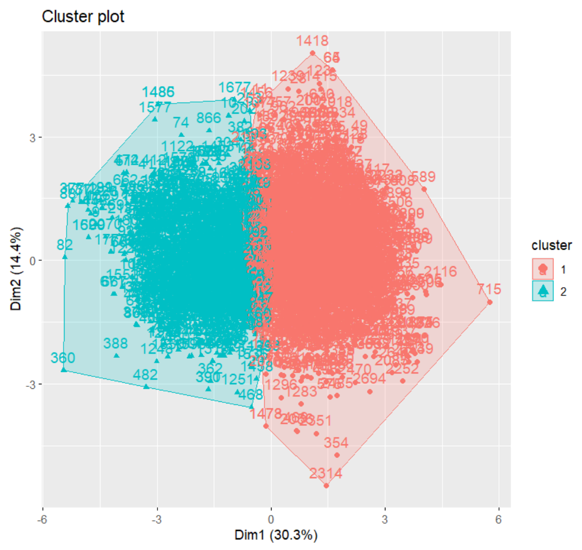
After applying these methods, the one that provides the most coherent and interpretable outcome will be selected to define the number of clusters for the k-means analysis. This approach ensures the analysis is grounded in a methodologically sound selection of k, leading to more meaningful and actionable insights from the clustering process.

For K = 2

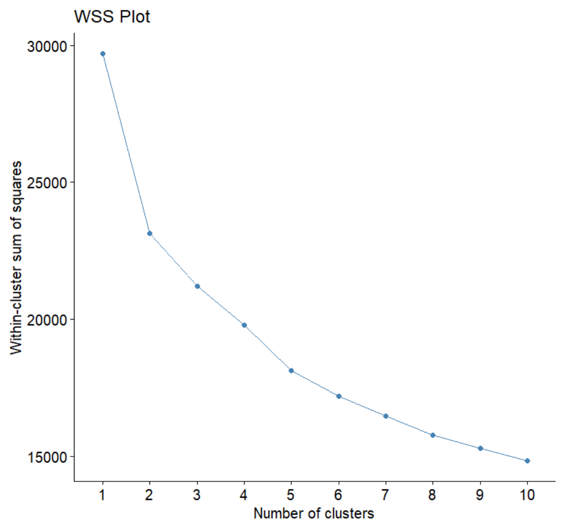
Output

```
# Add k=2
> k <- 2
> set.seed(123) # Set the seed for reproducibility
> km <- kmeans(standardized_features, centers = k, nstart = 25)
> fviz_cluster(km, data = standardized_features)
> print(km)
K-means clustering with 2 clusters of sizes 1615, 1085

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide
1    -0.1399463      -0.07052877  -0.1332093     -0.5941660 -0.2719014          -0.3987123
2     0.2083072       0.10498062   0.1982793      0.8844038  0.4047196           0.5934749
  total sulfur dioxide    density         pH   sulphates    alcohol
1           -0.5190249 -0.6522277  0.1625998 -0.02850696  0.5279349
2            0.7725578  0.9708274 -0.2420265  0.04243202 -0.7858202

Clustering vector:
   [1] 2 1 2 1 2 2 1 1 2 2 1 2 1 2 1 1 1 2 2 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 1 2 2 1 1 1 1
  [50] 2 2 1 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 1 2 2 1 2 2 2 2 2 2 1 1 1 2 2 1 1 1 2 2 1 2 1 2 1 1 2 2 1 2
  [99] 2 2 1 2 2 2 2 2 2 2 2 1 1 2 2 1 1 1 2 2 1 1 2 2 1 2 2 2 2 2 1 2 2 1 2 1 2 1 1 1 2 1 2 1 1 2 1 2 2
 [148] 1 2 2 2 2 1 1 1 2 2 2 1 2 2 2 2 1 1 2 2 1 2 2 2 2 2 1 1 1 2 2 2 2 1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2
 [197] 2 2 1 1 1 2 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 1 2 2 2 2 2 1 1 2 2 1 2 1 1 1 1
 [246] 1 2 2 1 1 1 1 2 1 2 2 1 2 2 2 1 2 1 2 2 1 1 1 2 2 2 2 1 2 1 2 2 2 2 2 2 1 1 2 2 1 2 2 2 2 2 2 2 2
 [295] 2 1 1 1 2 1 2 2 2 2 1 2 1 2 2 2 1 1 2 1 1 1 2 2 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2
 [344] 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 1 2 2 1 2 2 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2
 [393] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 2 2 1 2 1 1 2 2 2 1 1 2 2 2 2 2 1 2 2 1 2 2 2 1 2 2 2 1 2 1 2
 [442] 2 1 2 1 2 1 1 1 2 1 2 2 2 1 2 1 2 2 2 2 2 2 1 2 2 1 2 1 2 2 2 2 2 2 1 1 1 2 1 2 2 2 2 2 1 1 2 1 2
 [491] 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 1 2 1 2 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 2 2 1 2 2 2 1 1 1 2 2
 [540] 1 1 1 2 2 1 1 1 1 2 1 2 2 1 2 2 2 2 2 2 2 1 2 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 2 1 2 2 1
 [589] 1 1 2 2 2 2 2 2 1 1 1 2 1 1 2 1 1 2 1 1 1 1 2 1 2 1 1 2 2 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1
 [638] 1 2 1 2 1 2 2 2 2 1 1 2 1 1 1 1 2 2 2 2 2 1 1 2 2 2 2 1 2 2 2 1 1 2 2 1 1 2 2 1 2 2 1 2 2 2 2 2 2
 [687] 2 1 1 1 2 1 1 2 2 2 2 2 1 2 1 2 1 1 1 2 2 1 2 2 1 2 2 1 1 2 2 1 2 2 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1
 [736] 1 2 2 2 2 1 2 1 2 2 2 2 2 2 2 1 2 1 1 1 1 1 2 1 1 1 1 2 2 1 2 1 1 1 2 1 2 1 2 2 1 2 2 2 2 2 2 1 2
 [785] 1 1 1 1 1 1 2 2 1 1 1 1 2 2 2 2 2 1 1 2 1 2 2 1 1 2 2 2 1 1 1 1 1 2 1 1 2 1 1 1 2 2 1 1 1 2 2 2 1
 [834] 1 1 1 1 2 2 1 2 1 1 1 2 2 1 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 1 1 1 2 2 2 1 1 1 2 2 1 2 2 2 2 2 2 2 2 1
 [883] 2 2 2 1 1 1 1 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 2 2 2 2 1 2 2 2 2 1 1 2 1 2 2 2 2 1 1 2 2 1 1 2 1 1 1
 [932] 1 1 2 1 2 1 2 1 1 1 1 2 2 2 1 1 2 1 1 2 1 2 2 1 1 2 2 1 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1
 [981] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 2 2 1 1
 [ reached getOption("max.print") -- omitted 1700 entries ]

Within cluster sum of squares by cluster:
[1] 13054.25 10088.00
 (between_SS / total_SS =  22.1 %)

Available components:
```

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

>
```



```
# Display the BSS/TSS ratio
> BSS <- sum(km$betweenss)
> TSS <- sum(km$totss)
> BSS_TSS_ratio <- BSS/TSS
> cat("The ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")
The ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_squares (TSS) is 0.220511
>
> # Display the BSS and WSS indices
> cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")
The between_cluster_sums_of_squares (BSS) index is 6546.752
> cat("The within_cluster_sums_of_squares (WSS) index is", km$tot.withinss, "\n")
The within_cluster_sums_of_squares (WSS) index is 23142.25
>
> # Plot the WSS for k = 2
> fviz_nbclust(standardized_features, kmeans, method = "wss") +
+   ggtitle("WSS Plot") +
+   xlab("Number of clusters") +
+   ylab("Within-cluster sum of squares")

>
```



For K = 3

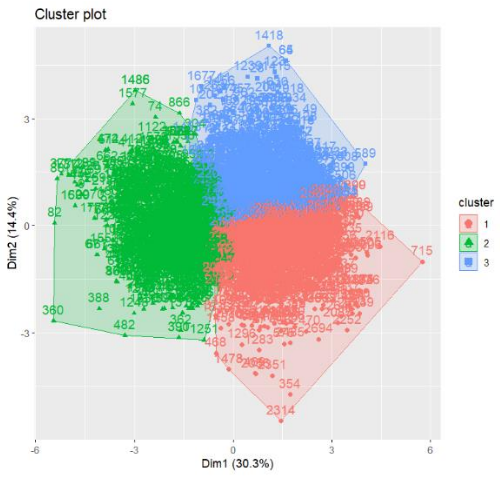
Outputs

```
> # Add k=3
> k <- 3
> set.seed(123) # Set the seed for reproducibility
> km <- kmeans(standardized_features, centers = k, nstart = 25)
> fviz_cluster(km, data = standardized_features)
> print(km)
K-means clustering with 3 clusters of sizes 881, 1019, 800

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar
1   -0.7777076     -0.115166257  -0.3562205     -0.6152924
2    0.1580391      0.103222127   0.1985254      0.9551923
3    0.6551482     -0.004652344   0.1394161     -0.5390855
   chlorides free sulfur dioxide total sulfur dioxide    density
1 -0.2931828          -0.2069455           -0.4116902 -0.6828114
2  0.4135009           0.6338607            0.8010510  1.0148477
3 -0.2038293          -0.5794813           -0.5669650 -0.5407162
         pH   sulphates    alcohol
1  0.8087160  0.25043639  0.5283395
```

```
2 -0.2445912  0.04231209 -0.8147122

3 -0.5790504 -0.32968809  0.4559058


Clustering vector:

  [1] 2 3 2 3 2 2 3 3 2 3 3 3 1 2 1 3 3 2 2 3 2 2 3 3 3 2 1 3 1

 [30] 3 1 3 1 1 3 2 2 2 2 2 3 3 3 2 2 1 3 3 3 2 2 3 2 1 3 3 3 3

 [59] 3 3 1 2 2 3 3 3 3 2 3 2 2 3 2 2 2 2 3 2 1 1 1 2 3 3 3 1 3

 [88] 2 3 3 1 2 3 1 2 2 3 2 2 2 1 2 2 2 2 2 2 2 2 3 1 2 2 1 1 3

[117] 2 2 3 3 2 2 3 2 2 2 2 2 1 2 2 3 3 1 2 1 1 3 2 3 2 3 3 2 1

[146] 2 2 1 2 2 2 2 3 3 3 2 2 2 3 2 2 2 2 1 3 2 2 3 2 2 2 2 2 3

[175] 3 1 2 2 2 2 1 3 3 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3

[204] 2 2 2 2 2 2 1 1 1 1 1 1 3 3 2 2 3 2 3 2 3 3 3 1 3 1 3 2 2

[233] 2 2 2 3 3 2 2 1 2 1 1 3 3 3 2 2 1 1 1 3 3 3 2 2 1 2 2 2 3

[262] 2 1 2 2 1 1 3 2 2 2 2 1 2 3 2 2 2 2 2 2 2 1 1 2 2 1 2 2 2 2

[291] 2 2 2 2 2 3 3 3 2 3 2 2 1 2 3 2 3 2 2 2 3 3 2 3 3 3 2 2 1

[320] 3 2 2 2 2 2 2 2 2 2 3 2 2 2 2 3 1 2 2 2 2 2 2 2 2 2 2 2 2 1

[349] 2 3 2 3 2 1 2 2 2 3 3 2 2 2 2 2 1 2 2 3 2 2 3 2 1 3 2 2 2

[378] 2 2 1 2 3 2 2 3 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

[407] 2 3 3 1 2 2 1 3 3 1 2 2 2 1 3 2 2 2 2 2 3 2 2 3 2 2 2 1 2

[436] 3 2 1 2 1 3 2 3 2 3 2 1 1 1 2 1 2 2 2 3 2 3 2 2 2 2 2 2 1

[465] 2 2 1 1 1 2 2 2 2 2 2 3 3 1 2 3 2 2 2 2 2 3 3 2 3 2 2 2 2

[494] 2 2 1 2 2 2 2 2 2 3 2 2 3 2 1 1 3 3 3 1 2 2 2 2 2 2 2 2 2

[523] 2 2 2 3 3 3 2 2 1 2 2 2 3 1 1 2 2 3 3 3 2 2 3 3 3 1 2 1 2

[552] 2 3 2 2 3 2 2 2 2 1 2 1 1 2 3 2 1 1 1 1 3 1 2 1 3 3 3 2 3

[581] 2 1 1 2 3 2 2 3 3 1 2 2 2 2 2 2 1 1 3 2 3 1 2 3 3 2 3 3 3

[610] 1 2 3 2 1 1 2 2 1 1 3 1 1 2 1 2 1 1 3 1 3 3 3 3 3 1 2 3 1

[639] 2 3 2 3 2 2 2 2 3 1 2 1 1 3 1 2 2 2 2 2 3 3 2 2 2 2 3 2 2

[668] 2 1 3 2 3 1 1 2 2 1 2 2 3 2 2 2 2 2 2 2 1 3 3 2 3 1 2 2 2

[697] 2 2 3 2 3 2 3 3 1 2 3 1 2 3 3 2 2 1 1 2 2 3 2 2 3 3 3 3 2

[726] 2 3 2 1 1 1 1 1 1 3 3 2 2 2 2 1 2 3 2 2 2 2 2 2 2 1 2 3 1

[755] 3 3 3 2 3 3 3 1 2 2 3 2 1 3 3 2 1 2 1 2 2 3 2 2 2 2 2 2 3

[784] 2 3 3 1 3 1 3 2 2 3 3 3 3 2 2 2 2 2 1 1 2 1 2 2 1 3 2 2 2

[813] 3 3 3 3 1 2 1 3 2 1 3 3 2 2 3 3 3 1 2 2 1 1 1 1 1 2 2 3 2

[842] 1 3 3 2 2 1 3 2 1 1 2 2 2 2 1 2 2 2 2 2 2 3 3 1 2 2 2 3 3
```

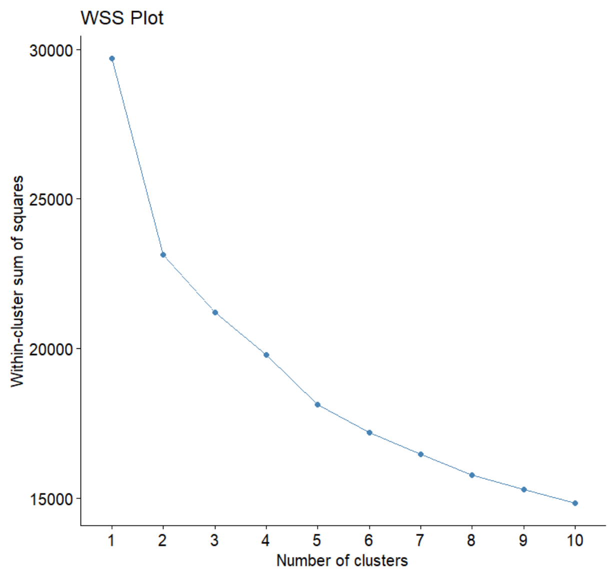
```
 [871] 3 2 2 1 2 2 2 2 2 2 2 3 2 2 2 3 1 1 3 2 2 2 2 2 3 2 2 2 3
 [900] 2 2 2 3 2 2 2 2 2 1 2 2 2 2 3 1 2 3 2 2 2 2 1 3 2 2 3 1 2
 [929] 1 3 1 3 1 2 1 2 3 2 3 3 1 3 2 2 2 1 1 2 1 1 2 3 2 2 3 3 2
 [958] 2 3 3 2 3 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 3 3 3 1 1 3 3 3
 [987] 3 3 3 1 3 2 3 3 2 2 2 2 3 3
 [ reached getOption("max.print") -- omitted 1700 entries ]

Within cluster sum of squares by cluster:
[1] 6403.376 9114.738 5691.925
 (between_SS / total_SS =  28.6 %)



Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
> # Display the BSS/TSS ratio
> BSS <- sum(km$betweenss)
> TSS <- sum(km$totss)
> BSS_TSS_ratio <- BSS/TSS
> cat("The ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")
The ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_squares (TSS) is 0.2855927
>
> # Display the BSS and WSS indices
> cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")
The between_cluster_sums_of_squares (BSS) index is 8478.961
> cat("The within_cluster_sums_of_squares (WSS) index is", km$tot.withinss, "\n")
The within_cluster_sums_of_squares (WSS) index is 21210.04
>
> # Plot the WSS
> fviz_nbclust(standardized_features, kmeans, method = "wss") +
+   ggtitle("WSS Plot") +
+   xlab("Number of clusters") +

+   ylab("Within-cluster sum of squares")

> # Add k=3

> k <- 3

> set.seed(123) # Set the seed for reproducibility

> km <- kmeans(standardized_features, centers = k, nstart = 25)

> fviz_cluster(km, data = standardized_features)

> print(km)

K-means clustering with 3 clusters of sizes 881, 1019, 800


Cluster means:

  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide

1    -0.7777076     -0.115166257  -0.3562205     -0.6152924 -0.2931828         -0.2069455

2     0.1580391      0.103222127   0.1985254      0.9551923  0.4135009          0.6338607

