INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

Informatics Institute of Technology

Machine Learning and Data Mining

5DATA001C

Individual Coursework

Name : M.G. Dilhara Manthilina

UOW ID : w1999520

IIT ID : 20221643

Table of Contents

1.Partitioning Clustering Part

1.1Overview

Conventional techniques of evaluating wine quality frequently depend on trained persons' subjective sensory assessments, which may introduce biases and inconsistencies. There is an increasing need for more objective and repeatable methods of assessing and classifying wines as the world's wine market grows and consumer preferences change. The physicochemical characteristics of wine present a viable pathway for these evaluations, furnishing measurable information on characteristics such as acidity, sugar concentrations, alcohol percentage, and additional aspects.

1.2 Materials and Methods

Data Description

The dataset consists of 2700 white wine samples, each of which is identified by 11 unique physicochemical characteristics that were determined through rigorous laboratory research. These characteristics which include density, pH level, sulphate concentration, alcohol content, citric acid content, residual sugar levels, chloride concentration, total and free sulphur dioxide content, and fixed and volatile acidity all work together to help explain the distinctive qualities of each wine. It is imperative to acknowledge the importance of these characteristics since it is thought that they impact the wine's perceived quality.

1.3 Materials and Methods

2700 white wine samples make up the dataset, and each one is unique due to 11 physicochemical characteristics that were determined by laboratory testing. These characteristics include alcohol content, density, pH, sulphate concentration, levels of citric acid, residual sugar content, concentration of chloride, levels of total and free sulphur dioxide, and fixed and volatile acidity. It is essential to appreciate these characteristics in order to understand the wine's profile and perceived quality.

1.4 Data Preprocessing

1.4.1 Scaling

All attributes were normalised to a zero mean and unit variance in order to standardise the dataset and mitigate the effects of different feature scales. By ensuring that every feature contributes equally to the clustering analysis, this normalisation process helps to prevent features with greater numerical ranges from controlling the distance calculations.

1.4.2 Outlier Detection

Because they tamper with the centroid computing process, outliers can skew the results of clustering. The interquartile range (IQR) approach was used for outlier detection. Outliers that were more than 1.5 times the range of the first and third quartiles were found and eliminated from each feature. This method sought to improve the clustering results' dependability.

1.4.3 PCA Reduction

Principal Component Analysis (PCA) was used to decrease the number of variables while maintaining 85% of the total variance because of the high dimensionality of the data. This reduction gets rid of noise and less useful variables, which makes the dataset simpler, less computationally demanding, and frequently improves clustering results. The NbClust software includes several statistical criteria for a thorough assessment and provides a wide variety of indices to help choose the best clustering scheme. For example, the Gap Statistic determines where real data clustering considerably outperforms random clustering by comparing the total intra-cluster variance across various cluster sizes with a null reference distribution.

1.5 Clustering Algorithms

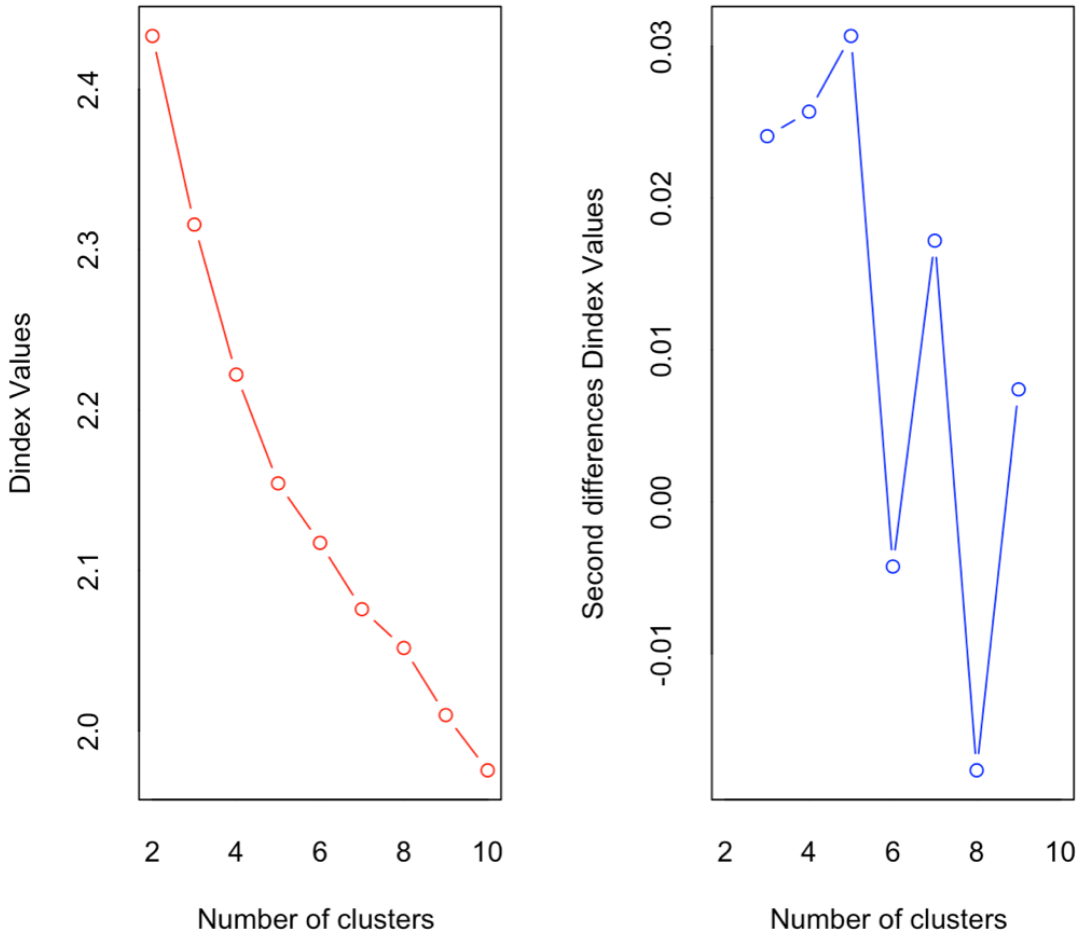
Elbow Method: This method locates the elbow point, which denotes a naturally occurring grouping in the data whereby the sum of squares is not considerably decreased by adding further clusters. This point is found by plotting the within-cluster sum of squares versus the total number of clusters.

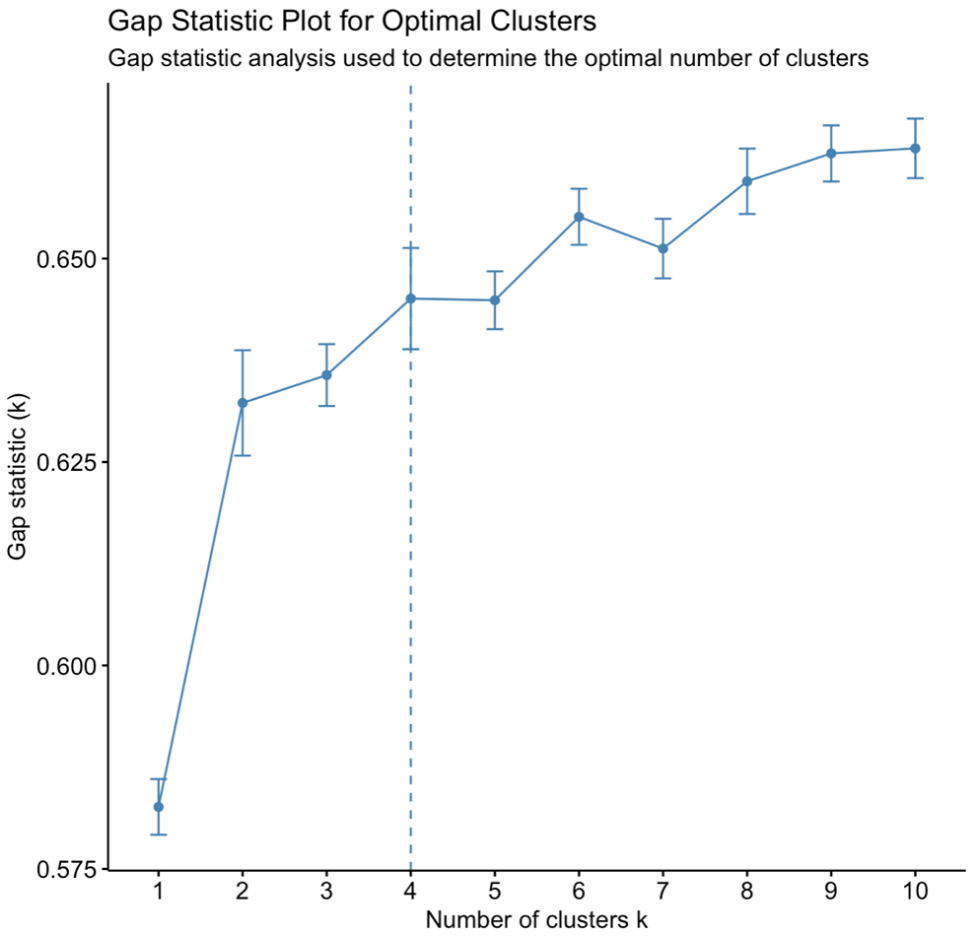
Silhouette Method: This method calculates the average silhouette width for different cluster counts. Greater silhouette values indicate clusters that are more clear and well-defined.