3     0.6551482     -0.004652344   0.1394161     -0.5390855 -0.2038293         -0.5794813

  total sulfur dioxide    density         pH   sulphates    alcohol

1          -0.4116902 -0.6828114  0.8087160  0.25043639  0.5283395

2           0.8010510  1.0148477 -0.2445912  0.04231209 -0.8147122

3          -0.5669650 -0.5407162 -0.5790504 -0.32968809  0.4559058


Clustering vector:

   [1] 2 3 2 3 2 2 3 3 2 3 3 3 1 2 1 3 3 2 2 3 2 2 3 3 3 2 1 3 1 3 1 3 1 1 3 2 2 2 2 2 3 3 3 2 2 1 3 3 3

  [50] 2 2 3 2 1 3 3 3 3 3 3 1 2 2 3 3 3 3 2 3 2 2 3 2 2 2 2 3 2 1 1 1 2 3 3 3 1 3 2 3 3 1 2 3 1 2 2 3 2

  [99] 2 2 1 2 2 2 2 2 2 2 2 3 1 2 2 1 1 3 2 2 3 3 2 2 3 2 2 2 2 2 1 2 2 3 3 1 2 1 1 3 2 3 2 3 3 2 1 2 2

 [148] 1 2 2 2 2 3 3 3 2 2 2 3 2 2 2 2 1 3 2 2 3 2 2 2 2 2 3 3 1 2 2 2 2 1 3 3 2 2 2 3 2 2 2 2 2 2 2 2 2

 [197] 2 2 3 3 3 3 3 2 2 2 2 2 2 1 1 1 1 1 1 3 3 2 2 3 2 3 2 3 3 3 1 3 1 3 2 2 2 2 2 3 3 2 2 1 2 1 1 3 3

 [246] 3 2 2 1 1 1 3 3 3 2 2 1 2 2 2 3 2 1 2 2 1 1 3 2 2 2 2 1 2 3 2 2 2 2 2 2 1 1 2 2 1 2 2 2 2 2 2 2 2

 [295] 2 3 3 3 2 3 2 2 1 2 3 2 3 2 2 2 3 3 2 3 3 3 2 2 1 3 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 3 1 2 2 2 2 2 2 2 2

 [344] 2 2 2 2 1 2 3 2 3 2 1 2 2 2 3 3 2 2 2 2 2 1 2 2 3 2 2 3 2 1 3 2 2 2 2 2 1 2 3 2 2 3 2 1 2 2 2 2 2 2

 [393] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 1 2 2 1 3 3 1 2 2 2 1 3 2 2 2 2 2 3 2 2 3 2 2 2 1 2 3 2 1 2 1 3

 [442] 2 3 2 3 2 1 1 1 2 1 2 2 2 3 2 3 2 2 2 2 2 2 1 2 2 1 1 1 2 2 2 2 2 2 3 3 1 2 3 2 2 2 2 2 3 3 2 3 2

 [491] 2 2 2 2 2 1 2 2 2 2 2 2 3 2 2 3 2 1 1 3 3 3 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 2 2 1 2 2 2 3 1 1 2 2

 [540] 3 3 3 2 2 3 3 3 1 2 1 2 2 3 2 2 3 2 2 2 2 1 2 1 1 2 3 2 1 1 1 1 3 1 2 1 3 3 3 2 3 2 1 1 2 3 2 2 3

 [589] 3 1 2 2 2 2 2 2 1 1 3 2 3 1 2 3 3 2 3 3 3 1 2 3 2 1 1 2 2 1 1 3 1 1 2 1 2 1 1 3 1 3 3 3 3 3 1 2 3
```

```
[638] 1 2 3 2 3 2 2 2 2 3 1 2 1 1 3 1 2 2 2 2 2 3 3 2 2 2 2 3 2 2 2 1 3 2 3 1 1 2 2 1 2 2 3 2 2 2 2 2 2

[687] 2 1 3 3 2 3 1 2 2 2 2 2 3 2 3 2 3 3 1 2 3 1 2 3 3 2 2 1 1 2 2 3 2 2 3 3 3 3 2 2 3 2 1 1 1 1 1 1 3

[736] 3 2 2 2 2 1 2 3 2 2 2 2 2 2 2 1 2 3 1 3 3 3 2 3 3 3 1 2 2 3 2 1 3 3 2 1 2 1 2 2 3 2 2 2 2 2 2 3 2

[785] 3 3 1 3 1 3 2 2 3 3 3 3 2 2 2 2 2 1 1 2 1 2 2 1 3 2 2 2 3 3 3 3 1 2 1 3 2 1 3 3 2 2 3 3 3 1 2 2 1

[834] 1 1 1 1 2 2 3 2 1 3 3 2 2 1 3 2 1 1 2 2 2 2 1 2 2 2 2 2 2 3 3 1 2 2 2 3 3 3 2 2 1 2 2 2 2 2 2 2 3

[883] 2 2 2 3 1 1 3 2 2 2 2 2 3 2 2 2 3 2 2 2 3 2 2 2 2 2 1 2 2 2 2 3 1 2 3 2 2 2 2 1 3 2 2 3 1 2 1 3 1

[932] 3 1 2 1 2 3 2 3 3 1 3 2 2 2 1 1 2 1 1 2 3 2 2 3 3 2 2 3 3 2 3 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 3 3

[981] 3 1 1 3 3 3 3 3 3 1 3 2 3 3 2 2 2 2 3 3

 [ reached getOption("max.print") -- omitted 1700 entries ]


Within cluster sum of squares by cluster:

[1] 6403.376 9114.738 5691.925

 (between_SS / total_SS =  28.6 %)


Available components:


[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"

[7] "size"         "iter"         "ifault"
```



```
# Display the BSS/TSS ratio
> BSS <- sum(km$betweenss)
> TSS <- sum(km$totss)
> BSS_TSS_ratio <- BSS/TSS
> cat("The ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")
The ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_squares (TSS) is 0.2855927
>
```

```
> # Display the BSS and WSS indices
> cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")
The between_cluster_sums_of_squares (BSS) index is 8478.961
> cat("The within_cluster_sums_of_squares (WSS) index is", km$tot.withinss, "\n")
The within_cluster_sums_of_squares (WSS) index is 21210.04
>
> # Plot the WSS
> fviz_nbclust(standardized_features, kmeans, method = "wss") +
+   ggtitle("WSS Plot") +
+   xlab("Number of clusters") +
+   ylab("Within-cluster sum of squares")
```

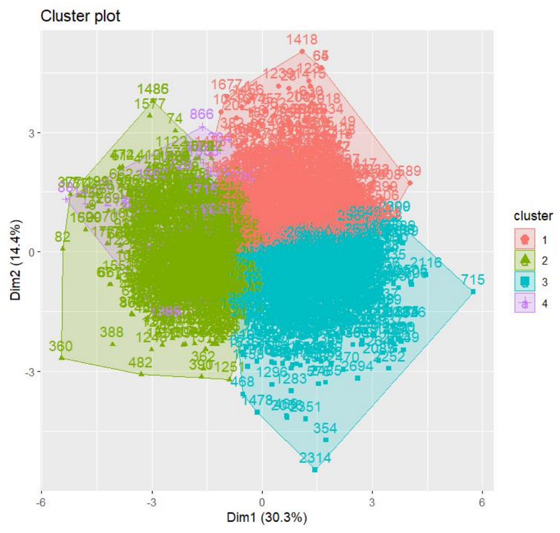


For K = 4

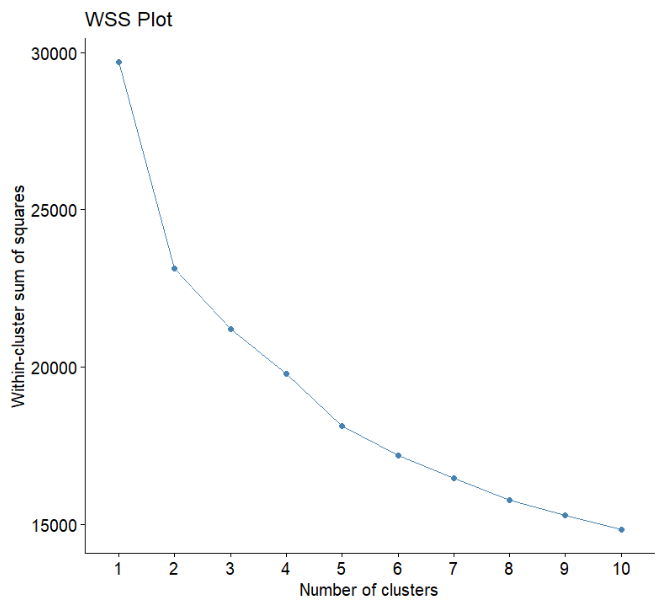
Output

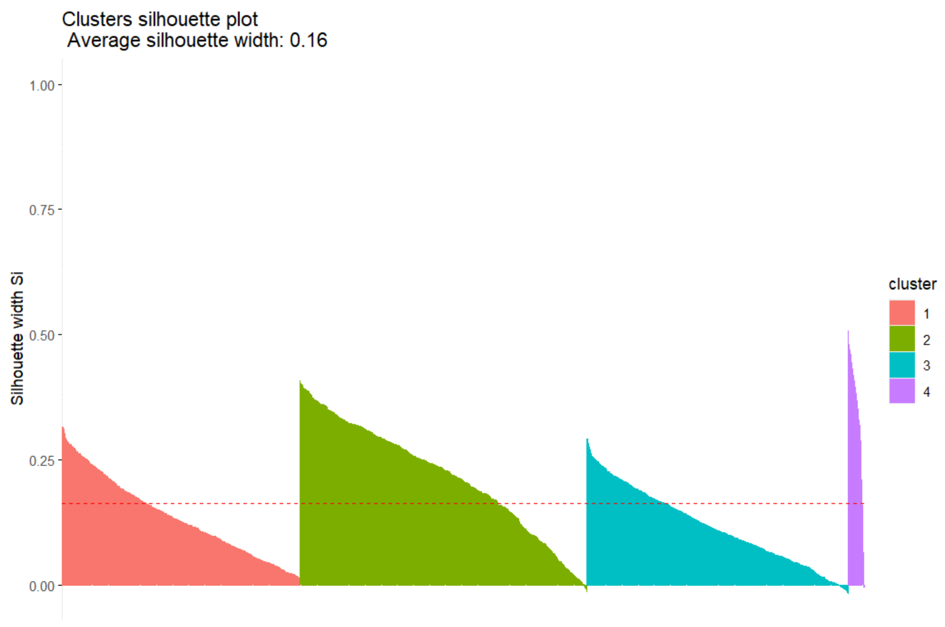
```
> # Add k=4
> k <- 4
> set.seed(123) # Set the seed for reproducibility
> km <- kmeans(standardized_features, centers = k, nstart = 25)
> fviz_cluster(km, data = standardized_features)
> print(km)
K-means clustering with 4 clusters of sizes 798, 967, 880, 55

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide total sulfur dioxide
1    0.6605643      0.000255662   0.1276565     -0.5406396 -0.2352937          -0.5751762          -0.55899067
2    0.1654316      0.083307128   0.1633114      1.0276079  0.1578381           0.6392218           0.82992885
3   -0.7685667     -0.122871956  -0.3576269     -0.6177972 -0.3017874          -0.1994406          -0.40743236
4   -0.1957083      0.497551095   0.9985393     -0.3382701  5.4674150           0.2976518           0.03770612
     density         pH   sulphates    alcohol
1 -0.5414788 -0.5777631 -0.32543302  0.4579895
2  1.0558841 -0.2261453  0.04722806 -0.8082136
3 -0.6800679  0.8128470  0.25968270  0.5252066
4  0.1730882 -0.6466893 -0.26354102 -0.8384526

Clustering vector:
   [1] 2 1 4 1 2 2 1 1 2 1 1 1 3 2 3 1 1 2 2 1 2 2 1 1 1 4 3 1 3 1 3 1 3 3 1 2 2 2 2 2 1 1 1 2 2 3 1 1 1 2 2 1 2 3 1 1
  [57] 1 1 1 1 3 2 2 1 1 1 1 2 1 2 2 1 2 2 2 2 1 2 3 3 3 2 2 1 1 3 1 4 1 1 3 2 1 3 2 2 1 4 4 2 3 2 2 2 2 2 2 2 2 1 3 4
 [113] 4 3 3 1 4 2 1 1 2 2 1 2 2 2 2 2 3 2 2 1 1 3 2 3 1 1 2 1 2 1 1 2 3 2 4 3 2 2 2 2 1 1 1 2 2 2 1 2 2 2 2 3 1 2 2 1
 [169] 2 2 2 2 2 1 1 3 2 2 2 2 3 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 1 1 4 4 4 2 1 2 1
 [225] 1 1 3 1 3 1 2 2 2 2 2 1 1 2 2 3 2 3 3 1 1 1 2 2 3 3 3 1 1 1 2 2 3 2 2 2 1 2 3 2 2 3 3 1 2 2 2 2 3 2 1 2 2 2 2 2
 [281] 2 3 3 2 2 3 2 2 2 2 2 2 2 2 2 2 1 1 1 2 1 2 2 3 4 1 2 1 2 2 2 1 1 2 1 1 1 2 2 3 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 3
```

```
[337] 4 2 2 2 2 4 4 2 2 2 2 3 2 1 2 1 2 3 2 2 2 1 1 2 2 2 2 2 3 4 2 1 2 4 1 2 3 1 2 4 2 2 2 3 2 1 2 2 1 2 3 2 2 2 2 2
[393] 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 1 1 3 2 2 3 1 1 3 2 2 2 3 1 2 2 2 2 2 1 2 2 1 2 2 2 3 2 1 2 3 2 3 1 2 1 2 1 2 3 3
[449] 3 2 3 2 2 2 1 2 1 2 2 2 2 2 2 3 2 2 3 3 3 2 2 2 2 2 2 1 1 3 2 1 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 3 2 2 2 2 2 2 1 2
[505] 2 1 2 3 3 1 1 1 3 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 4 2 3 2 2 2 1 3 3 4 2 1 1 1 2 2 1 1 1 3 2 3 2 2 1 2 2 1 2 2 2 2
[561] 3 2 3 3 2 1 2 3 3 3 3 1 3 4 3 1 1 1 2 1 2 3 3 2 1 4 4 1 1 3 2 2 2 2 2 2 3 3 1 2 1 3 2 1 1 2 1 1 1 3 2 1 2 3 3 2
[617] 2 3 3 1 3 3 2 3 2 3 3 1 3 1 1 1 1 1 3 2 1 3 2 1 2 1 2 2 2 2 1 3 2 3 3 1 3 2 2 2 2 2 1 1 2 2 2 2 1 2 2 2 3 1 2 1
[673] 3 3 2 2 3 2 2 1 2 2 2 2 2 2 2 4 1 1 2 1 3 2 2 2 2 2 1 2 1 4 1 1 3 2 1 3 2 1 1 2 2 3 3 2 2 1 2 2 1 1 1 1 2 4 1 2
[729] 3 3 3 3 3 3 1 1 2 2 2 2 3 2 1 2 2 2 2 2 2 2 3 2 1 3 1 1 1 2 1 1 1 3 2 2 1 2 3 1 1 2 3 2 3 2 2 1 2 2 2 2 2 2 1 2
[785] 1 1 3 1 3 1 2 2 1 1 1 1 2 2 2 4 2 3 3 2 3 2 2 3 1 2 2 2 1 1 1 1 3 2 3 1 2 3 1 1 2 2 1 1 1 3 2 2 2 3 3 3 3 2 2 1
[841] 2 3 1 1 2 2 3 1 2 3 3 2 2 2 2 3 2 2 2 2 2 2 1 1 3 4 2 2 1 1 1 2 2 3 2 2 2 2 2 2 2 1 2 2 2 1 3 3 1 2 2 2 2 2 1 2
[897] 2 2 1 2 4 2 1 2 2 2 2 2 3 2 2 2 2 1 3 2 1 2 2 2 2 3 1 2 2 1 3 2 3 1 3 1 3 2 3 2 1 2 1 1 3 1 4 4 2 3 3 2 3 3 4 1
[953] 2 2 1 1 2 2 1 1 2 1 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 1 1 1 3 3 1 1 1 1 1 1 3 1 2 1 1 2 2 2 2 1 1
 [ reached getOption("max.print") -- omitted 1700 entries ]

Within cluster sum of squares by cluster:
[1] 5450.8405 7126.9939 6376.7017  577.2049
 (between_SS / total_SS =  34.2 %)

Available components:

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



```
> # Display the BSS/TSS ratio
> BSS <- sum(km$betweenss)
> TSS <- sum(km$totss)
> BSS_TSS_ratio <- BSS/TSS
> cat("The ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")
The ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_squares (TSS) is 0.342122
>
> # Display the BSS and WSS indices
> cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")
The between_cluster_sums_of_squares (BSS) index is 10157.26
> cat("The within_cluster_sums_of_squares (WSS) index is", km$tot.withinss, "\n")
The within_cluster_sums_of_squares (WSS) index is 19531.74
>
> # Plot the WSS for k = 4
> fviz_nbclust(standardized_features, kmeans, method = "wss") +
+   ggtitle("WSS Plot") +
+   xlab("Number of clusters") +
+   ylab("Within-cluster sum of squares")
```





```
> # Display the silhouette plot
> # Compute silhouette values
> sil <- silhouette(km$cluster, dist(standardized_features))
>
> # Plot the silhouette
> fviz_silhouette(sil)
  cluster size ave.sil.width
1       1  798          0.14
2       2  967          0.22
3       3  880          0.12
4       4   55          0.34
```

```
>


> # Calculate average silhouette width
> average_silhouette_width <- mean(sil[, "sil_width"])
> cat("Average silhouette width:", average_silhouette_width)
Average silhouette width: 0.1622155
```

Silhouette plot and the computed average silhouette widths:

- We have identified four clusters, each with varying sizes: Cluster 1 (798 data points), Cluster 2 (967 data points), Cluster 3 (880 data points), and Cluster 4 (55 data points).
- The average silhouette width across all clusters is 0.1622155.
- Cluster 4 has the highest average silhouette width (0.34), indicating that the data points in this cluster are well-clustered and have high cohesion and separation compared to the other clusters.
- Cluster 2 also shows a relatively high average silhouette width (0.22), suggesting good clustering.
- Cluster 1 and Cluster 3 have lower average silhouette widths (0.14 and 0.12, respectively), indicating that the data points in these clusters are closer to neighboring clusters.