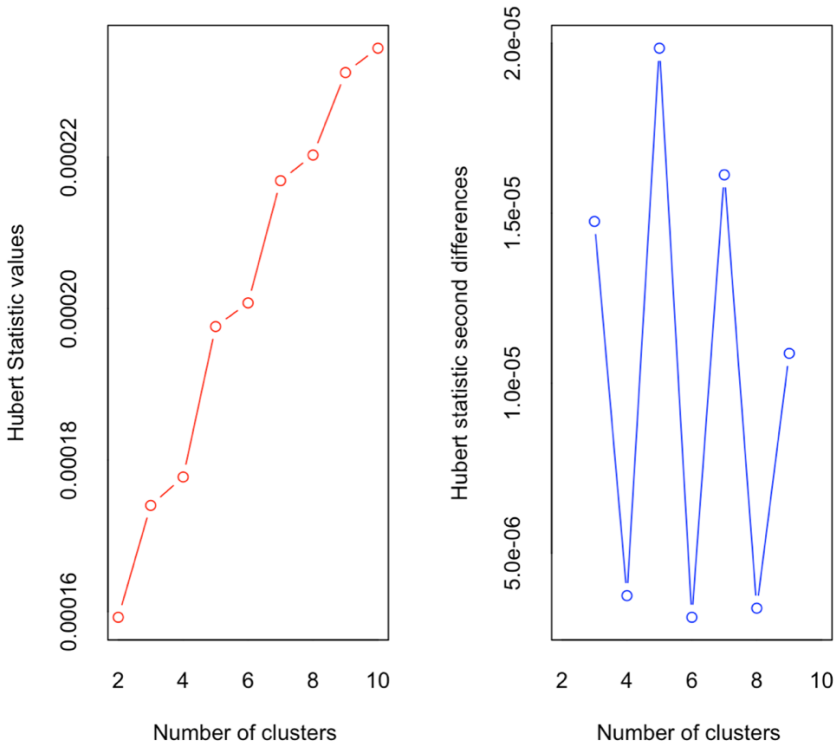
1.5.1 Identification of Clusters

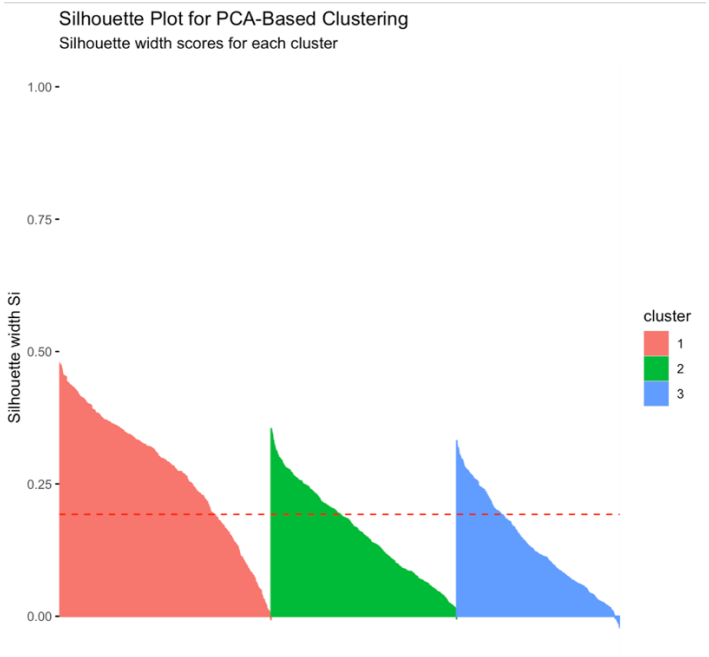
Three clusters are frequently the best option for this dataset, according to the integration of findings from the Elbow technique, Silhouette scores, NbClust, and the Gap Statistic. This result points to a significant split in the dataset, which may be due to natural variations in wine characteristics.

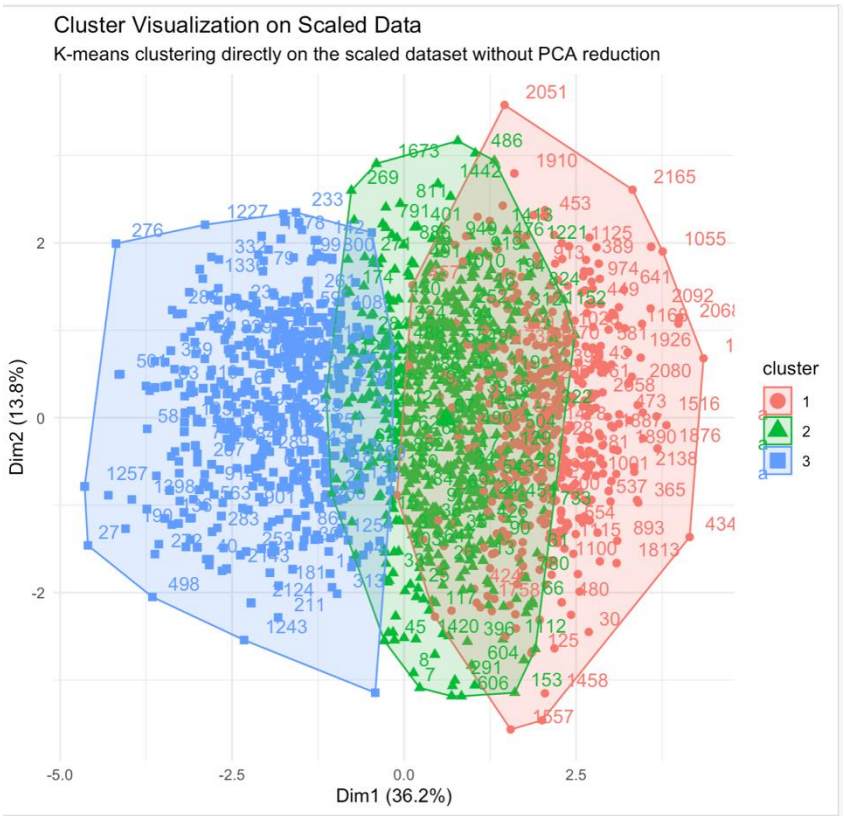
The graphs created for this method are shown below:

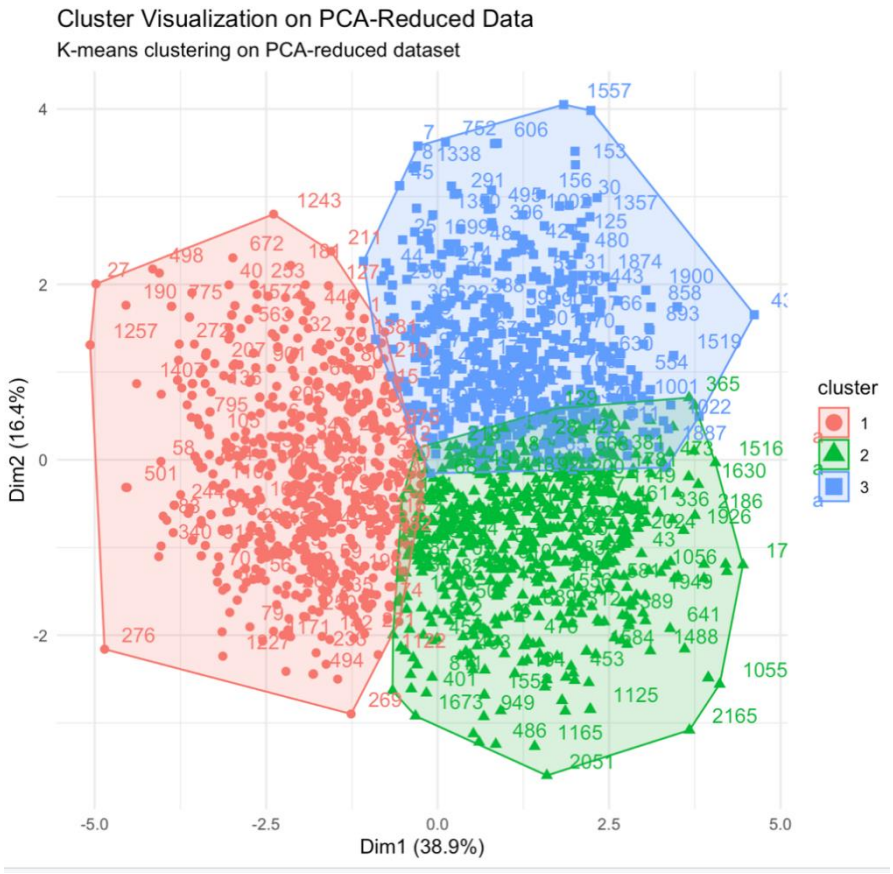


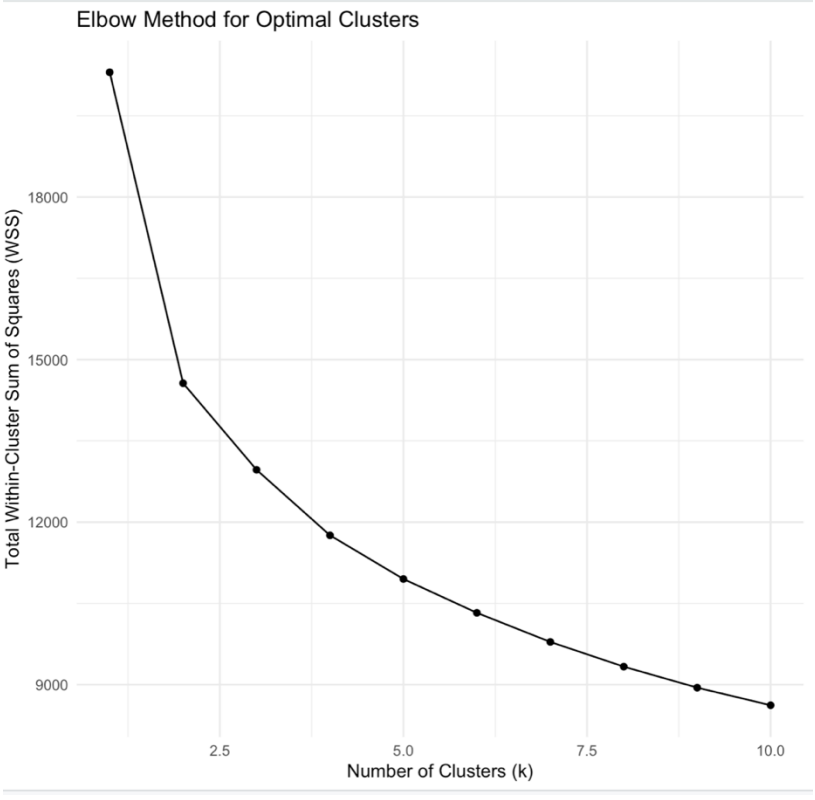

Gap Statistic Plot for Optimal Clusters
Gap statistic analysis used to determine the optimal number of clusters
Gap statistic (k)
0.650
0.625
0.600
0.575
1
2
3
4
5
6
7
8
9
10
Number of clusters k

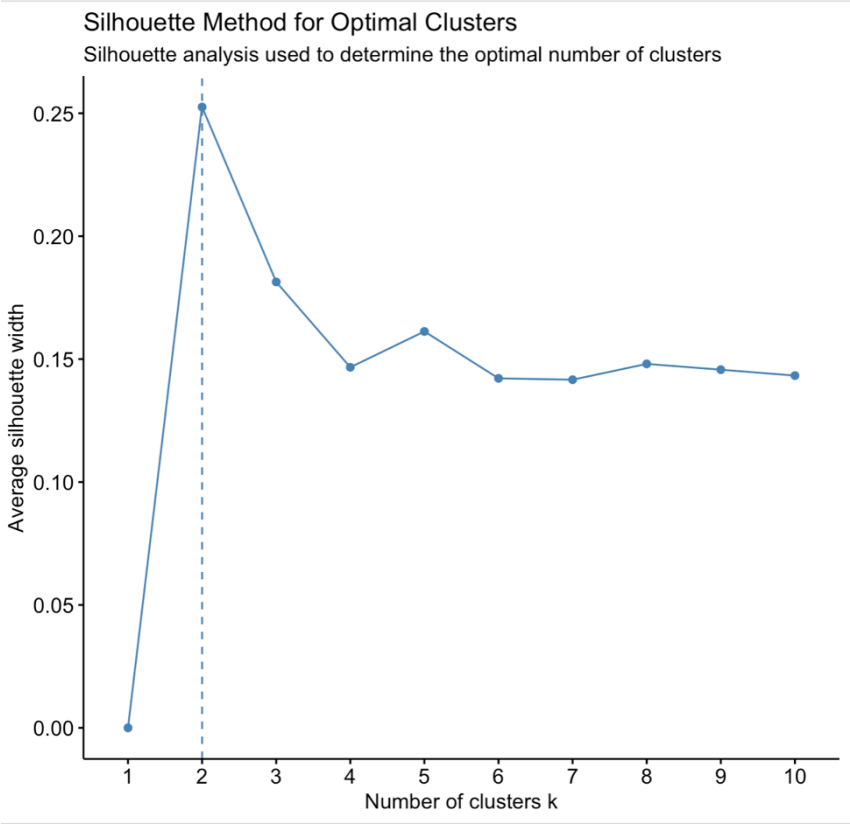

Hubert Statistic values
0.00016
0.00018
0.00020
0.00022
2
4
6
8
10
Number of clusters
Hubert statistic second differences
5.0e-06
1.0e-05
1.5e-05
2.0e-05
2
4
6
8
10
Number of clusters

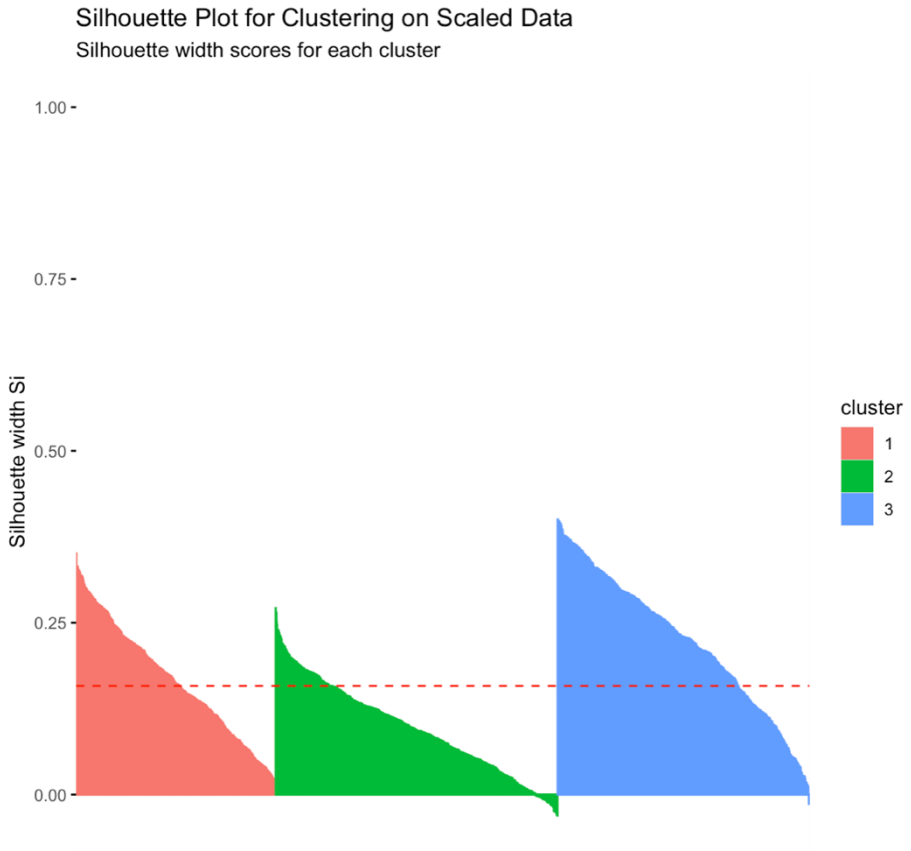

Silhouette Plot for PCA-Based Clustering
Silhouette width scores for each cluster
Silhouette width Si
1.00
0.75
0.50
0.25
0.00
cluster
1
2
3