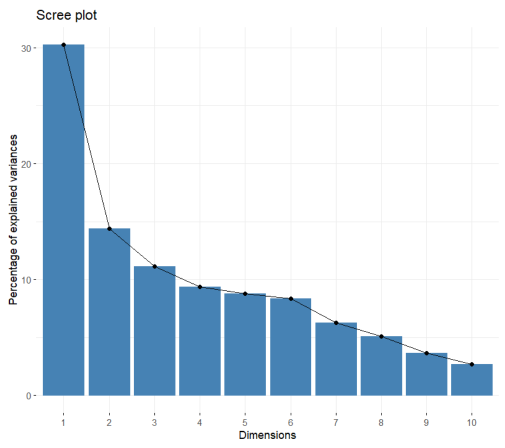
Overall, the average silhouette width of 0.1622155 suggests a moderate level of clustering quality, with some clusters showing better separation and cohesion than others.

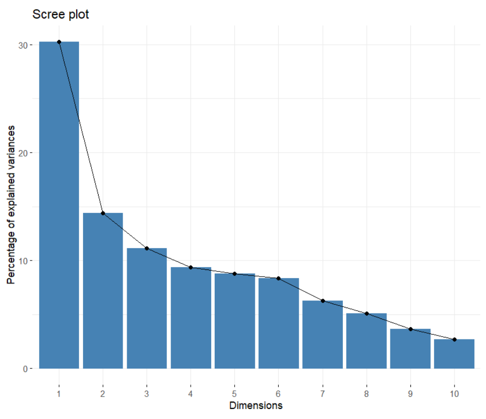
2nd Subtask - PCA Analaysis

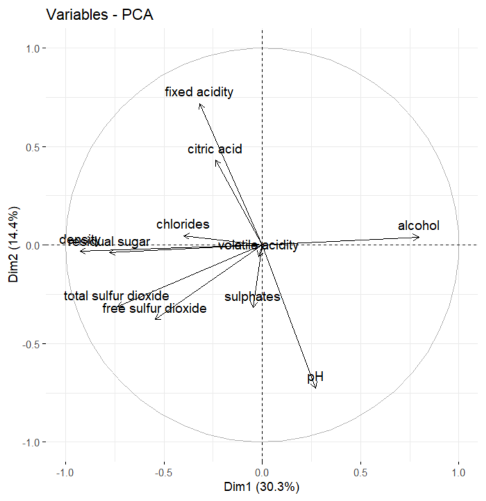
To conduct PCA analysis, we begin by standardizing the features, and then use prcomp() function to extract th e principal components. We can summarize the results of the PCA analysis by calling the summary() function o n the prcomp() object that we created. The output of the summary() function shows the importance of each c omponent, which is represented by the standard deviation, proportion of variance, and cumulative proportio n.

```
> # PCA analysis
> pca_result <- prcomp(standardized_features, center = TRUE)
> print(summary(pca_result))
Importance of components:
                         PC1    PC2    PC3    PC4     PC5     PC6    PC7     PC8     PC9    PC10    PC11
Standard deviation     1.8242 1.2570 1.1075 1.0147 0.98393 0.95690 0.8298 0.74653 0.63217 0.54177 0.11613
Proportion of Variance 0.3025 0.1436 0.1115 0.0936 0.08801 0.08324 0.0626 0.05066 0.03633 0.02668 0.00123
Cumulative Proportion  0.3025 0.4461 0.5576 0.6512 0.73926 0.82250 0.8851 0.93576 0.97209 0.99877 1.00000
```

```
# Print eigenvalues and eigenvectors
> print(pca_result$sdev) #this print the eigenvalues
 [1] 1.8241912 1.2569766 1.1074657 1.0146754 0.9839284 0.9569032 0.8298009
 [8] 0.7465269 0.6321703 0.5417698 0.1161261
> print(pca_result$rotation)#this prints the eigenvectors
                             PC1         PC2          PC3          PC4
fixed acidity        -0.17409345  0.57068553  0.087247713  0.209430682
volatile acidity     -0.01016873 -0.04701333 -0.667603697  0.061110370
citric acid          -0.12950667  0.34294115  0.520359898  0.003189287
residual sugar       -0.42544819 -0.03080478 -0.209924618  0.175014970
chlorides            -0.21811880  0.03812122  0.006272862 -0.758094988
free sulfur dioxide  -0.29885491 -0.30122472  0.147705942  0.380551214
total sulfur dioxide -0.40403330 -0.24982649  0.024873354  0.226699978
density              -0.50832743 -0.02467803 -0.051425500 -0.060220958
pH                    0.14858189 -0.57741901  0.195709942 -0.163980752
sulphates            -0.02602988 -0.25271766  0.410360266  0.076628530
alcohol               0.43649289  0.03254900 -0.005725580  0.338312875
                             PC5        PC6        PC7         PC8
fixed acidity         0.20466350 -0.1691181  0.2528023 -0.56140981
volatile acidity      0.56311078  0.1971664 -0.3184433 -0.13761562
citric acid           0.10591093  0.3127275 -0.6752618  0.03117685
residual sugar       -0.13074005 -0.2738351 -0.3288721  0.36160560
chlorides             0.20213482  0.3607222  0.1601849  0.07222723
free sulfur dioxide  -0.10605033  0.5047138  0.2199839  0.08639415
total sulfur dioxide  0.10941555  0.2807351  0.1242957 -0.29862242
density              -0.04827851 -0.3337286 -0.1442155 -0.06474637
pH                   -0.05957859 -0.1683622 -0.3408029 -0.57671562
sulphates             0.72411959 -0.3428958  0.1697914  0.29504600
alcohol               0.13565577  0.1969097 -0.1235886  0.09191924
                             PC9        PC10         PC11
fixed acidity        -0.31196772 -0.12199248  0.175362607
volatile acidity      0.12383398 -0.23777976  0.008402549
citric acid           0.16490149 -0.03979703  0.009611698
residual sugar       -0.41269261  0.15471137  0.467013298
chlorides            -0.40438749  0.10110071  0.024859732
free sulfur dioxide  -0.21364329 -0.53399998 -0.025963500
total sulfur dioxide  0.24411170  0.68210746  0.045413969
density              -0.07085401 -0.06888076 -0.765677786
pH                   -0.25938101 -0.10434956  0.143207351
sulphates             0.02244379 -0.03589486  0.039857346
alcohol              -0.59014224  0.35424492 -0.373188404
```







```
# Calculate cumulative variance explained by each principal component
> cumulative_variance <- cumsum(pca_result$sdev^2 / sum(pca_result$sdev^2))
>
> # Find the number of principal components required to achieve cumulative score > 0.85
> pc_cutoff <- min(which(cumulative_variance > 0.85))
>
> # Select principal components
> selected_pcs <- paste0("PC", 1:pc_cutoff)
>
> # Create a transformed dataset with only the selected principal components as attributes
> transformed_dataset <- predict(pca_result, standardized_features)[, 1:pc_cutoff]
> colnames(transformed_dataset) <- selected_pcs
>
> # Print selected PCs
> print(selected_pcs)
[1] "PC1" "PC2" "PC3" "PC4" "PC5" "PC6" "PC7"
>
```

```
> # Print transformed dataset with selected PCs
> head(transformed_dataset)
           PC1       PC2       PC3         PC4        PC5        PC6        PC7
[1,] -144.01001 -1.919518 -25.19102   -6.488273 -21.951113  -96.70759 -41.957092
[2,]   24.03465  4.551442 -12.35863   19.761717   9.851178   16.06237 -12.935786
[3,] -204.28496  3.371050 -22.80745 -238.325428  52.393641   15.95704   4.197964
[4,]   24.03465  4.551442 -12.35863   19.761717   9.851178   16.06237 -12.935786
[5,] -144.01001 -1.919518 -25.19102   -6.488273 -21.951113  -96.70759 -41.957092
[6,] -213.88575  4.936130 -12.31008  -11.934805 -20.938769 -132.15376 -82.067643

>
```

The cumulative variance explained by each principal component was computed, revealing that the first seven principal components were necessary to achieve a cumulative score exceeding 85%. These selected principal components denoted as PC1 to PC7, were then utilized to create a transformed dataset. This transformed dataset, comprising observations represented in the space of the selected principal components, exemplifies the data's reduced dimensionality while retaining significant information. Each row of the transformed dataset corresponds to an observation, while each column represents a selected principal component, facilitating a compact representation of the dataset's variability.

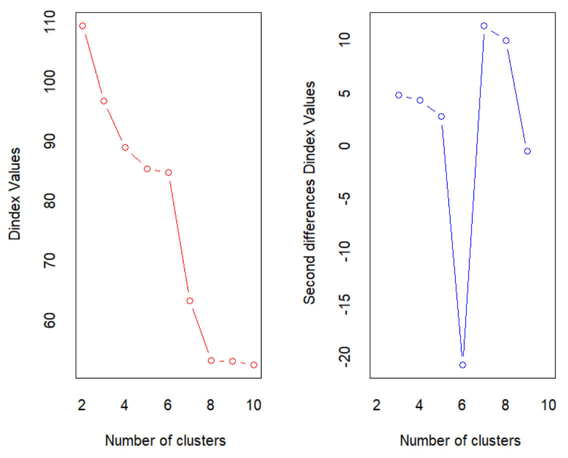
For our upcoming attempt, we have chosen the above components as new features to be clustered. As we have a new dataset, it is necessary to determine the appropriate k for our k-means clustering attempt. To accomplish this, we will apply the same four automated tools that we utilized earlier to this new PCA-based dataset.

NB Clust

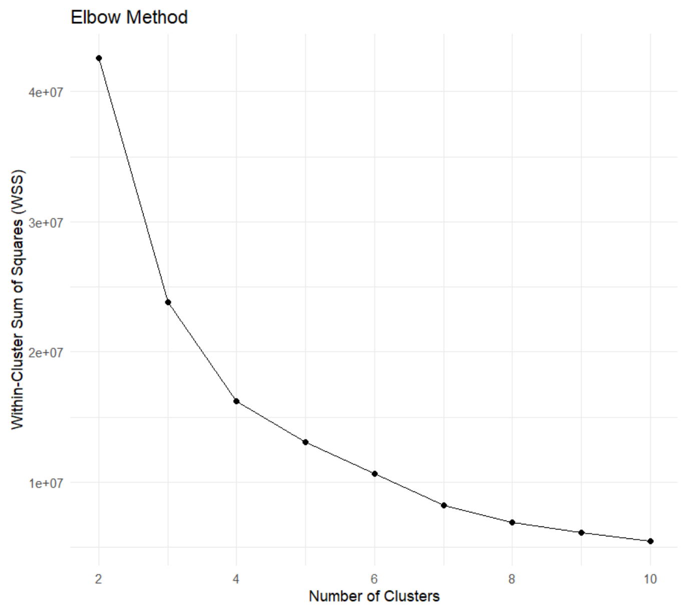
```
> # Determine the number of clusters using NbClust
> nb_clusters <- NbClust(transformed_dataset, distance = "euclidean", min.nc = 2, max.nc = 10, method = "complete", index = "all")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of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

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

*******************************************************************
* Among all indices:
* 9 proposed 2 as the best number of clusters
* 3 proposed 3 as the best number of clusters
* 2 proposed 5 as the best number of clusters
* 3 proposed 6 as the best number of clusters
* 2 proposed 7 as the best number of clusters
* 3 proposed 8 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


*******************************************************************
> print(nb_clusters)
$All.index
       KL       CH  Hartigan      CCC    Scott     Marriot       TrCovW      TraceW Friedman   Rubin Cindex     DB
2  14.0364 5243.068  670.8752  -10.2886  3375.911 3.008727e+37 1.403763e+14 45293649 291747.1  2.9433 0.1930 0.7099
```

```
3   1.1353 3607.495  551.7370  -45.2890  4105.731 5.166232e+37 8.522332e+13 36273907 324465.4  3.6752 0.2350 0.7588
4   1.7003 3079.767  375.3215  -96.8641  4769.455 7.182763e+37 5.600638e+13 30113465 347935.2  4.4270 0.2395 0.7247
5   2.5374 2724.206   69.1071 -133.2133  7427.825 4.192888e+37 4.902335e+13 26433540 359562.1  5.0433 0.2362 0.7495
6   0.0595 2248.236 2370.0926 -167.5654  7556.026 5.757774e+37 4.644758e+13 25772659 362453.9  5.1727 0.2942 0.6601
7   3.1337 3915.362 1061.8154 -119.0358  9583.459 3.698588e+37 1.130488e+13 13710560 404779.0  9.7234 0.2623 0.6711
8  11.7303 4829.156   13.2395  -95.2686 10813.349 3.063319e+37 5.124969e+12  9833383 426054.6 13.5572 0.2452 0.7123
9  10.9825 4246.370   89.4128  -98.6954 10846.012 3.830395e+37 5.066832e+12  9785258 426419.6 13.6239 0.2515 0.8017
10  0.0135 3908.448  729.1802  -99.3776 10978.987 4.501627e+37 4.688061e+12  9470583 428204.5 14.0766 0.2620 0.7798
   Silhouette   Duda  Pseudot2  Beale Ratkowsky       Ball Ptbiserial   Frey McClain   Dunn Hubert SDindex   Dindex   SDbw
2      0.5591 0.5616  846.3109 3.5615    0.4955 22646824.6      0.6433 1.4650  0.3698 0.0160      0  0.0116 109.3617 1.1251
3      0.4612 0.7508  535.0413 1.5145    0.4269 12091302.3      0.6297 0.6796  0.4438 0.0166      0  0.0140  96.7515 0.8801
4      0.3352 0.6540  456.6082 2.4130    0.3806  7528366.2      0.6342 0.0446  0.4943 0.0170      0  0.0135  88.9606 0.5898
5      0.3530 0.3332   80.0524 8.9150    0.3696  5286708.0      0.6452 0.0669  0.4950 0.0174      0  0.0145  85.4785 0.4420
6      0.3531 0.3412 2915.6097 8.8104    0.3386  4295443.2      0.6453 0.9927  0.4950 0.0217      0  0.0128  84.7702 0.3229
7      0.3933 0.3499 1525.3357 8.4727    0.3293  1958651.4      0.5739 0.7964  0.7942 0.0241      0  0.0156  63.3366 0.2643
8      0.3989 0.5080    5.8109 3.7903    0.3121  1229172.8      0.5422 0.2123  0.9076 0.0263      0  0.0189  53.2771 0.2273
9      0.3974 0.6572  114.2427 2.3710    0.2943  1087250.9      0.5422 0.4731  0.9076 0.0270      0  0.0184  53.2026 0.1866
10     0.3903 0.4298 1054.9045 6.0510    0.2795   947058.3      0.5418 1.1309  0.9085 0.0282      0  0.0209  52.6166 0.1591

$All.CriticalValues
   CritValue_Duda CritValue_PseudoT2 Fvalue_Beale
2          0.8602           176.1020       0.0008
3          0.8690           242.9647       0.1571
4          0.8544           147.1042       0.0182
5          0.6609            20.5259       0.0000
6          0.8677           230.2510       0.0000
7          0.8530           141.4975       0.0000
8          0.3404            11.6263       0.0028
9          0.8009            54.4553       0.0207
10         0.8521           138.0087       0.0000

$Best.nc
                    KL       CH Hartigan      CCC    Scott    Marriot       TrCovW       TraceW Friedman    Rubin  Cindex     DB
Number_clusters  2.0000    2.000    6.000   2.0000    5.00 5.000000e+00 3.000000e+00        7     7.00  8.0000  2.000 6.0000
Value_Index     14.0364 5243.068 2300.985 -10.2886 2658.37 4.554761e+37 5.515297e+13  8184923 42325.03 -3.7671  0.193 0.6601
                Silhouette  Duda PseudoT2  Beale Ratkowsky     Ball PtBiserial  Frey McClain    Dunn Hubert SDindex Dindex
Number_clusters     2.0000 8.000   8.0000 3.0000    2.0000        3     6.0000 2.000  2.0000 10.0000      0  2.0000      0
Value_Index         0.5591 0.508   5.8109 1.5145    0.4955 10555522     0.6453 1.465  0.3698  0.0282      0  0.0116      0
                   SDbw
Number_clusters 10.0000
Value_Index      0.1591

$Best.partition
   [1] 1 2 1 2 1 1 2 2 1 1 2 2 2 1 2 2 2 1 1 2 1 1 2 2 1 1 1 2 2 2 2 2 2 2 2 1 1 1 1 2 2 2 1 1 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2
  [61] 2 1 1 2 2 1 1 1 2 2 2 2 1 1 1 1 2 1 2 2 2 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 2 2 1 1 2 2 2 1 1 2 2
 [121] 1 1 2 1 1 1 1 1 2 1 1 2 1 2 1 2 2 2 1 2 1 2 2 1 2 1 1 2 1 1 1 1 1 2 2 1 1 1 2 1 1 1 1 2 2 1 1 2 1 1 1 1 1 2 2 2 1 1 1 1
 [181] 1 2 2 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 1 1 1 2 1 2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 2
 [241] 1 2 1 2 2 2 1 1 1 2 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1
 [301] 1 1 1 1 2 1 2 1 1 1 2 2 1 2 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 2 2 1
 [361] 1 2 1 1 2 1 1 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1 1 1 2 2 2 1 1 2 2 2 2 1 1 2 2
 [421] 1 1 1 1 1 1 2 1 1 2 1 1 2 2 1 1 1 2 1 2 1 1 2 1 2 1 2 2 2 2 2 1 1 1 2 2 2 2 2 1 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 2 1 2 1 2
 [481] 1 1 2 1 1 2 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 2 2 2 1 1 1
 [541] 2 2 1 2 2 2 2 2 1 1 1 1 2 1 1 2 1 1 1 2 2 1 2 2 1 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 1 2 1 1 2 2 2 1 2 2 2 1 1 2 1 2 1
 [601] 2 2 2 2 2 1 2 2 2 2 1 2 1 2 2 1 1 2 2 2 2 1 2 1 2 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 1 1 1 1 1 2 2
 [661] 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 1 1 1 2 2 1 2 2 1 1 1 1 1 2 1 2 1 2 2 2 1 2 2 1 2 2 1 1 2 2 1 1 2 1 1
 [721] 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 1 1 1 1 2 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 2 2 2 2 2 2 1 1 1 1 1 1 1
 [781] 1 1 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 1 1 1 1 2 2 1 2 1 1 2 2 1 1 2 2 2 2 2 2 1 2 2 1 2 2 2 1 1 2 2 2 2 1 1 1 2 2 2 2 2 1 2
 [841] 1 2 2 2 1 1 2 2 1 2 2 1 1 1 1 2 1 1 2 2 2 2 2 2 2 1 1 1 2 2 2 2 1 2 1 1 1 1 2 1 1 2 1 1 1 1 2 2 2 1 1 2 1 1 2 1 1 1 2 1
 [901] 1 1 2 1 1 1 1 1 2 1 1 1 1 2 2 1 2 1 1 1 1 2 1 1 1 1 1 1 2 2 2 2 1 1 2 1 2 1 2 2 2 2 1 1 1 2 2 1 2 2 1 2 1 1 2 2 1 1 2 2
 [961] 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 1 2 2
```

```
[ reached getOption("max.print") -- omitted 1700 entries ]
```