Cluster Visualization on Scaled Data
K-means clustering directly on the scaled dataset without PCA reduction
Dim2 (13.8%)
Dim1 (36.2%)
2
0
-2
-5.0
-2.5
0.0
2.5
cluster
1
2
3


Cluster Visualization on PCA-Reduced Data
K-means clustering on PCA-reduced dataset
Dim2 (16.4%)
Dim1 (38.9%)
cluster
1
2
3


Elbow Method for Optimal Clusters
Total Within-Cluster Sum of Squares (WSS)
Number of Clusters (k)


Silhouette Method for Optimal Clusters
Silhouette analysis used to determine the optimal number of clusters
Average silhouette width
0.25
0.20
0.15
0.10
0.05
0.00
1
2
3
4
5
6
7
8
9
10
Number of clusters k


Silhouette Plot for Clustering on Scaled Data
Silhouette width scores for each cluster
Silhouette width Si
1.00
0.75
0.50
0.25
0.00
cluster
1
2
3

Section 2: Financial Forecasting

2.1 Introduction

This section explores financial forecasting and highlights how important it is for making well-informed investing decisions. Our goal is to use past data—especially with regard to stock prices to predict future market dynamics, with an emphasis on time series analytic techniques. Through a thorough examination of historical patterns and trends in stock prices, analysts and investors can obtain important insights to improve their investing strategies and efficiently manage risks.

In order to produce projections that support wise financial decision-making, the main goal of this research is to investigate different approaches and techniques for financial forecasting. By conducting a thorough review of time series analysis methods, our goal is to provide stakeholders with the necessary resources and understanding to effectively navigate the constantly changing financial markets.

2.2 Data Preprocessing

The quality of data preparation is critical to the accuracy of financial forecasting models, particularly when working with historical stock price data. A variety of preprocessing methods are used to guarantee the data's dependability and accuracy. First, the data is split into training and testing sets, the latter of which is used to evaluate performance. It is important to handle missing data correctly. Imputation and interpolation are two techniques used to fill in the gaps while maintaining the temporal structure of the time series.

Moreover, by allowing the identification of underlying patterns, seasonality, and trends, turning historical stock price data into a time series object improves forecasting ability. Our goal is to maximise the use of historical data in forecasting model training by applying these preprocessing techniques, which will improve the models' accuracy and usefulness in supporting well-informed financial decision-making.

```r
# Load necessary libraries
library(readxl)
library(nnet)
library(caret)
library(dplyr)
library(ggplot2)

# Load the dataset
exchange_data <- read_excel("ExchangeUSD.xlsx")

# Extract the "USD/EUR" column from exchange_data
exchange_rate <- exchange_data %>% pull(`USD/EUR`)
```

2.3 Time Series Forecasting

We explore sophisticated financial forecasting techniques in this section, emphasising how to use a multi-layer perceptron neural network (MLP-NN) to estimate the USD/EUR exchange rate. To capture the complex dynamics of currency changes, we use an autoregressive (AR) approach with time-delayed exchange rate values as input variables. When building input/output matrices for MLP training and testing, we carefully investigate many input vectors with varying time lags to guarantee thorough coverage of predictive characteristics within historical exchange rate data.

Our goal is to combine the robustness of the autoregressive technique with the flexibility of the MLP-NN architecture to achieve outstanding forecast accuracy. We believe that this strategic combination will provide stakeholders with priceless knowledge, enabling them to make wise choices about investing and currency trading.

```r
# Define Input Variables for MLP Models (Autoregressive Approach)
create_input <- function(data, lag){
  if (!is.vector(data)) {
    stop("Input data must be a vector.")
  }
  lagged_data <- embed(data, lag + 1)
  input <- lagged_data[, -1]
  output <- lagged_data[, 1]
  return(list(input = input, output = output))
}

# Experiment with four input vectors
lag_values <- c(1, 4, 7, 10)  # Choose lag values
input_vectors <- lapply(lag_values, function(lag) create_input(as.vector(train_data), lag))

# Construct Input/Output Matrices for Training and Testing
train_input <- lapply(input_vectors, function(input) input$input)
train_output <- lapply(input_vectors, function(input) input$output)

# Train MLP Models
models <- lapply(input_vectors, function(input) {
  lapply(c(5, 10, 15), function(size) {
    nnet(train_input[[1]], train_output[[1]], size = size, decay = 1e-5, maxit = 1000, linout = TRUE)
  })
})

# Flatten the list of models
models <- unlist(models, recursive = FALSE)

# Evaluate MLP Models
model_evaluation <- lapply(models, function(model) evaluate_model(model, train_input[[1]], train_output[[1]]))
```

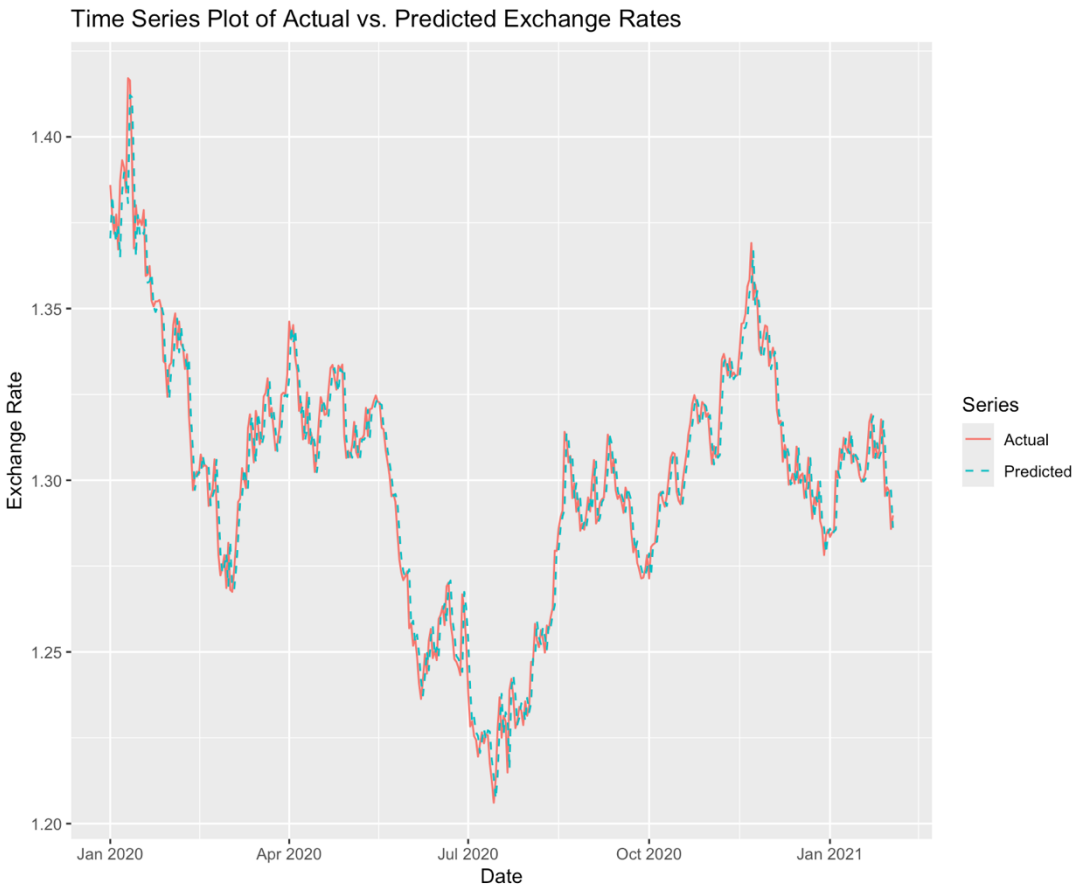
2.4 Evaluation

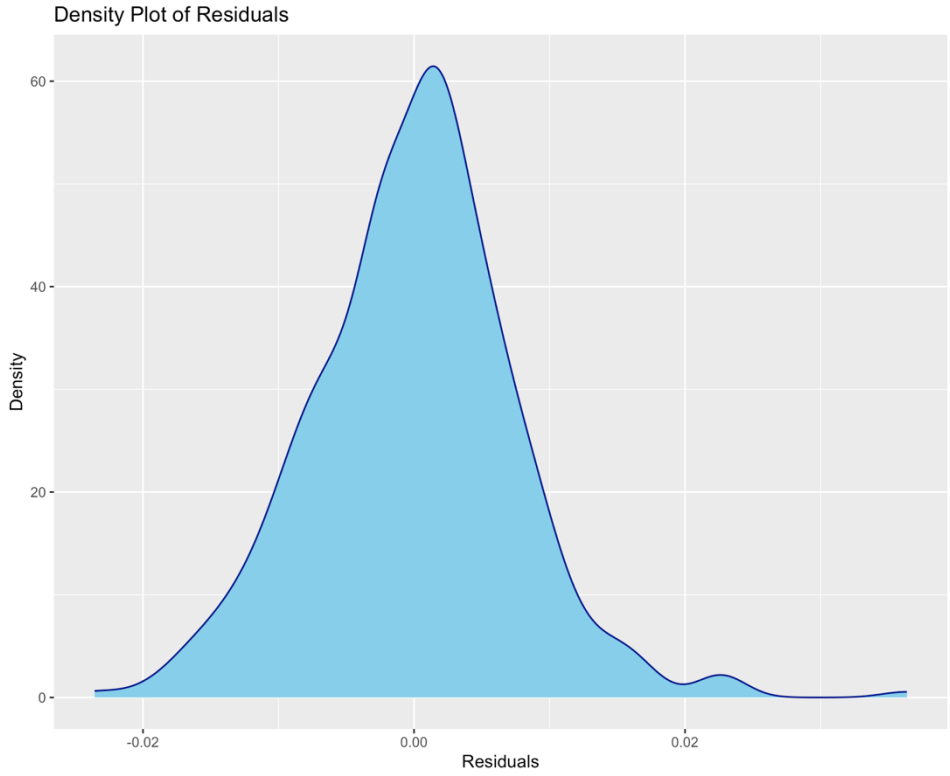
Conventional statistical indices like mean absolute percentage error (MAPE), symmetric MAPE (sMAPE), mean absolute error (MAE), and root mean square error (RMSE) are used to evaluate MLP models. A comparison table is created that summarises the test results of different MLP configurations according to the total number of weight components in order to demonstrate the effectiveness of the best-performing networks.

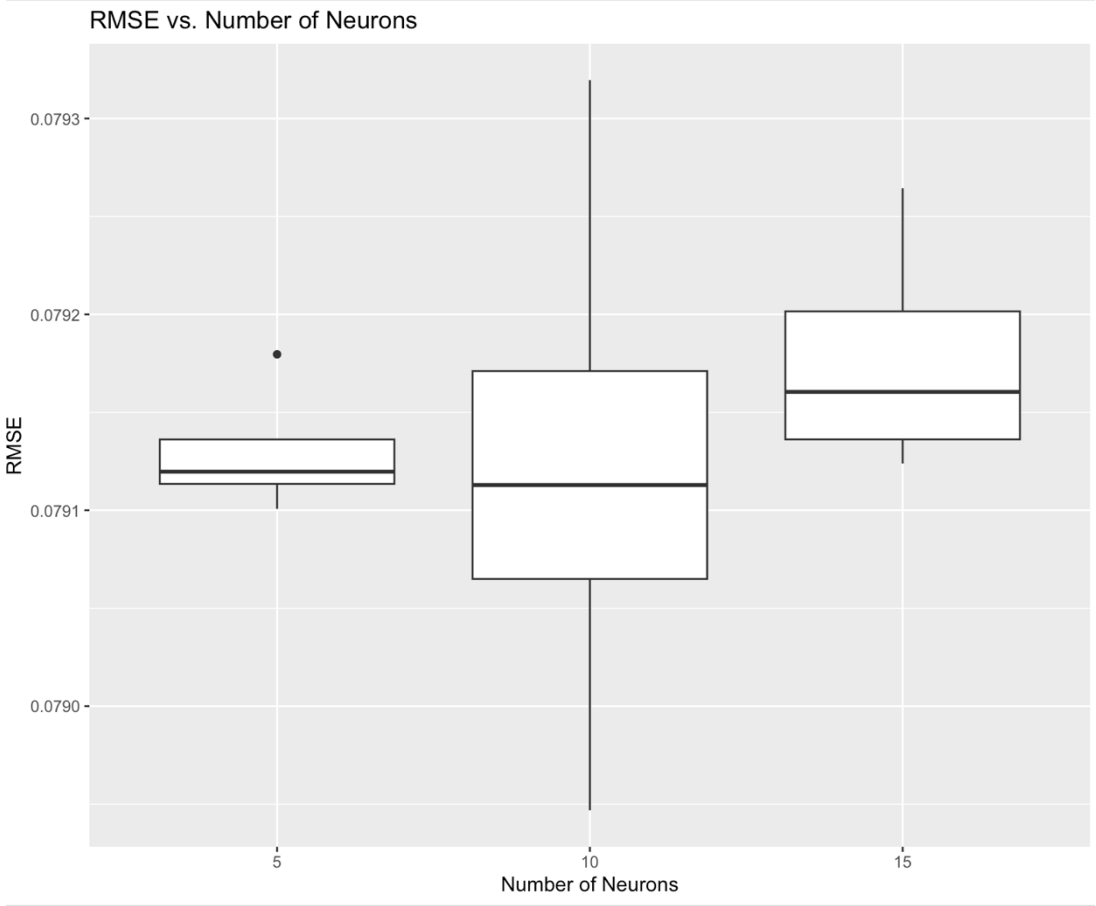
Deep insights into model performance and forecast accuracy can be gained from visualisations such as real vs. anticipated exchange rates, RMSE vs. number of neurons, residual density plots, and time series plots of actual vs. predicted exchange rates. Test performance measures like RMSE, MAE, MAPE, and sMAPE are important factors in determining which model is best to use in order to increase the accuracy of predicting.

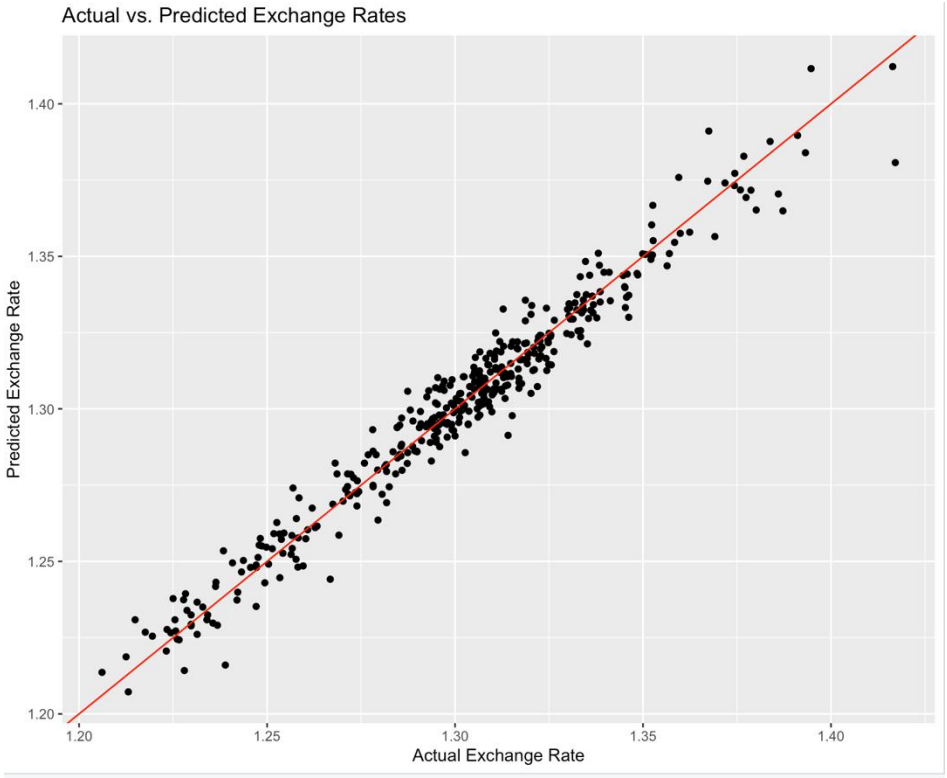
```r
# Evaluate MLP Models
model_evaluation <- lapply(models, function(model) evaluate_model(model, train_input[[1]], train_output[[1]]))

# Performance table
performance_table <- data.frame(
  Model = paste("Model", 1:length(model_evaluation)),
  Neurons = rep(c(5, 10, 15), each = 4),
  RMSE = sapply(model_evaluation, function(x) x$rmse),
  MAE = sapply(model_evaluation, function(x) x$mae),
  MAPE = sapply(model_evaluation, function(x) x$mape),
  sMAPE = sapply(model_evaluation, function(x) x$smape)
)

# Print performance table
print(performance_table)
```