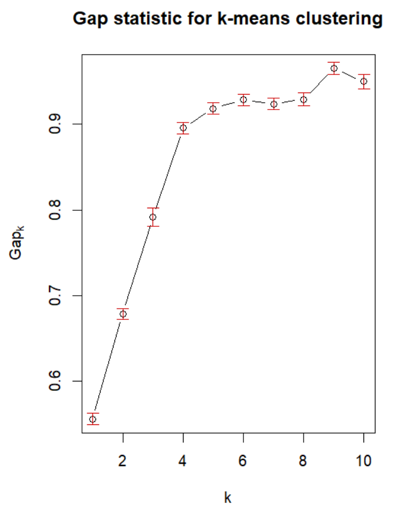
Elbow Method



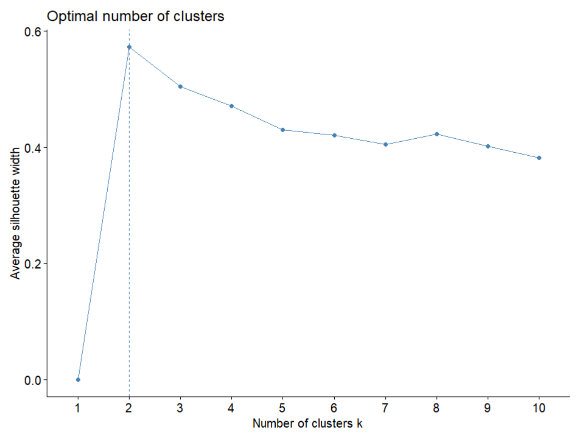
Gap cluster

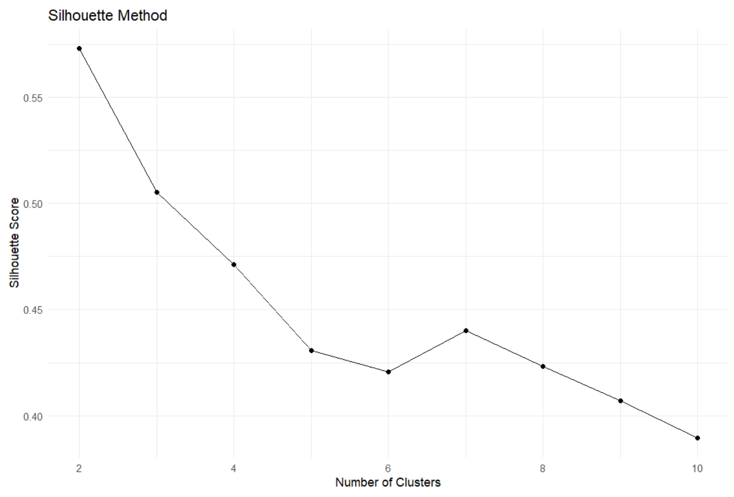
```
# Compute gap statistics with increased number of bootstrap replicates
> gap <- clusGap(transformed_dataset, FUN = function(x, k) kmeans(x, k, iter.max = 100), K.max = 10, B = 100)
Clustering k = 1,2,..., K.max (= 10): .. done
Bootstrapping, b = 1,2,..., B (= 100)  [one "." per sample]:
.................................................. 50
.................................................. 100
>
> # Plot the gap statistics
> plot(gap, main = "Gap statistic for k-means clustering")
```

```
>
> # Determine the optimal number of clusters using maxSE function
> best_clusters <- maxSE(gap$Tab[, "gap"], gap$Tab[, "SE.sim"], method = "Tibs2001SEmax")
>
> # Check contents of best_clusters list
> str(best_clusters)
 int 6
```



Silhouette Method



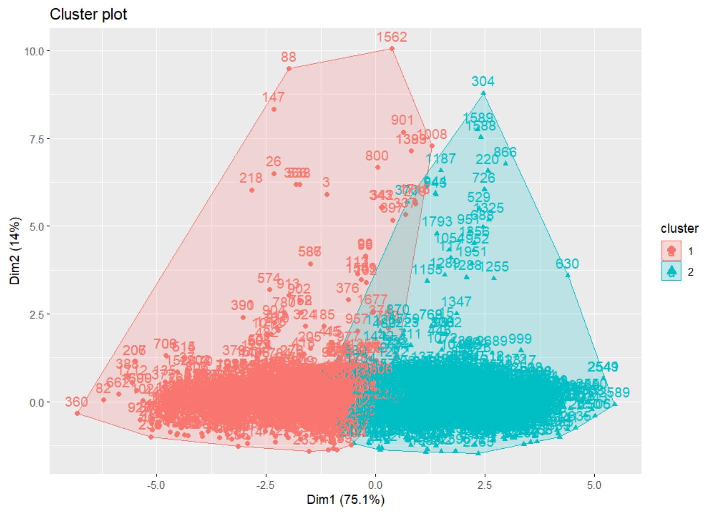

Silhouette Method
Silhouette Score
0.55
0.50
0.45
0.40
2
4
6
8
10
Number of Clusters

For K=2

Output

```
> # Add k=2
> k <- 2
> set.seed(123) # Set the seed for reproducibility
> km <- kmeans(transformed_dataset, centers = k, nstart = 25)
> fviz_cluster(km, data = transformed_dataset)
> print(km)
K-means clustering with 2 clusters of sizes 1205, 1495

Cluster means:
       PC1       PC2       PC3       PC4        PC5        PC6       PC7
1 -167.0028 -6.801375 -16.29043 -29.37453 -12.144097 -102.36826 -44.77182
2  134.6076  5.482045  13.13042  23.67646   9.788386   82.51087  36.08698

Clustering vector:
   [1] 1 2 1 2 1 1 2 2 1 1 2 1 2 1 2 2 2 1 1 2 1 1 2 2 1 1 1 2 2 2 2 2 2 2 2 1 1 1
  [39] 1 1 2 2 2 1 1 2 2 2 1 1 2 1 2 2 2 2 2 2 2 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1
  [77] 1 1 2 2 2 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 2 2 1 1 2
 [115] 1 1 2 1 2 2 1 1 2 1 1 1 1 1 2 1 1 2 1 2 1 1 2 2 1 2 1 2 2 1 2 1 1 2 1 1 1 1
 [153] 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 2 2 1 1 1 1 1 2 2 1 1 1 2 1 1 1
 [191] 1 1 2 2 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 2 2 1 1 2 2 2 1 1 2 1 2 1 2 2 2 1 2
 [229] 1 2 1 1 1 1 1 2 2 1 1 2 1 2 1 2 2 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 2
 [267] 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
 [305] 2 1 2 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
 [343] 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 2 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1
 [381] 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1 1 1 2 2 2 1 1 2 2 1 2 1 1
 [419] 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 2 1 2 1 2 2 2 1 2 1 1 1 2 1
 [457] 2 2 2 1 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1 2 2 1 2 1 1 1 1 1
 [495] 1 2 1 1 1 1 1 1 2 1 1 2 1 1 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1
 [533] 1 1 2 2 2 1 1 1 2 2 1 1 2 2 2 2 1 1 1 1 2 1 1 2 1 1 1 2 1 1 2 2 1 1 1 2 1 2
 [571] 2 2 1 1 2 2 2 2 2 2 1 2 1 1 2 1 1 2 2 2 1 1 1 1 1 1 2 1 2 1 2 2 1 2 2 1 2 2
 [609] 2 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 1 1 2 2 1 2 2 2 2 2 2 1 2 2 1 2 1 2 1 1 1 1
 [647] 2 2 1 2 2 2 2 1 1 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
 [685] 1 1 1 2 2 2 1 2 2 1 1 1 1 1 2 1 2 1 2 2 2 1 1 2 1 1 2 1 1 2 2 1 1 2 1 1 2 2
 [723] 2 2 1 2 2 1 2 2 2 2 2 2 2 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 2 2 2 2 2 1 2 2
 [761] 2 2 1 1 2 1 2 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 1
 [799] 1 1 1 2 1 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 2 1 1 1 2 2 2
 [837] 2 1 1 2 1 2 2 2 1 1 2 2 1 2 2 1 1 1 1 2 1 1 2 1 2 1 2 2 2 2 1 1 2 2 2 2 1 1 2
 [875] 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1
 [913] 1 2 2 1 2 1 1 1 1 2 1 1 1 1 1 1 2 2 2 2 1 1 2 1 2 1 2 2 2 2 2 2 2 1 2 2 1 2 2
 [951] 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 1 2 2 1 2 2 2 2 2 2 2 1 2
 [989] 2 2 2 1 2 1 1 1 1 1 1 2 2
 [ reached getOption("max.print") -- omitted 1700 entries ]

Within cluster sum of squares by cluster:
[1] 21825249 20747870
 (between_SS / total_SS =  68.1 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
> print(km$centers)
       PC1       PC2       PC3       PC4        PC5        PC6       PC7
1 -167.0028 -6.801375 -16.29043 -29.37453 -12.144097 -102.36826 -44.77182
2  134.6076  5.482045  13.13042  23.67646   9.788386   82.51087  36.08698
>
> # Display the BSS/TSS ratio
> BSS <- sum(km$betweenss)
> WSS<-sum(km$tot.withinss)
> TSS <- sum(km$totss)
> BSS_TSS_ratio <- BSS/TSS
> WSS_TSS_ratio <- WSS/TSS
> cat("The ratio of between_cluster_sums_of_squares (BSS) over
+ total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")
The ratio of between_cluster_sums_of_squares (BSS) over
total_sum_of_squares (TSS) is 0.6806542
> cat("The ratio of within_cluster_sums_of_squares WSS) over
+ total_sum_of_squares (TSS) is", WSS_TSS_ratio, "\n")
The ratio of within_cluster_sums_of_squares WSS) over
total_sum_of_squares (TSS) is 0.3193458
>
```

```
> # Display the BSS and WSS indices
> cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")
The between_cluster_sums_of_squares (BSS) index is 90740422
> cat("The within_cluster_sums_of_squares (WSS) index is", WSS, "\n")
The within_cluster_sums_of_squares (WSS) index is 42573119
```