This section's generated plots are shown below




Density Plot of Residuals
Density
60
40
20
0
-0.02
0.00
0.02
Residuals


RMSE vs. Number of Neurons
RMSE
0.0793
0.0792
0.0791
0.0790
5
10
15
Number of Neurons



2.5 Conclusion

Financial forecasting using MLP neural networks provides insightful information about possible trends and patterns in exchange rates. Analysts can determine which MLP models are the most accurate and dependable for forecasting by comparing them. This makes it possible for people to make well-informed financial decisions based on reliable prediction models, improving their capacity to confidently and precisely negotiate the complexities of currency markets.

Overall Conclusion

This research concludes with two separate but related domains: financial time series forecasting and white wine variety segmentation and clustering.

Sophisticated clustering techniques like principal component analysis and k-means have been very helpful in the field of wine analysis to reveal the intrinsic relationships between different chemical aspects of white wines. By means of thorough preprocessing and comprehensive investigation of clustering approaches, we have acquired distinctive understandings of the dataset's fundamental structures. This has improved our understanding of wine varietals and how they are categorised according to chemical profiles, which has improved oenological research and informed wine industry decision-making.

Within the finance domain, our research has proficiently utilised time series forecasting methodologies, specifically utilising a multi-layer perceptron neural network, to anticipate forthcoming trends in USD/EUR exchange rates. By utilising historical data and cutting-edge modelling techniques, we hope to provide stakeholders with actionable forecast insights so they can make wise decisions in the middle of the unstable financial markets.

We have carefully considered the technical nuances of the used approaches as well as the wider ramifications for industry stakeholders throughout our investigation. The findings shared here have concrete implications and open up new avenues for investigation and useful applications in these dynamic environments. Our research helps to improve knowledge and practical implementation of insights learned, which benefits everyone from winemakers looking to expand their product offers to investors navigating the complexity of global currency markets.

Appendix

The code of part1

```
# Load necessary libraries
library(readxl)
library(dplyr)
library(ggplot2)
library(factoextra)
library(cluster)
library(stats)
library(data.table)
library(NbClust)  # Add this line to load the NbClust package

# Load the dataset
wine_data <- read_excel("Whitewine_v6.xlsx")

# Selecting the first 11 chemical attributes
wine_data <- select(wine_data, 1:11)

# Data Preprocessing
## Scaling the data
wine_scaled <- scale(wine_data)

# Outlier Detection
outliers <- c()
for (i in 1:11) {
  quantile1 <- quantile(wine_scaled[, i], probs = 0.25)
  quantile3 <- quantile(wine_scaled[, i], probs = 0.75)
  iqr <- quantile3 - quantile1
  bottom_outlier_rows <- which(wine_scaled[, i] < quantile1 - 1.5 * iqr)
  top_outlier_rows <- which(wine_scaled[, i] > quantile3 + 1.5 * iqr)
  outliers <- c(outliers, top_outlier_rows, bottom_outlier_rows)
}
outliers <- unique(outliers)
```

```
wine_cleaned <- wine_scaled[-outliers, ]

# PCA Analysis
pca_result <- prcomp(wine_cleaned, center = TRUE, scale. = TRUE)

# Cumulative Variance
cum_var <- cumsum(pca_result$sdev^2 / sum(pca_result$sdev^2)) * 100

# Select PCs with cumulative variance > 85%
pca_selected <- pca_result$x[, cum_var <= 85]

# Determining the number of clusters using silhouette method
silhouette_plot <- fviz_nbclust(pca_selected, kmeans, method = "silhouette") +
  labs(title = "Silhouette Method for Optimal Clusters",
       subtitle = "Silhouette analysis used to determine the optimal number of clusters")
print(silhouette_plot)

# Elbow Method for Optimal Clusters
total_within_ss <- function(k, data) {
  kmeans_model <- kmeans(data, centers = k, nstart = 25)
  return(kmeans_model$tot.withinss)
}

# Compute total within-cluster sum of squares for different values of k
k_values <- 1:10  # Try different values of k
wss_values <- sapply(k_values, function(k) total_within_ss(k, pca_selected))

# Plotting the Elbow Method
elbow_plot <- ggplot(data = data.frame(k = k_values, WSS = wss_values), aes(x = k, y =
WSS)) +
  geom_line() +
  geom_point() +
  labs(title = "Elbow Method for Optimal Clusters",
       x = "Number of Clusters (k)",
```

```
      y = "Total Within-Cluster Sum of Squares (WSS)") +
  theme_minimal()
print(elbow_plot)

# Assuming optimal number of clusters from visual inspection or prior analysis
optimal_clusters <- 3  # Example, adjust based on actual analysis

# Clustering on PCA-reduced dataset
set.seed(123)
pca_kmeans_result <- kmeans(pca_selected, centers = optimal_clusters, nstart = 25)
print(pca_kmeans_result$centers)

# Enhanced cluster visualization to avoid overlap on PCA-reduced data
cluster_plot_pca <- fviz_cluster(pca_kmeans_result, data = pca_selected, stand = FALSE,
                                 geom = "point", pointsize = 2, alpha.point = 0.8) +
  geom_text(aes(label = 1:nrow(pca_selected), color = factor(pca_kmeans_result$cluster)),
            check_overlap = TRUE, vjust = -0.5, hjust = -0.5) +
  labs(title = "Cluster Visualization on PCA-Reduced Data",
       subtitle = "K-means clustering on PCA-reduced dataset") +
  theme_minimal()
print(cluster_plot_pca)

# Silhouette plot for PCA-based clustering
pca_silhouette_scores <- silhouette(pca_kmeans_result$cluster, dist(pca_selected))
silhouette_scores_plot <- fviz_silhouette(pca_silhouette_scores) +
  labs(title = "Silhouette Plot for PCA-Based Clustering",
       subtitle = "Silhouette width scores for each cluster")
print(silhouette_scores_plot)

# Clustering directly on the scaled data without PCA
kmeans_result_scaled <- kmeans(wine_cleaned, centers = optimal_clusters, nstart = 25)

# Enhanced cluster visualization for the scaled data without PCA
```

```
cluster_plot_scaled <- fviz_cluster(kmeans_result_scaled, data = wine_cleaned, stand = FALSE,
                                    geom = "point", pointsize = 2, alpha.point = 0.8) +
  geom_text(aes(label = 1:nrow(wine_cleaned), color = factor(kmeans_result_scaled$cluster)),
            check_overlap = TRUE, vjust = -0.5, hjust = -0.5) +
  labs(title = "Cluster Visualization on Scaled Data",
       subtitle = "K-means clustering directly on the scaled dataset without PCA reduction") +
  theme_minimal()
print(cluster_plot_scaled)

# Additional Plots
# Silhouette plot for PCA-based clustering on the scaled data
silhouette_scores_scaled <- silhouette(kmeans_result_scaled$cluster, dist(wine_cleaned))
silhouette_scores_plot_scaled <- fviz_silhouette(silhouette_scores_scaled) +
  labs(title = "Silhouette Plot for Clustering on Scaled Data",
       subtitle = "Silhouette width scores for each cluster")
print(silhouette_scores_plot_scaled)

# Gap Statistic Plot
gap_stats <- clusGap(wine_cleaned, FUN = kmeans, K.max = 10, B = 100)
gap_stat_plot <- fviz_gap_stat(gap_stats) +
  labs(title = "Gap Statistic Plot for Optimal Clusters",
       subtitle = "Gap statistic analysis used to determine the optimal number of clusters")
print(gap_stat_plot)

# NbClust Plot
nb <- NbClust(wine_cleaned, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
nbclust_plot <- barplot(table(nb$Best.nc), xlab = "Number of Clusters", ylab = "Votes", main = "NbClust Results")
print(nbclust_plot)

# PCA Scree Plot
fviz_eig(pca_result, addlabels = TRUE, ylim = c(0, 100))
```

The code of part 2

```
# Load necessary libraries
library(readxl)
library(nnet)
library(caret)
library(dplyr)
library(ggplot2)

# Load the dataset
exchange_data <- read_excel("ExchangeUSD.xlsx")

# Extract the "USD/EUR" column from exchange_data
exchange_rate <- exchange_data %>% pull(`USD/EUR`)

# Split dataset into training and testing sets
train_data <- exchange_rate[1:400]
test_data <- exchange_rate[401:length(exchange_rate)]

# Define Input Variables for MLP Models (Autoregressive Approach)
create_input <- function(data, lag){
  if (!is.vector(data)) {
    stop("Input data must be a vector.")
  }
  lagged_data <- embed(data, lag + 1)
  input <- lagged_data[, -1]
  output <- lagged_data[, 1]
  return(list(input = input, output = output))
}

# Experiment with four input vectors
lag_values <- c(1, 4, 7, 10)  # Choose lag values
input_vectors <- lapply(lag_values, function(lag) create_input(as.vector(train_data), lag))
```

```
# Construct Input/Output Matrices for Training and Testing
train_input <- lapply(input_vectors, function(input) input$input)
train_output <- lapply(input_vectors, function(input) input$output)

# Train MLP Models
models <- lapply(input_vectors, function(input) {
  lapply(c(5, 10, 15), function(size) {
    nnet(train_input[[1]], train_output[[1]], size = size, decay = 1e-5, maxit = 1000, linout = TRUE)
  })
})

# Flatten the list of models
models <- unlist(models, recursive = FALSE)

# Evaluate MLP Models
model_evaluation <- lapply(models, function(model) evaluate_model(model, train_input[[1]], train_output[[1]]))

# Performance table
performance_table <- data.frame(
  Model = paste("Model", 1:length(model_evaluation)),
  Neurons = rep(c(5, 10, 15), each = 4),
  RMSE = sapply(model_evaluation, function(x) x$rmse),
  MAE = sapply(model_evaluation, function(x) x$mae),
  MAPE = sapply(model_evaluation, function(x) x$mape),
  sMAPE = sapply(model_evaluation, function(x) x$smape)
)

# Print performance table
print(performance_table)

# Plot Actual vs. Predicted Exchange Rates
predicted_values <- predict(models[[1]], as.matrix(train_input[[1]]))
```

```
actual_vs_predicted <- data.frame(Actual = train_output[[1]], Predicted = predicted_values)

ggplot(actual_vs_predicted, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  labs(x = "Actual Exchange Rate", y = "Predicted Exchange Rate") +
  ggtitle("Actual vs. Predicted Exchange Rates")

# Plot RMSE vs. Number of Neurons
rmse_vs_neurons <- data.frame(Neurons = rep(c(5, 10, 15), each = 4), RMSE =
performance_table$RMSE)

ggplot(rmse_vs_neurons, aes(x = factor(Neurons), y = RMSE)) +
  geom_boxplot() +
  labs(x = "Number of Neurons", y = "RMSE") +
  ggtitle("RMSE vs. Number of Neurons")

# Density Plot of Residuals
residuals <- actual_vs_predicted$Actual - actual_vs_predicted$Predicted
ggplot(data.frame(Residuals = residuals), aes(x = Residuals)) +
  geom_density(fill = "skyblue", color = "darkblue") +
  labs(x = "Residuals", y = "Density") +
  ggtitle("Density Plot of Residuals")

# Time Series Plot of Actual vs. Predicted Exchange Rates
exchange_ts <- data.frame(
  Date = seq(as.Date("2020-01-01"), by = "day", length.out = length(train_output[[1]])),
  Actual = train_output[[1]],
  Predicted = predicted_values
)

ggplot(exchange_ts, aes(x = Date)) +
  geom_line(aes(y = Actual, color = "Actual")) +
  geom_line(aes(y = Predicted, color = "Predicted"), linetype = "dashed") +
```

```
  labs(x = "Date", y = "Exchange Rate", color = "Series") +
  ggtitle("Time Series Plot of Actual vs. Predicted Exchange Rates")

# Print test performance metrics
test_rmse <- sapply(models, function(model) {
  predicted_values <- predict(model, as.matrix(train_input[[1]]))
  sqrt(mean((predicted_values - train_output[[1]])^2))
})
test_mae <- sapply(models, function(model) {
  predicted_values <- predict(model, as.matrix(train_input[[1]]))
  mean(abs(predicted_values - train_output[[1]]))
})
test_mape <- sapply(models, function(model) {
  predicted_values <- predict(model, as.matrix(train_input[[1]]))
  mean(abs((predicted_values - train_output[[1]]) / train_output[[1]])) * 100
})
test_smape <- sapply(models, function(model) {
  predicted_values <- predict(model, as.matrix(train_input[[1]]))
  mean(2 * abs(predicted_values - train_output[[1]]) / (abs(predicted_values) +
abs(train_output[[1]]))) * 100
})

# Print test performance metrics
cat("Test RMSE:", test_rmse, "\n")
cat("Test MAE:", test_mae, "\n")
cat("Test MAPE:", test_mape, "\n")
cat("Test sMAPE:", test_smape, "\n")

# Find the index of the best model based on minimum RMSE
best_model_index <- which.min(test_rmse)

# Check if models list is not empty
if (length(models) > 0) {
  # Check if best_model_index is within bounds
```

```
  if (best_model_index > 0 && best_model_index <= length(models)) {
    # Access the best model
    best_model <- models[[best_model_index]]

    # Print the best model's details
    print(best_model)
  } else {
    cat("best_model_index is out of bounds.\n")
  }
} else {
  cat("No models found.\n")
}
```