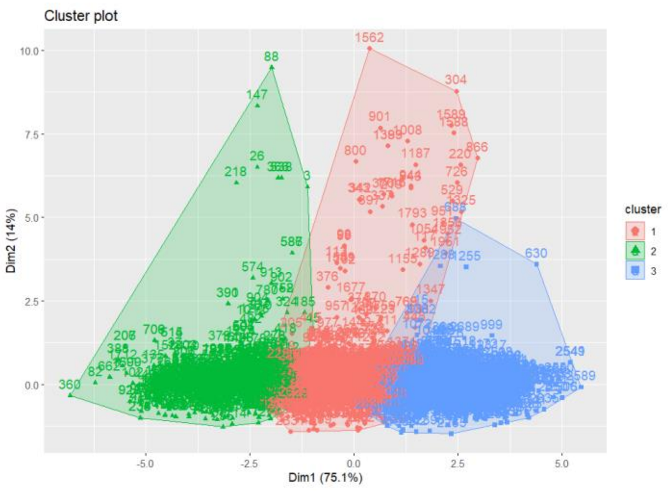


For K = 3

Output

```
# Add k=3
> k <- 3
> set.seed(123) # Set the seed for reproducibility
> km <- kmeans(transformed_dataset, centers = k, nstart = 25)
> fviz_cluster(km, data =transformed_dataset)
> print(km)
K-means clustering with 3 clusters of sizes 1006, 672, 1022

Cluster means:
        PC1       PC2       PC3       PC4        PC5        PC6       PC7
1  -23.44734 -1.018645  -1.786856 -11.37044  -0.1860167  -10.04448  -3.86489
2 -241.19333 -9.461794 -23.609137 -38.44076 -18.2476460 -150.08983 -65.72846
3  181.67313  7.224151  17.282697  36.46854  12.1815567  108.57644  47.02309

Clustering vector:
  [1] 2 1 2 1 2 2 3 1 2 1 1 1 3 1 3 1 1 1 1 1 1 3 3 1 2 1 1 1 3 3 3 3 3 3 2 2 1
 [39] 1 2 3 1 1 2 2 1 3 1 3 1 2 3 2 1 1 1 1 1 1 3 2 2 3 3 1 1 2 1 1 1 1 2 2 2 2
 [77] 1 2 3 3 1 2 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 2 1 1 2 2 3 1 1 1 3
[115] 1 1 1 2 1 1 1 2 3 1 1 2 1 1 3 2 1 1 1 1 2 1 3 1 2 1 2 1 3 2 1 2 2 3 2 2 2 2
[153] 1 1 1 1 2 2 1 2 2 2 2 1 1 2 2 1 2 2 2 2 1 1 1 1 2 1 1 2 1 3 3 2 2 2 3 1 2 1
[191] 2 2 1 1 1 2 2 2 1 1 1 1 1 1 1 2 2 2 1 1 3 3 1 1 3 1 1 2 1 1 1 1 2 3 3 1 1 3
[229] 1 3 2 1 1 2 2 3 3 2 2 1 2 3 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 2 1 2 1 2 2 1
[267] 1 1 2 2 2 2 1 2 1 2 2 2 1 2 1 3 3 2 2 1 2 2 2 2 1 2 2 1 2 1 1 1 2 1 2 1 1 1
[305] 1 2 1 1 1 1 1 1 2 3 3 1 2 2 1 1 1 2 1 2 2 2 1 1 1 3 2 2 2 2 1 1 1 2 2 2 2 1
[343] 1 2 2 1 2 1 2 1 2 1 2 1 1 2 2 1 3 2 2 1 2 2 3 2 2 3 2 1 1 2 1 1 2 1 2 2 2 1
[381] 2 1 1 2 1 2 3 2 2 2 2 1 1 1 1 1 1 2 1 2 1 1 2 2 2 2 2 1 1 1 2 2 1 1 1 1 1 2
[419] 1 3 2 2 2 2 1 2 3 1 2 1 2 2 1 3 2 2 2 1 2 1 2 2 1 2 1 2 3 3 1 1 3 1 1 2 1 1
[457] 1 1 1 2 2 1 1 3 2 1 3 1 3 2 2 2 2 2 2 1 1 3 1 1 2 2 1 2 1 1 1 1 3 2 2 2 2 2
[495] 2 1 1 2 2 2 2 2 1 2 2 3 2 1 1 3 3 3 3 2 2 1 1 1 1 1 1 1 1 2 2 1 3 3 1 2 1 2
[533] 1 1 1 3 3 2 2 1 3 3 2 1 1 1 3 3 2 1 2 2 3 2 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1
[571] 1 1 1 2 1 3 3 1 1 1 1 3 1 1 3 2 2 1 3 3 1 1 1 1 2 2 3 1 1 1 3 3 1 3 3 2 1 3
[609] 1 3 1 1 2 3 3 2 2 1 3 3 3 3 2 1 2 1 3 3 1 3 1 1 1 3 3 2 3 3 2 3 1 3 1 1 1 1
[647] 3 3 2 3 3 3 1 2 2 1 1 2 3 3 2 2 2 2 3 1 2 1 3 3 2 1 3 3 1 2 3 2 2 3 1 1 1 1
[685] 1 1 1 3 1 1 2 3 3 1 1 1 1 2 3 2 3 1 3 1 1 2 1 1 2 1 1 2 2 3 3 2 2 3 2 1 1 1
[723] 1 1 2 1 3 2 3 3 3 3 3 3 3 3 2 2 2 2 3 2 3 2 1 2 1 1 1 1 3 2 3 3 3 3 3 2 1 3
[761] 3 3 2 1 3 1 3 1 1 1 1 1 1 2 2 1 1 1 1 2 1 2 3 2 3 3 1 1 1 1 1 2 3 3 3 3 2 2
```

```
 [799] 2 1 1 3 1 1 3 2 2 3 3 2 2 1 1 3 3 3 1 1 1 1 1 1 3 3 2 2 1 1 3 1 1 1 1 1 1 3
 [837] 3 1 1 1 1 1 3 1 1 1 1 3 3 2 1 1 1 2 2 2 3 2 2 3 1 3 1 3 3 3 1 2 2 3 1 3 1 2 3
 [875] 2 2 2 2 1 2 2 3 2 2 2 1 1 1 3 2 2 1 2 2 1 2 2 2 1 2 1 2 3 2 1 1 1 1 1 1 1 1
 [913] 2 3 1 1 3 2 1 2 2 1 1 1 1 1 1 2 1 1 3 3 1 1 1 1 3 2 1 3 1 3 1 1 1 1 1 1 3 3
 [951] 1 3 1 2 1 3 1 2 1 3 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 3 3 1 1 3 3 3 3 3 1 3
 [989] 3 3 1 2 1 2 2 2 2 2 3 3
 [ reached getOption("max.print") -- omitted 1700 entries ]

Within cluster sum of squares by cluster:
[1] 9094339 6587322 8133321
 (between_SS / total_SS =  82.1 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
> print(km$centers)
        PC1       PC2        PC3       PC4         PC5        PC6       PC7
1  -23.44734 -1.018645  -1.786856 -11.37044  -0.1860167  -10.04448  -3.86489
2 -241.19333 -9.461794 -23.609137 -38.44076 -18.2476460 -150.08983 -65.72846
3  181.67313  7.224151  17.282697  36.46854  12.1815567  108.57644  47.02309
>
> # Display the BSS/TSS ratio
> BSS <- sum(km$betweenss)
> WSS<-sum(km$tot.withinss)
> TSS <- sum(km$totss)
> BSS_TSS_ratio <- BSS/TSS
> WSS_TSS_ratio <- WSS/TSS
>
> cat("The ratio of between_cluster_sums_of_squares (BSS) over
+ total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")
The ratio of between_cluster_sums_of_squares (BSS) over
total_sum_of_squares (TSS) is 0.8213611
> cat("The ratio of within_cluster_sums_of_squares WSS) over
+ total_sum_of_squares (TSS) is", WSS_TSS_ratio, "\n")
The ratio of within_cluster_sums_of_squares WSS) over
total_sum_of_squares (TSS) is 0.1786389
>
> # Display the BSS and WSS indices
> cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")
The between_cluster_sums_of_squares (BSS) index is 109498559
> cat("The within_cluster_sums_of_squares (WSS) index is", WSS, "\n")
The within_cluster_sums_of_squares (WSS) index is 23814982
```

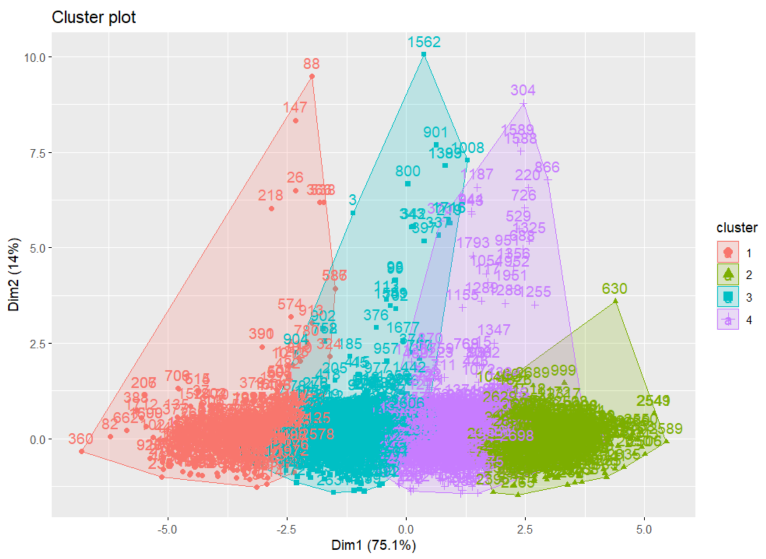


For K = 4

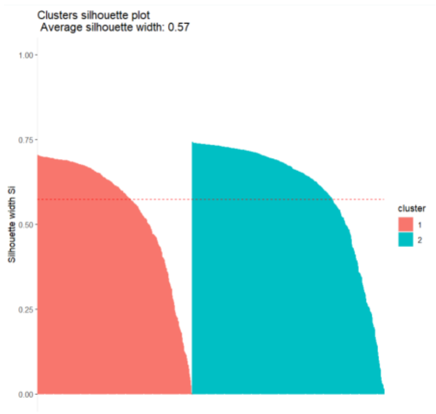
Output

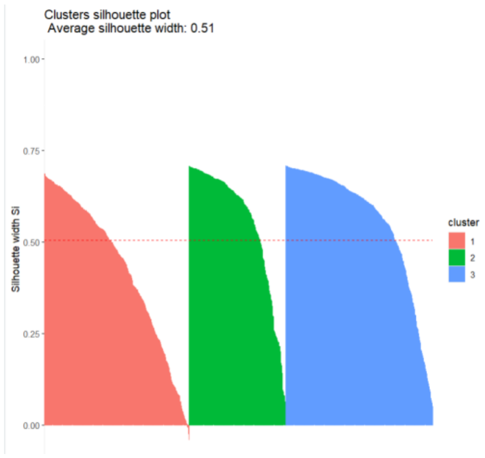
```
# Add k=4
> k <- 4
> set.seed(123) # Set the seed for reproducibility
> km <- kmeans(transformed_dataset, centers = k, nstart = 25)
> fviz_cluster(km, data = transformed_dataset)
> print(km)
K-means clustering with 4 clusters of sizes 529, 584, 740, 847

Cluster means:
        PC1        PC2        PC3        PC4        PC5        PC6       PC7
1 -264.96200 -10.294026 -25.907290 -42.441383 -19.929052 -164.62898 -72.24676
2  228.91312   8.930601  20.842027  46.429379  15.737057  136.67631  59.01978
3  -83.26944  -3.816659  -8.352903 -17.475030  -5.710557  -49.61930 -21.38539
4   80.40026   3.606135   9.107864   9.761813   6.585406   51.93394  23.11237

Clustering vector:
   [1] 3 4 3 4 3 1 4 4 1 3 4 3 4 3 4 4 4 3 3 4 3 3 2 2 3 1 3 4 4 4 4 4 4 4 4 1 1 3
  [39] 3 1 4 4 4 1 3 4 4 4 2 3 1 4 3 4 4 4 4 3 4 4 4 1 1 4 4 3 3 1 3 3 3 4 1 1 1 1
  [77] 3 1 2 2 3 1 3 3 3 4 3 1 4 3 3 3 4 4 3 3 4 3 3 3 3 1 1 3 3 3 3 1 1 4 4 3 3 2
 [115] 3 3 4 1 4 4 3 3 4 3 3 3 3 3 4 1 3 4 3 3 1 3 4 3 1 3 1 3 4 1 3 1 1 4 1 3 1 1
 [153] 3 3 3 3 1 1 4 1 1 1 1 3 4 1 1 4 1 1 1 1 3 3 4 4 3 3 3 3 3 4 4 3 3 3 4 3 1 3
 [191] 1 3 4 4 3 1 1 1 4 4 4 3 3 3 3 1 1 1 3 3 4 4 3 3 4 4 4 1 3 4 3 3 1 4 4 3 3 4
 [229] 3 4 1 3 3 1 1 4 4 1 3 4 3 2 3 4 4 4 3 3 3 3 3 3 3 4 3 3 4 1 1 1 3 1 3 1 1 4
 [267] 3 3 1 1 1 1 4 1 4 3 1 3 3 1 3 4 4 3 3 4 1 1 1 1 3 1 1 3 1 3 3 3 1 3 1 3 3 4
 [305] 4 1 4 3 3 3 4 3 1 4 4 4 1 1 3 3 3 3 3 1 1 1 3 3 3 4 1 1 1 1 3 3 3 1 1 1 1 3
 [343] 3 1 1 3 1 4 1 3 1 3 1 3 3 3 3 3 2 1 1 3 1 1 4 1 1 4 1 4 3 1 3 3 3 3 1 1 1 3
 [381] 1 3 3 1 4 1 4 1 3 1 1 3 3 3 3 3 3 1 4 1 4 4 3 3 1 1 1 4 4 4 1 1 4 3 3 4 3 3
 [419] 3 4 3 1 1 1 3 1 4 3 1 4 1 1 3 4 1 3 3 3 1 3 3 1 4 3 4 1 4 4 4 3 4 3 3 1 4 3
 [457] 4 3 3 1 3 3 3 2 1 3 2 4 4 1 1 1 1 1 3 3 3 4 3 4 1 1 3 1 3 4 4 3 4 1 1 1 1 1
 [495] 1 4 3 1 1 1 1 1 4 1 1 2 1 3 4 4 4 2 2 1 1 3 3 3 3 3 3 3 3 1 1 4 2 4 4 3 3 3
 [533] 3 3 4 2 4 1 1 3 2 4 3 3 4 3 4 4 1 3 3 1 2 1 3 4 3 3 3 4 3 3 4 4 3 3 3 4 3 4
 [571] 4 4 3 1 4 2 2 4 4 3 3 4 3 3 2 1 1 3 2 4 3 3 3 3 1 1 2 3 3 3 2 4 3 2 4 1 4 2
 [609] 4 2 3 4 1 2 2 3 3 4 4 4 4 2 3 4 3 3 2 2 3 2 4 4 4 4 2 1 4 4 1 4 3 2 3 3 3 3
 [647] 2 4 3 4 2 4 4 1 1 3 3 3 4 4 3 1 1 1 2 3 1 3 2 4 1 3 2 2 3 1 4 1 1 2 3 3 3 3
 [685] 3 3 3 4 4 4 3 4 4 3 3 3 3 1 2 1 2 3 4 4 4 1 3 4 1 3 4 1 1 4 2 3 1 2 1 3 4 4
 [723] 4 4 1 4 2 1 4 4 4 4 4 4 4 2 1 1 1 1 2 1 4 1 3 1 3 3 3 3 4 3 4 4 4 4 4 3 4 2
 [761] 4 4 3 3 2 3 2 4 4 3 3 3 3 1 1 3 3 3 3 1 3 3 2 3 4 4 4 4 4 4 3 1 4 4 4 2 1 1
 [799] 1 3 3 2 3 3 2 1 1 4 2 1 1 3 4 4 4 4 3 3 3 4 3 4 2 4 1 1 4 4 4 4 3 3 3 4 4 4
 [837] 2 3 3 4 3 4 4 4 3 3 2 4 1 4 4 3 1 3 3 4 3 1 4 3 4 3 4 4 2 4 1 1 4 4 2 3 1 4
 [875] 1 1 1 1 3 1 1 4 1 1 1 3 4 4 2 1 1 3 1 1 4 1 1 1 3 1 3 3 2 3 3 3 3 3 3 3 3 3
 [913] 1 4 4 3 4 3 3 3 1 4 3 3 3 3 3 1 4 4 4 4 3 3 4 3 2 1 4 4 4 2 4 4 3 4 4 3 4 2
 [951] 4 2 3 1 4 4 3 1 4 4 3 4 4 1 3 1 1 1 1 1 3 1 1 1 1 1 3 2 2 3 4 2 4 4 4 4 3 4
 [989] 2 4 4 3 4 3 1 1 1 1 2 4
 [ reached getOption("max.print") -- omitted 1700 entries ]

Within cluster sum of squares by cluster:
[1] 4243423 2887203 4463655 4643832
 (between_SS / total_SS =  87.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
> print(km$centers)
        PC1        PC2        PC3        PC4        PC5        PC6       PC7
1 -264.96200 -10.294026 -25.907290 -42.441383 -19.929052 -164.62898 -72.24676
2  228.91312   8.930601  20.842027  46.429379  15.737057  136.67631  59.01978
3  -83.26944  -3.816659  -8.352903 -17.475030  -5.710557  -49.61930 -21.38539
4   80.40026   3.606135   9.107864   9.761813   6.585406   51.93394  23.11237
>
> # Display the BSS/TSS ratio
> BSS <- sum(km$betweenss)
> WSS<-sum(km$tot.withinss)
> TSS <- sum(km$totss)
> BSS_TSS_ratio <- BSS/TSS
> WSS_TSS_ratio <- WSS/TSS
> cat("The ratio of between_cluster_sums_of_squares (BSS) over
+ total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")
The ratio of between_cluster_sums_of_squares (BSS) over
total_sum_of_squares (TSS) is 0.8781961
```

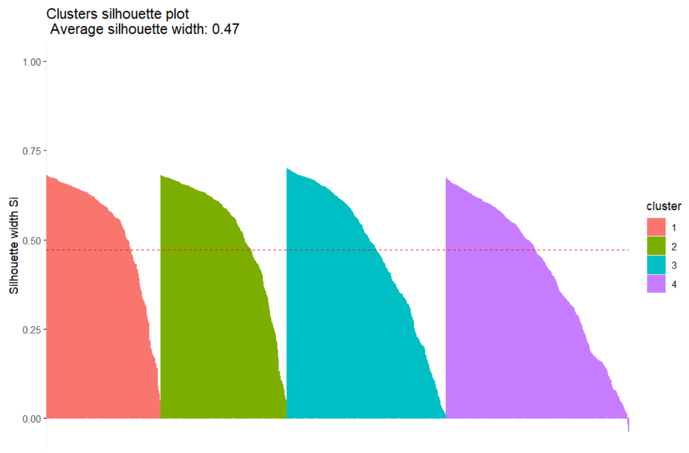
```
> cat("The ratio of within_cluster_sums_of_squares WSS) over
+ total_sum_of_squares (TSS) is", WSS_TSS_ratio, "\n")
The ratio of within_cluster_sums_of_squares WSS) over
total_sum_of_squares (TSS) is 0.1218039
>
> # Display the BSS and WSS indices
> cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")
The between_cluster_sums_of_squares (BSS) index is 117075428
> cat("The within_cluster_sums_of_squares (WSS) index is", WSS, "\n")
The within_cluster_sums_of_squares (WSS) index is 16238113
```



```
# Display the BSS/TSS ratio
> BSS <- sum(km$betweenss)
> WSS<-sum(km$tot.withinss)
> TSS <- sum(km$totss)
> BSS_TSS_ratio <- BSS/TSS
> WSS_TSS_ratio <- WSS/TSS
> cat("The ratio of between_cluster_sums_of_squares (BSS) over
+ total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")
The ratio of between_cluster_sums_of_squares (BSS) over
total_sum_of_squares (TSS) is 0.8781961
> cat("The ratio of within_cluster_sums_of_squares WSS) over
+ total_sum_of_squares (TSS) is", WSS_TSS_ratio, "\n")
The ratio of within_cluster_sums_of_squares WSS) over
total_sum_of_squares (TSS) is 0.1218039
>
> # Display the BSS and WSS indices
> cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")
The between_cluster_sums_of_squares (BSS) index is 117075428
> cat("The within_cluster_sums_of_squares (WSS) index is", WSS, "\n")
The within_cluster_sums_of_squares (WSS) index is 16238113
>
> # Display the silhouette plot
> # Compute silhouette values
> sil <- silhouette(km$cluster, dist(transformed_dataset))
>
> # Add k=2
> k <- 2
> set.seed(123) # Set the seed for reproducibility
> km <- kmeans(transformed_dataset, centers = k, nstart = 25)
> sil_2 <- silhouette(km$cluster, dist(transformed_dataset))
>
> # Add k=3
> k <- 3
> set.seed(123) # Set the seed for reproducibility
> km <- kmeans(transformed_dataset, centers = k, nstart = 25)
> sil_3 <- silhouette(km$cluster, dist(transformed_dataset))
>
> # Add k=4
> k <- 4
> set.seed(123) # Set the seed for reproducibility
> km <- kmeans(transformed_dataset, centers = k, nstart = 25)
> sil_4 <- silhouette(km$cluster, dist(transformed_dataset))
>
> # Calculate average silhouette width for each scenario
> average_silhouette_width_2 <- mean(sil_2[, "sil_width"])
> average_silhouette_width_3 <- mean(sil_3[, "sil_width"])
> average_silhouette_width_4 <- mean(sil_4[, "sil_width"])
>
> # Print average silhouette widths
> cat("Average silhouette width for k=2:", average_silhouette_width_2, "\n")
Average silhouette width for k=2: 0.5729813
> cat("Average silhouette width for k=3:", average_silhouette_width_3, "\n")
Average silhouette width for k=3: 0.5051656
> cat("Average silhouette width for k=4:", average_silhouette_width_4, "\n")
Average silhouette width for k=4: 0.4710818
>
> # Plot silhouette for k=2
> fviz_silhouette(sil_2)
  cluster size ave.sil.width
1       1 1205          0.54
2       2 1495          0.60
>
> # Plot silhouette for k=3
> fviz_silhouette(sil_3)
  cluster size ave.sil.width
1       1 1006          0.42
2       2  672          0.55
3       3 1022          0.56
>
> # Plot silhouette for k=4
> fviz_silhouette(sil_4)
  cluster size ave.sil.width
1       1  529          0.53
2       2  584          0.52
3       3  740          0.46
4       4  847          0.41
```