Console output of part1

```
> # Load necessary libraries
> library(readxl)
> library(dplyr)
> library(ggplot2)
> library(factoextra)
> library(cluster)
> library(stats)
> library(data.table)
>
> # Load the dataset
> wine_data <- read_excel("Whitewine_v6.xlsx")
>
> # Selecting the first 11 chemical attributes
> wine_data <- select(wine_data, 1:11)
>
> # Data Preprocessing
> ## Scaling the data
> wine_scaled <- scale(wine_data)
>
> # Outlier Detection
> outliers <- c()
> for (i in 1:11) {
+   quantile1 <- quantile(wine_scaled[, i], probs = 0.25)
+   quantile3 <- quantile(wine_scaled[, i], probs = 0.75)
+   iqr <- quantile3 - quantile1
+   bottom_outlier_rows <- which(wine_scaled[, i] < quantile1 - 1.5 * iqr)
+   top_outlier_rows <- which(wine_scaled[, i] > quantile3 + 1.5 * iqr)
+   outliers <- c(outliers, top_outlier_rows, bottom_outlier_rows)
+ }
> outliers <- unique(outliers)
> wine_cleaned <- wine_scaled[-outliers, ]
>
> # PCA Analysis
> pca_result <- prcomp(wine_cleaned, center = TRUE, scale. = TRUE)
>
> # Cumulative Variance
> cum_var <- cumsum(pca_result$sdev^2 / sum(pca_result$sdev^2)) * 100
>
> # Select PCs with cumulative variance > 85%
> pca_selected <- pca_result$x[, cum_var <= 85]
>
> # Determining the number of clusters using silhouette method
> silhouette_plot <- fviz_nbclust(pca_selected, kmeans, method = "silhouette") +
```

```
> silhouette_plot <- fviz_nbclust(pca_selected, kmeans, method = "silhouette") +
+   labs(title = "Silhouette Method for Optimal Clusters",
+        subtitle = "Silhouette analysis used to determine the optimal number of clusters")
> print(silhouette_plot)
>
> # Elbow Method for Optimal Clusters
> total_within_ss <- function(k, data) {
+   kmeans_model <- kmeans(data, centers = k, nstart = 25)
+   return(kmeans_model$tot.withinss)
+ }
>
> # Compute total within-cluster sum of squares for different values of k
> k_values <- 1:10  # Try different values of k
> wss_values <- sapply(k_values, function(k) total_within_ss(k, pca_selected))
>
> # Plotting the Elbow Method
> elbow_plot <- ggplot(data = data.frame(k = k_values, WSS = wss_values), aes(x = k, y = WS
S)) +
+   geom_line() +
+   geom_point() +
+   labs(title = "Elbow Method for Optimal Clusters",
+        x = "Number of Clusters (k)",
+        y = "Total Within-Cluster Sum of Squares (WSS)") +
+   theme_minimal()
> print(elbow_plot)
>
> # Assuming optimal number of clusters from visual inspection or prior analysis
> optimal_clusters <- 3  # Example, adjust based on actual analysis
>
> # Clustering on PCA-reduced dataset
> set.seed(123)
> pca_kmeans_result <- kmeans(pca_selected, centers = optimal_clusters, nstart = 25)
> print(pca_kmeans_result$centers)
        PC1         PC2         PC3         PC4         PC5          PC6
1 -2.031158 -0.1346161 -0.08990609  0.06847423 -0.01536381 -0.001373542
2  1.336321 -0.9083672  0.23316396 -0.17601828 -0.09706545 -0.010606956
3  1.117206  1.2083786 -0.14854589  0.11135594  0.13039531  0.013852576
>
> # Enhanced cluster visualization to avoid overlap on PCA-reduced data
> cluster_plot_pca <- fviz_cluster(pca_kmeans_result, data = pca_selected, stand = FALSE,
+                                  geom = "point", pointsize = 2, alpha.point = 0.8) +
+   geom_text(aes(label = 1:nrow(pca_selected), color = factor(pca_kmeans_result$cluster)),
+             check_overlap = TRUE, vjust = -0.5, hjust = -0.5) +
+   labs(title = "Cluster Visualization on PCA-Reduced Data",
+        subtitle = "K-means clustering on PCA-reduced dataset") +
+   theme_minimal()
> print(cluster_plot_pca)
>
> # Silhouette plot for PCA-based clustering
> pca_silhouette_scores <- silhouette(pca_kmeans_result$cluster, dist(pca_selected))
> silhouette_scores_plot <- fviz_silhouette(pca_silhouette_scores) +
+   labs(title = "Silhouette Plot for PCA-Based Clustering",
+        subtitle = "Silhouette width scores for each cluster")
  cluster size ave.sil.width
1       1  848          0.27
2       2  743          0.16
3       3  653          0.13
> print(silhouette_scores_plot)
>
> # Clustering directly on the scaled data without PCA
> kmeans_result_scaled <- kmeans(wine_cleaned, centers = optimal_clusters, nstart = 25)
>
> # Enhanced cluster visualization for the scaled data without PCA
> cluster_plot_scaled <- fviz_cluster(kmeans_result_scaled, data = wine_cleaned, stand = FA
LSE,
+                                    geom = "point", pointsize = 2, alpha.point = 0.8) +
+   geom_text(aes(label = 1:nrow(wine_cleaned), color = factor(kmeans_result_scaled$cluste
r)),
+             check_overlap = TRUE, vjust = -0.5, hjust = -0.5) +
+   labs(title = "Cluster Visualization on Scaled Data",
+        subtitle = "K-means clustering directly on the scaled dataset without PCA reductio
n") +
+   theme_minimal()
> print(cluster_plot_scaled)
>
```

console output part2

```
iter 150 value 0.022393
iter 160 value 0.022388
iter 170 value 0.022380
iter 180 value 0.022371
iter 190 value 0.022368
iter 200 value 0.022367
final  value 0.022366
converged
>
> # Flatten the list of models
> models <- unlist(models, recursive = FALSE)
>
> # Evaluate MLP Models
> model_evaluation <- lapply(models, function(model) evaluate_model(model, train_input[[1]], train_output[[1]]))
There were 50 or more warnings (use warnings() to see the first 50)
>

>
> # Performance table
> performance_table <- data.frame(
+   Model = paste("Model", 1:length(model_evaluation)),
+   Neurons = rep(c(5, 10, 15), each = 4),
+   RMSE = sapply(model_evaluation, function(x) x$rmse),
+   MAE = sapply(model_evaluation, function(x) x$mae),
+   MAPE = sapply(model_evaluation, function(x) x$mape),
+   sMAPE = sapply(model_evaluation, function(x) x$smape)
+ )
>
> # Print performance table
> print(performance_table)
       Model Neurons       RMSE        MAE     MAPE    sMAPE
1   Model 1       5 0.07911144 0.07095408 5.535856 5.349623
2   Model 2       5 0.07911370 0.07095644 5.536039 5.349795
3   Model 3       5 0.07915071 0.07099508 5.539028 5.352612
4   Model 4       5 0.07914230 0.07098630 5.538349 5.351972
5   Model 5      10 0.07907571 0.07091677 5.532970 5.346903
6   Model 6      10 0.07912754 0.07097089 5.537157 5.350848
7   Model 7      10 0.07912390 0.07096709 5.536863 5.350571
8   Model 8      10 0.07912867 0.07097207 5.537248 5.350934
9   Model 9      15 0.07911804 0.07096097 5.536390 5.350125
10 Model 10      15 0.07912104 0.07096410 5.536632 5.350353
11 Model 11      15 0.07907473 0.07091574 5.532891 5.346828
12 Model 12      15 0.07910824 0.07095074 5.535598 5.349379
>
> # Plot Actual vs. Predicted Exchange Rates
> predicted_values <- predict(models[[1]], as.matrix(train_input[[1]]))
> actual_vs_predicted <- data.frame(Actual = train_output[[1]], Predicted = predicted_values)
>
> ggplot(actual_vs_predicted, aes(x = Actual, y = Predicted)) +
+   geom_point() +
+   geom_abline(intercept = 0, slope = 1, color = "red") +
+   labs(x = "Actual Exchange Rate", y = "Predicted Exchange Rate") +
+   ggtitle("Actual vs. Predicted Exchange Rates")
>
> # Plot RMSE vs. Number of Neurons
```

```
>
> # Print test performance metrics
> cat("Test RMSE:", test_rmse, "\n")
Test RMSE: 0.007469994 0.007470501 0.007470683 0.007469589 0.007470587 0.007470374 0.007470461 0.007470564 0.0074
70101 0.007470223 0.007470664 0.007470626
> cat("Test MAE:", test_mae, "\n")
Test MAE: 0.005689525 0.005689713 0.005690105 0.005689632 0.005689403 0.005689795 0.005689821 0.005689892 0.00568
9633 0.005689717 0.00568934 0.005689686
> cat("Test MAPE:", test_mape, "\n")
Test MAPE: 0.436929 0.436944 0.4369753 0.4369376 0.4369193 0.4369506 0.4369526 0.4369582 0.4369376 0.4369443 0.43
69145 0.4369419
> cat("Test sMAPE:", test_smape, "\n")
Test sMAPE: 0.4369544 0.4369692 0.4369993 0.4369621 0.4369456 0.4369753 0.4369774 0.4369829 0.4369627 0.4369692
0.4369411 0.4369674
>
> # Find the index of the best model based on minimum RMSE
> best_model_index <- which.min(test_rmse)
>
> # Check if models list is not empty
> if (length(models) > 0) {
+   # Check if best_model_index is within bounds
+   if (best_model_index > 0 && best_model_index <= length(models)) {
+     # Access the best model
+     best_model <- models[[best_model_index]]
+
+     # Print the best model's details
+     print(best_model)
+   } else {
+     cat("best_model_index is out of bounds.\n")
+   }
+ } else {
+   cat("No models found.\n")
+ }
a 1-5-1 network with 16 weights
options were - linear output units  decay=1e-05
>
```