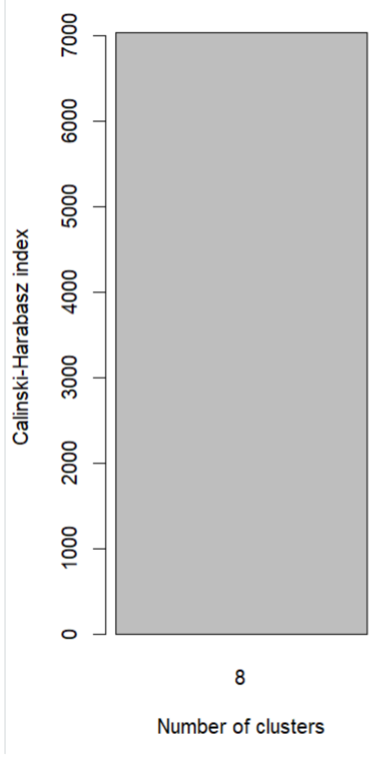
The analysis begins with the examination of the BSS/TSS ratio, revealing a substantial proportion of variance explained by between-cluster differences, with a ratio of 0.878. The BSS and WSS indices further support the notion of well-separated clusters, with BSS at 117,075,428 and WSS at 16,238,113. Evaluating the average silhouette widths for k=2,3, and 44 provides insight into cluster cohesion and separation. For k=2, the average silhouette width is 0.573, indicating clear separation between clusters. Increasing k to 3 and 4 yields average silhouette widths of 0.505 and 0.471, respectively, suggesting reasonable cluster structures. Silhouette plots for each scenario depict the distribution of data points within clusters, affirming the clustering quality. Overall, the analysis supports the conclusion that k=2 yields the most distinct clusters, as evidenced by the superior average silhouette width and well-defined separation.

The Calinski - Harabasz Index Internal Metric Evaluation

Clustering validation is essential for assessing clustering algorithms. We focus here on the Calinski-Harabasz (CH) Index, a measure introduced in 1974. It evaluates clustering quality without known labels, using dataset features. The CH Index quantifies how well objects within a cluster are grouped compared to other clusters. It's calculated using the ratio of separation to cohesion, where separation is the distance between cluster centroids and cohesion is the average distance of data points to their centroid.

The calculation of the Calinski-Harabasz index and then the bar plot graph is illustrated to define the Calinski-Harabasz index vs the number of clusters assigned.

```
# Calculate the Calinski-Harabasz index for 2-10 clusters
> res <- NbClust(transformed_dataset, diss = NULL, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans", index = "ch")
> print(res)
$All.index
       2        3        4        5        6        7        8        9       10
5750.522 6199.867 6479.311 5865.924 6194.741 6808.409 7033.304 6992.658 7024.087

$Best.nc
Number_clusters     Value_Index
        8.000       7033.304

$Best.partition
   [1] 6 5 1 5 6 2 4 5 8 6 5 5 4 6 4 5 5 6 6 4 6 6 7 7 6 2 6 5 4 7 4 7 4 4 7 2 8 5 6 2 4 5 4 8 6 4
  [47] 4 4 7 6 8 7 6 4 5 4 4 5 5 4 4 2 2 4 4 6 6 8 5 5 5 4 2 8 2 2 5 2 7 7 5 8 6 6 6 4 6 1 4 6 5 6
  [93] 5 4 6 6 5 1 1 6 5 8 2 6 6 6 6 8 2 4 5 1 1 7 5 5 1 2 5 4 6 6 4 6 6 6 6 6 4 8 6 5 6 5 8 5 7 5
 [139] 2 5 2 5 4 2 5 2 2 4 2 6 2 8 6 5 5 6 2 2 5 8 8 8 8 5 4 2 8 4 8 8 8 8 6 5 4 4 6 6 6 6 6 4 4 6
 [185] 6 6 4 6 8 6 8 6 4 4 6 8 8 2 5 4 4 6 6 6 6 8 8 2 6 6 4 4 5 5 4 4 4 8 1 1 6 5 2 4 4 5 5 4 5 4
 [231] 2 6 6 8 8 7 7 8 6 4 6 7 6 4 5 5 6 6 6 5 5 6 6 5 6 6 4 2 2 2 6 2 5 2 2 5 5 6 8 8 8 8 4 8 4 6
 [277] 2 6 6 2 6 7 4 6 6 5 2 8 8 2 6 2 8 6 2 5 5 6 8 6 8 6 5 1 4 2 4 6 6 6 5 5 2 7 7 5 2 2 6 5 6 6
 [323] 6 2 2 2 6 6 6 4 2 2 2 2 5 6 1 8 8 8 8 1 1 8 2 6 2 5 2 6 2 6 2 5 6 6 6 5 7 8 2 5 2 2 4 2 8 4
 [369] 2 1 6 2 6 5 6 6 8 2 2 6 8 6 6 2 5 2 4 8 2 2 2 6 6 6 6 6 1 2 4 2 4 4 6 6 8 2 2 5 5 4 2 2 5 5
 [415] 5 4 6 6 5 4 6 8 8 8 5 2 4 6 8 4 8 2 5 4 2 6 6 5 8 5 6 2 4 2 4 8 7 7 4 5 4 6 6 8 4 5 5 5 5 2
 [461] 2 6 6 7 8 6 7 4 7 2 2 8 2 8 6 5 6 7 6 4 2 2 5 2 6 4 4 6 4 8 8 8 8 8 8 4 6 8 2 2 2 2 4 2 2 3
 [507] 2 5 4 4 4 3 3 8 8 6 6 6 6 6 6 6 6 2 2 4 7 4 1 6 5 6 6 6 4 3 7 2 2 6 7 7 6 5 5 5 7 7 2 6 6 2
 [553] 7 2 6 5 6 6 6 5 5 6 4 4 6 6 6 4 5 4 4 4 5 2 5 3 3 4 5 5 6 4 5 6 7 2 2 5 3 7 6 5 5 5 2 2 7 6
 [599] 5 6 3 4 5 7 7 2 4 3 4 7 6 4 8 3 3 6 6 4 7 7 7 7 6 5 6 5 3 7 5 3 5 4 4 7 3 2 7 7 2 7 6 7 6 6
 [645] 6 6 7 7 6 7 7 4 4 2 2 6 6 6 4 4 6 8 8 8 7 6 8 6 3 7 8 6 3 7 6 2 7 2 2 7 6 6 6 6 6 6 6 1 5 5
 [691] 6 4 4 6 6 6 6 2 3 2 3 1 4 5 4 8 5 5 8 5 4 8 2 7 3 6 2 7 8 6 5 5 4 4 8 1 3 2 4 4 4 4 4 4 4 7
 [737] 2 2 2 2 3 2 4 2 6 2 6 6 6 6 4 6 7 7 4 7 4 6 4 7 7 7 6 6 7 6 3 4 4 5 5 5 5 2 2 6 6 6 6 2 6 6
 [783] 7 2 4 4 5 5 5 5 6 2 4 4 4 7 2 2 8 1 6 7 5 6 7 8 8 4 7 2 2 5 4 7 4 4 5 6 5 4 6 4 3 4 8 8 4 4
 [829] 4 4 6 6 6 5 5 4 3 5 5 5 6 4 5 4 6 6 7 7 2 5 5 6 2 6 6 4 6 2 4 5 4 5 7 7 3 1 2 2 4 5 7 5 2 4
 [875] 2 2 2 2 5 2 2 7 2 2 2 6 5 5 3 2 2 5 2 2 4 2 8 8 5 8 1 2 3 6 6 6 6 6 5 6 6 6 2 7 4 5 7 6 6 6
 [921] 8 5 5 6 6 5 5 8 5 5 4 7 5 6 4 6 7 2 4 7 5 7 1 1 6 4 4 6 7 7 1 7 6 2 4 4 5 2 4 4 6 5 4 2 6 2
 [967] 2 2 2 2 6 2 2 2 8 8 6 7 7 6 4 7 4 7 4 7 5 4 3 7 5 6 5 6 2 2 2 2 7 7
 [ reached getOption("max.print") -- omitted 1700 entries ]

> print(res$Best.nc)
Number_clusters     Value_Index
        8.000       7033.304
> # Plot the results
> barplot(res$Best.nc[2], names.arg = res$Best.nc[1], xlab = "Number of clusters", ylab ="Calinski-Harabasz index")
```



Through the use of the k-means algorithm, various cluster counts were evaluated on a given dataset and the Calinski-Harabasz index was used to determine the best solution. The highest value of the index, 7033.304, was achieved with 8 clusters, indicating that this clustering solution yielded the most distinct and well-defined clusters with minimal within-cluster variance and maximal between-cluster variance. By assigning each observation to a specific cluster, the optimal partition provides a clear understanding of the cluster assignments. This analysis confirms that the 8-cluster partitioning effectively captures the data's inherent structure, making it suitable for further analysis and interpretation.

Financial Forecasting Part

Financial forecasting is instrumental in anticipating future exchange rates, a pivotal aspect of global business operations and financial decision-making. Much like weather forecasting, accurate exchange rate predictions empower businesses to plan effectively, optimize currency exchanges, and mitigate financial risks.

Choosing the right input variables for Neural Networks (NNs) in financial forecasting is paramount for precise predictions. These variables, forming the input vector for NNs, significantly influence the model's performance, determining its ability to forecast future exchange rates accurately.
Input variables in financial forecasting encompass a spectrum of factors, including historical exchange rates, economic indicators, market sentiment, and external influences. These variables encapsulate trends, patterns, and factors impacting exchange rate movements. To ensure the forecasting model's precision and reliability, input variables must be meticulously selected and processed. This involves filtering out noise and identifying meaningful patterns through techniques like data cleaning, normalization, and feature engineering.

Commonly employed schemes/methods for defining the input vector in financial forecasting encompass a variety of approaches:

1. AutoRegression Models (AR Models): Leveraging historical exchange rate values as input variables, AR models assume future exchange rate movements depend on past values.

2. Moving Average (MA) Models: Accounting for the influence of previous prediction errors on future exchange rate movements, MA models use a rolling window of historical forecast errors.

3. Time Series Models (ARIMA): Integrating both autoregressive and moving average components, ARIMA models employ differencing to make data stationary before forecasting exchange rates.

4. Seasonal ARIMA (SARIMA) Models: Extending ARIMA models to detect seasonal patterns, SARIMA models utilize seasonal differencing for enhanced forecasting accuracy.

5. ARMA Models: Merging autoregressive and moving average models, ARMA models facilitate statistical analysis and exchange rate forecasting.

6. Exponential Smoothing (ES) Models: Forecasting exchange rates using weighted averages of previous observations, ES models incorporate options to include seasonal patterns and trend elements.

7. Artificial Neural Networks (ANNs): Capable of discerning nonlinear correlations, ANNs utilize historical exchange rate data and external variables for accurate future rate forecasts.

In practice, defining the input vector for NNs in exchange rate forecasting often involves a hybrid approach, combining various methods to capture the market's complex dynamics effectively. The selection of input variables depends on factors such as data availability, forecasting horizon, and data complexity. Each method presents its advantages and drawbacks, necessitating careful consideration based on the specific forecasting requirements. Brownlee, J. (2020, August 28)

For instance, when employing a Multilayer Perceptron Neural Network (MLP-NN) to predict the next day's exchange rate between USD and EUR, the focus typically lies on the autoregressive (AR) approach. In this scenario, past USD/EUR rates serve as crucial input variables for predicting future values. Additionally, lagged variables, technical indicators, fundamental indicators, sentiment analysis, and hybrid models can further enrich the input vector, enhancing the forecasting model's accuracy and reliability.

Performance evaluation metrics such as RMSE, MAE, MAPE, and S-MAPE allow for the comparison of different models, providing insights into how various inputs and neural network structures influence forecasting accuracy.

By judiciously selecting and preprocessing relevant input data, NN models can adeptly anticipate future exchange rate patterns, enabling businesses and financial institutions to optimize currency transactions, reduce costs, and enhance financial decision-making processes.

Use of normalization

In financial forecasting, normalization is a critical step to ensure data is on a common scale. This improves model convergence, prevents bias towards larger-scale variables, enhances performance, and maintains numerical stability. Normalization is an essential process for accurate and efficient predictions in financial models, particularly when using Neural Networks.

Min-Max normalization is a scaling technique used in data preprocessing for machine learning. It scales features so that they fall within a range of 0 to 1. The formula used is (Value - Min)/(Max - Min). It helps with algorithm convergence and is useful for distance-based algorithms. However, it's sensitive to outliers and other scaling methods may be necessary.

Before Normalization	After Normalization

`> # Take only the exchange rate column`
`> dataset <- as.data.frame(data[[3]])`
`> head(dataset)`
`  data[[3]]`
`1    1.3730`
`2    1.3860`
`3    1.3768`
`4    1.3718`
`5    1.3774`
`6    1.3672`
	`> # Normalizing the dataset`
	`> datast_norm <- as.data.frame(lapply(dataset, normalize))`
	`> head(datast_norm)`
	`  data..3..`
	`1 0.7909953`
	`2 0.8526066`
	`3 0.8090047`
	`4 0.7853081`
	`5 0.8118483`
	`6 0.7635071`

The Input/Output Matrix for the MLP training/testing

Input/Output Matrix

T1 Train

```
previous_Day1 t1_expected
<dbl>          <dbl>
1          0.791       0.853
2          0.853       0.809
3          0.809       0.785
4          0.785       0.812
5          0.812       0.764
6          0.764       0.858
```

T1 Test

```
previous_Day1 t1_expected
          <dbl>       <dbl>
1          0.358       0.402
2          0.402       0.381
3          0.381       0.416
4          0.416       0.407
5          0.407       0.415
6          0.415       0.383
```

T2 Train

```
previous_Day2 previous_Day1 t2_expected
          <dbl>          <dbl>       <dbl>
1          0.791          0.853       0.809
2          0.853          0.809       0.785
3          0.809          0.785       0.812
4          0.785          0.812       0.764
5          0.812          0.764       0.858
6          0.764          0.858       0.887
```

T2 Test

```
previous_Day2 previous_Day1 t2_expected
          <dbl>          <dbl>       <dbl>
1          0.358          0.402       0.381
2          0.402          0.381       0.416
3          0.381          0.416       0.407
4          0.416          0.407       0.415
5          0.407          0.415       0.383
6          0.415          0.383       0.418
```

T3 Train

```
previous_Day3 previous_Day2 previous_Day1 t3_expected
          <dbl>          <dbl>          <dbl>       <dbl>
1          0.791          0.853          0.809       0.785
2          0.853          0.809          0.785       0.812
3          0.809          0.785          0.812       0.764
4          0.785          0.812          0.764       0.858
5          0.812          0.764          0.858       0.887
6          0.764          0.858          0.887       0.877
```

T3 Test

```
previous_Day3 previous_Day2 previous_Day1 t3expected
          <dbl>          <dbl>          <dbl>      <dbl>
1          0.358          0.402          0.381      0.416
2          0.402          0.381          0.416      0.407
3          0.381          0.416          0.407      0.415

4          0.416          0.407          0.415      0.383
5          0.407          0.415          0.383      0.418
6          0.415          0.383          0.418      0.465
```

Statistical Indices

RMSE, MAE, MAPE, and S-MAPE are statistical indices commonly used to measure the accuracy of predictive models, especially in regression analysis.

RMSE (Root Mean Square Error)

RMSE is a common measure of precision for machine learning models, particularly neural networks. It calculates the average of the squared differences between predicted and actual values. RMSE is often used as a loss function during training, which aims to reduce the RMSE loss function to make more accurate predictions. RMSE penalizes large errors more severely than small errors and is differentiable, making it compatible with gradient-based optimization techniques.

However, RMSE has some limitations. It is sensitive to outliers and only considers the size of errors. To overcome these limitations, alternative metrics such as MAE and MAPE can be used. These metrics have different attributes and might be more suitable for specific use cases. Therefore, it is crucial to choose the appropriate metric based on the problem and available data. (Statistical Point, No date)

MAE (Mean Absolute Error)

MAE is a statistic used to evaluate machine learning models, especially neural networks. It measures the average difference between predicted and actual values. During training, MAE is used as a loss function to adjust the model's parameters and minimize the difference. MAE is differentiable, making it compatible with gradient-based optimization techniques. However, unlike RMSE, MAE doesn't penalize large errors more heavily than small ones. (Towards Data Science,2019)

MAPE (Mean Absolute Percentage Error)

MAPE is a popular statistic for measuring the accuracy of machine learning models, including neural networks. It calculates the average percentage difference between the predicted and actual values. When training neural network models, MAPE is occasionally employed as a loss function. One benefit of using MAPE as a loss function is that it provides a clear indication of the percentage error of the model's predictions. Another benefit is that it is scale-independent and can be used to compare the accuracy of models that predict data at various scales. However, using MAPE has a drawback in that it is ill-defined when the true value is zero, which can be problematic when working with datasets containing zero values. (Vandeput, N.,2019)

S-MAPE (Symmetric-Mean Absolute Percentage Error)

Symmetric-Mean Absolute Percentage Error (S-MAPE) is an enhanced version of Mean Absolute Percentage Error (MAPE) utilized to assess machine learning models' accuracy, particularly neural networks. S-MAPE is symmetric and indifferent to the direction of the mistake, making it more helpful than MAPE. It is utilized as a loss function during neural network model training, and the aim is to minimize it by adjusting the model's parameters. S-MAPE is equally sensitive to over-

and under-predictions, making it useful in situations like financial forecasting, where both types of predictions have distinct costs.(Sefidian.,2022)

These indices are essential for evaluating the performance of forecasting models, and each has its own advantages and limitations. It’s important to choose the right metric based on the specific context and requirements of the analysis.

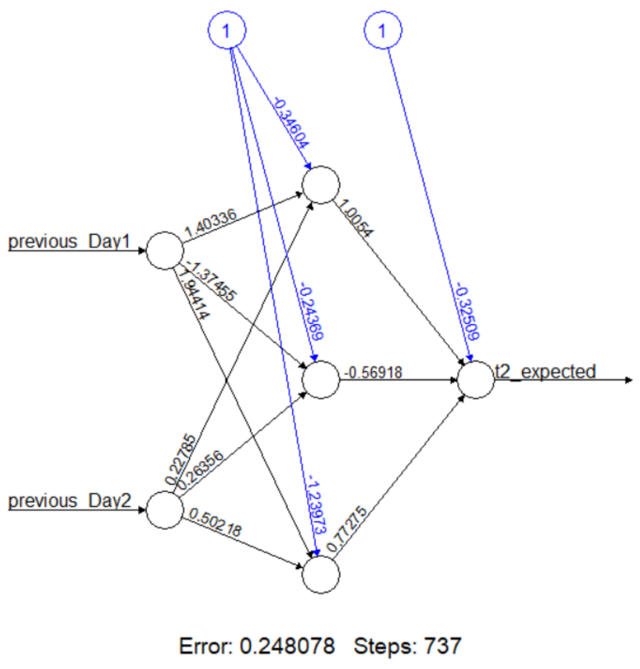
	Hidden Layer Nodes	RMSE	MAE	MAPE	sMape	Description
T1	1	0.00619723	0.004626263	0.3504072	0.3507528	A neural network that has 1 lagged value with 1 hidden layer and 1 hidden layer nodes.
T1	2	0.00613078	0.004608081	0.3490092	0.349305	A neural network that has 1 lagged value with 1 hidden layer and 2 hidden layer nodes.
T1	1, 2	0.0061524	0.004629293	0.350598	0.3509197	A neural network that has 1 lagged value with 2 hidden layers and 1,2 hidden layer nodes.
T1	2, 3	0.006162841	0.004620202	0.3499627	0.3502762	A neural network that has 1 lagged value with 2 hidden layer and 2,3 hidden layer nodes.
T2	2	0.005979327	0.004435714	0.3361654	0.336443	A neural network that has 2 lagged values with 1 hidden layer and 2 hidden layer nodes.
T2	3	0.005680777	0.004279592	0.3241289	0.3244178	A neural network 48 that has 2 lagged values with 1 hidden layer and 3 hidden layer nodes.
T2	2, 3	0.0102101	0.008458163	0.6337816	0.6365208	A neural network that has 2 lagged values with 2 hidden layers and 2,3 hidden layer nodes.

T2	3, 4	0.0138153	0.01132245	0.8498401	0.8540287	A neural network that has 2 lagged values with 2 hidden layers and 3,4 hidden layer nodes.
T3	3	0.006609085	0.004981443	0.3774587	0.3777973	A neural network that has 3 lagged values with 1 hidden layer and 3 hidden layer nodes.
T3	4	0.005745469	0.004314433	0.3268973	0.3272199	A neural network that has 3 lagged values with 1 hidden layer and 4 hidden layer nodes.
T3	3, 4	0.005717346	0.004202062	0.3186009	0.3188649	A neural network that has 3 lagged values with 2 hidden layers and 1,2 hidden layer nodes.
T3	4, 5	0.00537026	0.004056701	0.3073449	0.307644	A neural network that has 3 lagged values with 2 hidden layers and 4,5 hidden layer nodes.

Hidden layer nodes in neural network models impact performance. Metrics like RMSE, MAE, MAPE, and sMAPE help identify weaknesses and improve accuracy. Using plain language, prioritize important information and keep it concise.

The Best Single Hidden Node Layer Neural Network

Based on the analysis previously conducted on various single hidden layer networks, it was determined that the configuration featuring 2-time lagged values and a single hidden layer comprising 3 nodes stands out as the most effective. This conclusion is drawn from the observation that this particular network configuration exhibits lower RMSE, MAE, MAPE, and S-MAPE values compared to other single hidden layer networks analyzed.



Total Number of Learnable Parameters in the Input Layer (I) =0

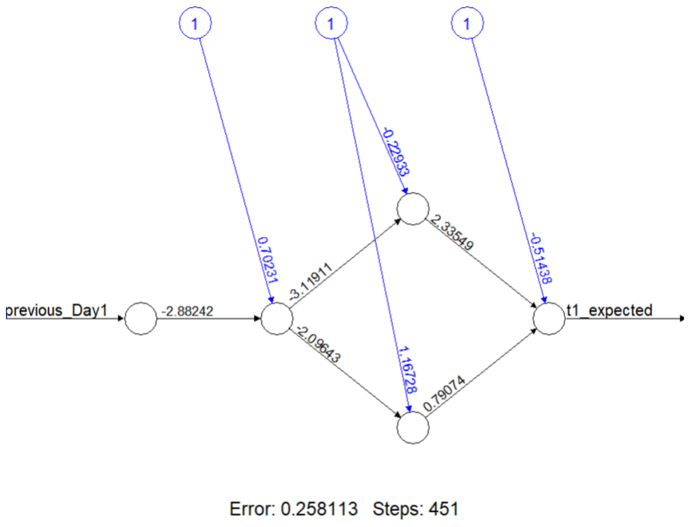
Total Number of Learnable Parameters in the Hidden Single Layer (H) = (2*3) + 3 = 9

Total Number of Learnable Parameters in the Output Layer (O) = (3*1) +1 = 4

Total Number of Learnable Parameters = I + H + O = 0 + 9 + 4 = 13

The Best Double Hidden Node Layers Neural Network

Based on our findings, the double-layer network that performed the best had 2 hidden layers with 1 and 2 nodes and utilized 2-time lagged values. In comparison to the other single-hidden layer networks, it demonstrated the lowest RMSE, MAE, MAPE, and S-MAPE values.



Total Number of Learnable Parameters in the Input Layer (I) =0

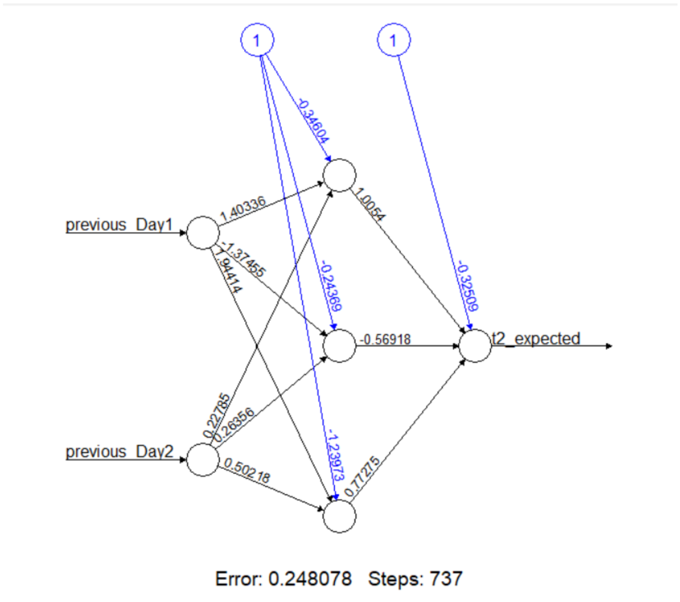
Total Number of Learnable Parameters in the Hidden Single Layer (H) = (1*2) + 2 = 4

Total Number of Learnable Parameters in the Output Layer (O) = (2*1) +1 = 3

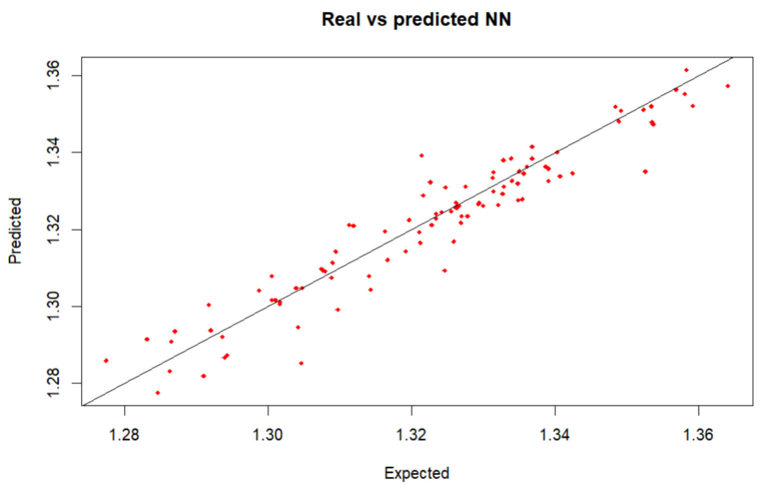
Total Number of Learnable Parameters = I + H + O = 0 + 4 + 3 = 7

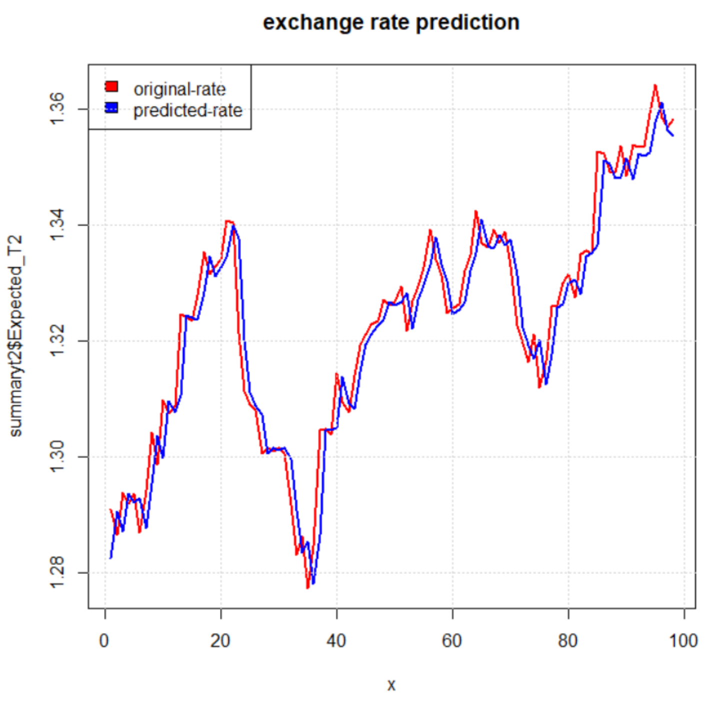
The Best MLP Neural Network

After analyzing various neural network configurations, it has been determined that the model with a single hidden layer containing 3 nodes and using 2-time lagged values is the most effective. This configuration exhibits lower error metrics than other single-layer networks, having a total of 13 learnable parameters. In contrast, the best-performing double hidden layer network has only 7 learnable parameters but does not reach the predictive performance of the single-layer model. These findings highlight the importance of selecting the right network architecture based on specific requirements and show that relatively simple configurations can achieve high accuracy. The single hidden layer network with 3 nodes strikes an optimal balance between complexity, performance, and computational efficiency.



	Hidden Layer Nodes	RMSE	MAE	MAPE	S-MAPE
T2	3	0.005680777	0.004279592	0.3241289	0.3244178


Real vs predicted NN
Predicted
Expected
1.28
1.30
1.32
1.34
1.36


exchange rate prediction
original-rate
predicted-rate
summaryt2$Expected_T2
x
1.28
1.30
1.32
1.34
1.36
0
20
40
60
80
100

Appendix

Partitioning Clustering Part

```
# Load necessary libraries

library(readxl)

library(NbClust)

library(ggplot2)

library(cluster)

library(factoextra)


# View the dataset using read_excel()

wine_data <- read_excel("Whitewine_v6.xlsx")

View(wine_data)


# Select only the first 11 attributes

wine_data <- wine_data[, 1:11]


# Z-score normalization (Standardization)

standardized_features <- scale(wine_data)


# Determine the number of clusters using NbClust

nb_clusters <- NbClust(standardized_features, distance = "euclidean", min.nc = 2, max.nc
= 10, method = "complete", index = "all")

print(nb_clusters)


# Elbow method

wss <- numeric(10)

for (i in 2:10) {
 tryCatch({
  kmeans_model <- kmeans(standardized_features, centers = i, nstart = 25, iter.max = 100)
  wss[i] <- kmeans_model$tot.withinss
 }, warning = function(w) {
  cat("Warning:", conditionMessage(w), "\n")
  wss[i] <- NA
 })
}

# Plot the elbow method
elbow_plot <- data.frame(NumClusters = 2:10, WSS = wss[2:10])
elbow <- ggplot(elbow_plot, aes(x = NumClusters, y = WSS)) +
 geom_point() +
 geom_line() +
 labs(title = "Elbow Method", x = "Number of Clusters", y = "Within-Cluster Sum of Squares
(WSS)") +
 theme_minimal()
fviz_nbclust(standardized_features, kmeans, method = "wss") + labs(subtitle = "Elbow
Method")

print(elbow)

# Compute gap statistics with increased number of bootstrap replicates
gap <- clusGap(standardized_features, FUN = function(x, k) kmeans(x, k, iter.max = 100),
K.max = 10, B = 100)

# Plot the gap statistics
plot(gap, main = "Gap statistic for k-means clustering")
```

```
# Determine the optimal number of clusters using maxSE function
best_clusters <- maxSE(gap$Tab[, "gap"], gap$Tab[, "SE.sim"], method = "Tibs2001SEmax")

# Check contents of best_clusters list
str(best_clusters)

# Silhouette method
# Compute silhouette scores
silhouette <- sapply(2:10, function(k) {
  kmeans_model <- kmeans(standardized_features, centers = k, nstart = 25, iter.max = 100)
# Increase iter.max
  silhouette_avg <- silhouette(kmeans_model$cluster, dist(standardized_features))
  mean(silhouette_avg[, "sil_width"])
})

# Plot silhouette scores
silhouette_plot <- ggplot(data.frame(K = 2:10, Silhouette = silhouette), aes(x = K, y =
Silhouette)) +
  geom_line() +
  geom_point() +
  labs(title = "Silhouette Method", x = "Number of Clusters", y = "Silhouette Score") +
  theme_minimal()
print(silhouette_plot)

# PCA analysis
pca_result <- prcomp(standardized_features, center = TRUE)
print(summary(pca_result))
```

```
# Print eigenvalues and eigenvectors
print(pca_result$sdev) #this print the eigenvalues
print(pca_result$rotation)#this prints the eigenvectors

# Create a plot of the explained variance by each principal component
fviz_eig(pca_result)

# Create a plot of the cumulative variance explained by each principal component
fviz_eig(pca_result, choice = "variance")

# Create a plot of the correlation between variables and principal components
fviz_pca_var(pca_result)

# Calculate cumulative variance explained by each principal component
cumulative_variance <- cumsum(pca_result$sdev^2 / sum(pca_result$sdev^2))

# Find the number of principal components required to achieve cumulative score > 0.85
pc_cutoff <- min(which(cumulative_variance > 0.85))

# Select principal components
selected_pcs <- paste0("PC", 1:pc_cutoff)

# Create a transformed dataset with only the selected principal components as attributes
transformed_dataset <- predict(pca_result, standardized_features)[, 1:pc_cutoff]
colnames(transformed_dataset) <- selected_pcs
```

```
# Print selected PCs

print(selected_pcs)


# Print transformed dataset with selected PCs

head(transformed_dataset)


# Determine the number of clusters using NbClust

nb_clusters <- NbClust(transformed_dataset, distance = "euclidean", min.nc = 2, max.nc =
10, method = "complete", index = "all")

print(nb_clusters)


# Elbow method

wss <- numeric(10)

for (i in 2:10) {

 tryCatch({

  kmeans_model <- kmeans(transformed_dataset, centers = i, nstart = 25, iter.max = 100)

  wss[i] <- kmeans_model$tot.withinss

 }, warning = function(w) {

  cat("Warning:", conditionMessage(w), "\n")

  wss[i] <- NA

 })

}


# Plot the elbow method

elbow_plot <- data.frame(NumClusters = 2:10, WSS = wss[2:10])

elbow <- ggplot(elbow_plot, aes(x = NumClusters, y = WSS)) +

 geom_point() +
 geom_line() +
 labs(title = "Elbow Method", x = "Number of Clusters", y = "Within-Cluster Sum of Squares
(WSS)") +
 theme_minimal()

print(elbow)

# Compute gap statistics with increased number of bootstrap replicates
gap <- clusGap(transformed_dataset, FUN = function(x, k) kmeans(x, k, iter.max = 100),
K.max = 10, B = 100)

# Plot the gap statistics
plot(gap, main = "Gap statistic for k-means clustering")

# Determine the optimal number of clusters using maxSE function
best_clusters <- maxSE(gap$Tab[, "gap"], gap$Tab[, "SE.sim"], method = "Tibs2001SEmax")

# Check contents of best_clusters list
str(best_clusters)

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

# Add k=2
k <- 2
set.seed(123) # Set the seed for reproducibility
km <- kmeans(transformed_dataset, centers = k, nstart = 25)
fviz_cluster(km, data = transformed_dataset)
```

```
print(km)

print(km$centers)


# Display the BSS/TSS ratio

BSS <- sum(km$betweenss)

WSS<-sum(km$tot.withinss)

TSS <- sum(km$totss)

BSS_TSS_ratio <- BSS/TSS

WSS_TSS_ratio <- WSS/TSS

cat("The ratio of between_cluster_sums_of_squares (BSS) over

total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")

cat("The ratio of within_cluster_sums_of_squares WSS) over

total_sum_of_squares (TSS) is", WSS_TSS_ratio, "\n")


# Display the BSS and WSS indices

cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")

cat("The within_cluster_sums_of_squares (WSS) index is", WSS, "\n")



# Add k=3

k <- 3

set.seed(123) # Set the seed for reproducibility

km <- kmeans(transformed_dataset, centers = k, nstart = 25)

fviz_cluster(km, data =transformed_dataset)

print(km)

print(km$centers)
```

```
# Display the BSS/TSS ratio

BSS <- sum(km$betweenss)

WSS<-sum(km$tot.withinss)

TSS <- sum(km$totss)

BSS_TSS_ratio <- BSS/TSS

WSS_TSS_ratio <- WSS/TSS


cat("The ratio of between_cluster_sums_of_squares (BSS) over

total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")

cat("The ratio of within_cluster_sums_of_squares WSS) over

total_sum_of_squares (TSS) is", WSS_TSS_ratio, "\n")


# Display the BSS and WSS indices

cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")

cat("The within_cluster_sums_of_squares (WSS) index is", WSS, "\n")


# Add k= 8

k <- 8

set.seed(123) # Set the seed for reproducibility

km <- kmeans(transformed_dataset, centers = k, nstart = 25)

fviz_cluster(km, data = transformed_dataset)

print(km)

print(km$centers)


# Display the BSS/TSS ratio

BSS <- sum(km$betweenss)

WSS<-sum(km$tot.withinss)
```

```
TSS <- sum(km$totss)
BSS_TSS_ratio <- BSS/TSS
WSS_TSS_ratio <- WSS/TSS
cat("The ratio of between_cluster_sums_of_squares (BSS) over
total_sum_of_squares (TSS) is", BSS_TSS_ratio, "\n")
cat("The ratio of within_cluster_sums_of_squares WSS) over
total_sum_of_squares (TSS) is", WSS_TSS_ratio, "\n")

# Display the BSS and WSS indices
cat("The between_cluster_sums_of_squares (BSS) index is", BSS, "\n")
cat("The within_cluster_sums_of_squares (WSS) index is", WSS, "\n")

# Display the silhouette plot
# Compute silhouette values
sil <- silhouette(km$cluster, dist(transformed_dataset))

# Plot the silhouette
fviz_silhouette(sil)

# Calculate average silhouette width
average_silhouette_width <- mean(sil[, "sil_width"])
cat("Average silhouette width:", average_silhouette_width)

library(fpc) # Load the fpc package

# Calculate the Calinski-Harabasz index for 2-10 clusters
```

```
res <- NbClust(transformed_dataset, diss = NULL, distance = "euclidean", min.nc = 2,
max.nc = 10, method = "kmeans", index = "ch")

print(res)

print(res$Best.nc)


# Plot the results

barplot(res$Best.nc[2], names.arg = res$Best.nc[1], xlab = "Number of clusters", ylab
="Calinski-Harabasz index")
```

Financial Forecasting Part (AR Approach)

```
# Load libraries

library("readxl")

library("dplyr")

library("ggplot2")

library("reshape2")

library("gridExtra")

library("neuralnet")

library("grid")

library("MASS")

library("MLmetrics")


# Import the dataset using the read_excel() function

data <- read_excel("ExchangeUSD.xlsx")

head(dataset)

dim(dataset)


# Take only the exchange rate column

dataset <- as.data.frame(data[[3]])

head(dataset)


# Data Preparation


#Function used to normalize using the Min-max normalization method

normalize <- function(x) {

  return((x - min(x)) / (max(x) - min(x)))

}
```

```
#Function used to de-normalize(Min-max normalization)
unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}

# Normalizing the dataset
datast_norm <- as.data.frame(lapply(dataset, normalize))
head(datast_norm)

# Splitting the normalized data into training and testing sets
data_train <- datast_norm[1:400, ]
data_test <- datast_norm[401:500, ]

# Creating lagged features for 1 lag
t1_train <- bind_cols(previous_Day1 = lag(data_train, 1),
                      t1_expected = data_train)
t1_test <- bind_cols(previous_Day1 = lag(data_test, 1),
                     t1_expected = data_test)

# Removing rows with missing values
t1_train <- t1_train[complete.cases(t1_train), ]
t1_test <- t1_test[complete.cases(t1_test), ]

# Displaying the first few rows of the training dataset with lagged features (1 lag)
head(t1_train)

# Displaying the first few rows of the testing dataset with lagged features (1 lag)
head(t1_test)
```

```
# Creating lagged features for 2 lag
t2_train <- bind_cols(previous_Day2 = lag(data_train, 2),
                previous_Day1 = lag(data_train, 1),
                t2_expected = data_train)
t2_test <- bind_cols(previous_Day2 = lag(data_test, 2),
               previous_Day1 = lag(data_test, 1),
               t2_expected = data_test)



# Removing rows with missing values
t2_train <- t2_train[complete.cases(t2_train), ]
t2_test <- t2_test[complete.cases(t2_test), ]

# Displaying the first few rows of the training dataset with lagged features (2 lags)
head(t2_train)

# Displaying the first few rows of the testing dataset with lagged features (2 lags)
head(t2_test)

# Creating lagged features for 3 lags in the training dataset
t3_train <- bind_cols(previous_Day3 = lag(data_train, 3),
                previous_Day2 = lag(data_train, 2),
                previous_Day1 = lag(data_train, 1),
                t3_expected = data_train)

# Creating lagged features for 3 lags in the testing dataset
t3_test <- bind_cols(previous_Day3 = lag(data_test, 3),
               previous_Day2 = lag(data_test, 2),
               previous_Day1 = lag(data_test, 1),
          t3expected = data_test)

# Removing rows with missing values
t3_train <- t3_train[complete.cases(t3_train), ]
t3_test <- t3_test[complete.cases(t3_test), ]

# Displaying the first few rows of the training dataset with lagged features (3 lags)
head(t3_train)

# Displaying the first few rows of the testing dataset with lagged features (3 lags)
head(t3_test)

# For 1 lagged value
modelt1 <- neuralnet(t1_expected ~ previous_Day1, hidden = c(2, 3),
          data = t1_train, linear.output = TRUE)

# Plotting the neural network model for the 1 lag time horizon
plot(modelt1)

# Displaying the result matrix of the trained neural network model
modelt1$result.matrix

# Making predictions on the test set using the trained model
resultt1 <- compute(modelt1, t1_test[1])

# Extracting normalized predictions from the neural network output
norm_predictiont1 <- resultt1$net.result

# Denormalizing the normalized predictions to obtain actual values
```

```
denorm_predictiont1 <- unnormalize(norm_predictiont1, min(dataset), max(dataset))

denorm_predictiont1 <- round(denorm_predictiont1, 4)


# Denormalizing the test set to obtain actual values

denorm_resultt1 <- unnormalize(t1_test, min(dataset), max(dataset))

denorm_resultt1 <- round(denorm_resultt1, 4)


# Creating a summary table of expected and predicted values

summaryt1 <- cbind(denorm_resultt1[2], denorm_predictiont1)

colnames(summaryt1) <- c("Expected_T1", "Prediction1")


# Displaying the summary table

summaryt1


# Evaluation metrics for t1 lagged values

rmse_t1 <- sqrt(mean((summaryt1$Expected_T1 - summaryt1$Prediction1)^2))

mae_t1 <- mean(abs(summaryt1$Expected_T1 - summaryt1$Prediction1))

mape_t1 <- mean(abs((summaryt1$Expected_T1 - summaryt1$Prediction1) / summaryt1$Expected_T1)) * 100

smape_t1 <- mean(2 * abs(summaryt1$Expected_T1 - summaryt1$Prediction1) / (abs(summaryt1$Expected_T1) + abs(summaryt1$Prediction1))) * 100


# Print evaluation metrics for t1 lagged values

cat("RMSE:", rmse_t1, "\n")

cat("MAE:", mae_t1, "\n")

cat("MAPE:", mape_t1, "\n")

cat("s-MAPE:", smape_t1, "\n")


# For 2 lagged values
```

```
modelt2 <- neuralnet(t2_expected ~ previous_Day1 + previous_Day2, hidden = 2,
          data = t2_train, linear.output = TRUE)

# Plotting the neural network model for the 2 lag time horizon
plot(modelt2)

# Displaying the result matrix of the trained neural network model for 2 lags
modelt2$result.matrix

# Making predictions on the test set using the trained model for 2 lags
resultt2 <- compute(modelt2, t2_test[1:2])

# Extracting normalized predictions from the neural network output
norm_predictiont2 <- resultt2$net.result

# Denormalizing the normalized predictions to obtain actual values
denorm_predictiont2 <- unnormalize(norm_predictiont2, min(dataset), max(dataset))
denorm_predictiont2 <- round(denorm_predictiont2, 4)

# Denormalizing the test set to obtain actual values
denorm_resultt2 <- unnormalize(t2_test, min(dataset), max(dataset))
denorm_resultt2 <- round(denorm_resultt2, 4)

# Creating a summary table of expected and predicted values for 2 lags
summaryt2 <- cbind(denorm_resultt2[2], denorm_predictiont2)
colnames(summaryt2) <- c("Expected_T2", "Prediction2")

# Displaying the summary table for 2 lags
summaryt2
```

```
# Evaluation metrics for t2 lagged values

rmse_t2 <- sqrt(mean((summaryt2$Expected_T2 - summaryt2$Prediction2)^2))

mae_t2 <- mean(abs(summaryt2$Expected_T2 - summaryt2$Prediction2))

mape_t2 <- mean(abs((summaryt2$Expected_T2 - summaryt2$Prediction2) / summaryt2$Expected_T2)) * 100

smape_t2 <- mean(2 * abs(summaryt2$Expected_T2 - summaryt2$Prediction2) / (abs(summaryt2$Expected_T2) + abs(summaryt2$Prediction2))) * 100


# Print evaluation metrics for t2 lagged values

cat("RMSE:", rmse_t2, "\n")

cat("MAE:", mae_t2, "\n")

cat("MAPE:", mape_t2, "\n")

cat("s-MAPE:", smape_t2, "\n")


# For 3 lagged values

modelt3 <- neuralnet(t3_expected ~ previous_Day1 + previous_Day2 + previous_Day3, hidden = c(4, 5),

            data = t3_train, linear.output = TRUE)


# Plotting the neural network model for the 3 lag time horizon

plot(modelt3)


# Displaying the result matrix of the trained neural network model for 3 lags

modelt3$result.matrix



# Making predictions on the test set using the trained model for 3 lags

resultt3 <- compute(modelt3, t3_test[1:3])
```

```
# Extracting normalized predictions from the neural network output

norm_predictiont3 <- resultt3$net.result


# Denormalizing the normalized predictions to obtain actual values

denorm_predictiont3 <- unnormalize(norm_predictiont3, min(dataset), max(dataset))

denorm_predictiont3 <- round(denorm_predictiont3, 4)


# Denormalizing the test set to obtain actual values

denorm_resultt3 <- unnormalize(t3_test, min(dataset), max(dataset))

denorm_resultt3 <- round(denorm_resultt3, 4)


# Creating a summary table of expected and predicted values for 3 lags

summaryt3 <- cbind(denorm_resultt3[2], denorm_predictiont3)

colnames(summaryt3) <- c("Expected_T3", "Prediction3")


# Displaying the summary table for 3 lags

summaryt3


# Evaluation metrics for t3 lagged values

rmse_t3 <- sqrt(mean((summaryt3$Expected_T3 - summaryt3$Prediction3)^2))

mae_t3 <- mean(abs(summaryt3$Expected_T3 - summaryt3$Prediction3))

mape_t3 <- mean(abs((summaryt3$Expected_T3 - summaryt3$Prediction3) / summaryt3$Expected_T3)) * 100

smape_t3 <- mean(2 * abs(summaryt3$Expected_T3 - summaryt3$Prediction3) / (abs(summaryt3$Expected_T3) + abs(summaryt3$Prediction3))) * 100


# Print evaluation metrics for t3 lagged values

cat("RMSE:", rmse_t3, "\n")

cat("MAE:", mae_t3, "\n")
cat("MAPE:", mape_t3, "\n")

cat("s-MAPE:", smape_t3, "\n")


# Scatter Plot of predicted vs. expected values

par(mfrow = c(1, 1))

plot(summaryt2$Expected_T2, summaryt2$Prediction2, col = 'red', main = 'Real vs predicted NN',
pch = 18, cex = 0.7, xlab = "Expected", ylab = "Predicted")

abline(a = 0, b = 1, h = 90, v = 90)


# Line chart of predicted vs. expected values

x <- 1:length(summaryt2$Expected_T2)

plot(x, summaryt2$Expected_T2, col = "red", type = "l", lwd = 2,

    main = "exchange rate prediction")

lines(x, summaryt2$Prediction2, col = "blue", lwd = 2)

legend("topleft", legend = c("original-rate", "predicted-rate"),

      fill = c("red", "blue"), col = 2:3, adj = c(0, 0.6))

grid()
```

References

1. Brownlee, J. (2020) *How to Develop Multilayer Perceptron Models for Time Series Forecasting*. Available at: https://machinelearningmastery.com/how-to-develop-multilayer-perceptron-models-for-time-series-forecasting/ (Accessed: 05 May 2024).

2. Statistical Point. (No date) *MAE vs RMSE*. Available at: https://statisticalpoint.com/mae-vs-rmse/ (Accessed: 06 May 2024).

3. Towards Data Science. (2019) *Forecast KPI: RMSE, MAE, MAPE, Bias*. Available at: https://towardsdatascience.com/forecast-kpi-rmse-mae-mape-bias-cdc5703d242d (Accessed: 06 May 2024).

4. Sefidian. (2022) *A Guide on Regression Error Metrics with Python Code*. Available at: https://sefidian.com/2022/08/18/a-guide-on-regression-error-metrics-with-python-code/ (Accessed: 06 May 2024).

5. Vandeput, N. (2019) *Forecast KPI: RMSE, MAE, MAPE & Bias*. Available at: https://towardsdatascience.com/forecast-kpi-rmse-mae-mape-bias-cdc5703d242d (Accessed: 09 May 2024).