National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Punya Nandakumar Kedambadi
Student ID: X23134658

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Punya Nandakumar Kedambadi

Student ID: X23134658

Programme: Data Analytics **Year:** 2024-2025

Module: MSc Research Project

Lecturer: Hicham Rifai

Submission Due Date: 29/01/2025

Project Title: Tailoring Customer Engagement: Advanced Segmentation for Growth in the Garden Business

Word Count: 2222 **Page Count:** 15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Punya Nandakumar Kedambadi

Date: 29/01/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Punya Nandakumar Kedambadi
X23134658

1 Introduction

The presented configuration manual provides detailed instructions for configuring and deploying the research with title “Tailoring Customer Engagement: Advanced Segmentation for Growth in the Garden Business”. The system utilises advanced segmentation of Machine Learning and Data Analytics to provide valuable insights for companies and the agricultural enthusiasts.

2 System Configuration

Necessary hardware and software resources must be understood to guarantee efficient model processing and ensuring there is less time duration for the model execution.

2.1 Hardware Requirements:

The implementation is performed on a MacBook Air. The configuration of the device is as follows:

- **Processor:** Apple M2 Chip with incredible CPU, GPU, and machine learning performance
- **CPU:** 8-core CPU
- **GPU:** 8-core GPU
- **Memory:** Unified memory, 8 GB
- **Storage:** 512 GB SSD
- **Operating System:** macOS Ventura Version 13.0

2.2 Software Requirements:

Machine Learning and Data Mining algorithms are implemented using the below software as in Table 1 and Figure 1.

Table 1: Software Requirements

Software Configuration	Version
R Language	4.3.1
RStudio	2024.09.0+375

dplyr	1.1.3
ggplot2	3.4.0
caret	6.0-94
factoextra	1.0.7
cluster	2.1.4
lubridate	1.9.3

Figure 1: R Console and R studio.

3 Project Implementation

This section focuses on the subsections like pre-processing, visualization, RFM analysis which is done before the model implementation and evaluation.

3.1 Data Pre-processing

Using file.choose(), we can choose the path where our dataset required to do our segmentation and as per our research we are naming the selected CSV file as "mdg" as shown in Figure 2. Also as the date in our dataset was not in the exact format as required for R, as.Date() function is used to obtain an explicit format.

```
# 2) Data Understanding
## 2.1) Data Acquisition
mdg <- read.csv(file.choose())

## 2.2) Data Exploration
sapply(mdg, class)  # Check column names and data types
str(mdg)  # Structure of the dataset (e.g., 6569 obs. of 12 variables)

# 3) Data Pre-Processing
#3.1)Convert 'Date' to Date format using explicit format
mdg$Date <- as.Date(mdg$Date, format = "%d/%m/%y")

# Check if the conversion was successful
print(head(mdg$Date))
```

Figure 2: Data acquisition and pre-processing.

3.2 Data Visualization

Once the data is analysed with basic explorations, visualisation is done as it assists in getting the valuable insights through graphs where we can obtain extensive information which was missed earlier. Nine visualisations were done to get the trend analysis, demographic insights as shown in Figure 3 and Figure 4. Under the Figure 4, boxplot is used to showcase the outliers.

```r
# 3.2.1) Aggregate sales by Date
sales_trend <- mdg %>%
  group_by(Date) %>%
  summarize(Total_Value = sum(Total.value.of.sale), .groups = 'drop')

sales_trend$Date <- as.Date(sales_trend$Date, format = "%Y-%m-%d")

# Plot Sales Trend Over Time
ggplot(sales_trend, aes(x = Date, y = Total_Value)) +
  geom_line(color = "blue") +  # Create a line plot
  labs(title = "Sales Trend Over Time", x = "Date", y = "Total Sales Value") +
  theme_minimal()

# 3.2.2) Aggregate sales by month
sales_trend$Month <- floor_date(sales_trend$Date, "month")
monthly_sales <- sales_trend %>%
  group_by(Month) %>%
  summarize(Total_Value = sum(Total_Value), .groups = 'drop')

# Plot Monthly Sales Trend
ggplot(monthly_sales, aes(x = Month, y = Total_Value)) +
  geom_line(color = "blue") +
  labs(title = "Monthly Sales Trend", x = "Month", y = "Total Sales Value") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

#3.2.3) Aggregate sales by year
sales_trend$Year <- year(sales_trend$Date)
yearly_sales <- sales_trend %>%
  group_by(Year) %>%
  summarize(Total_Value = sum(Total_Value), .groups = 'drop')

# Plot Yearly Sales Trend
ggplot(yearly_sales, aes(x = Year, y = Total_Value)) +
  geom_line(color = "blue") +
  labs(title = "Yearly Sales Trend", x = "Year", y = "Total Sales Value") +
  theme_minimal()
```

```r
#3.2.4) Sales by Customer Age Group
mdg$Age_Group <- cut(mdg$Age, breaks = seq(0, 100, by = 10), right = FALSE,
                     labels = paste(seq(0, 90, by = 10), seq(9, 99, by = 10), sep = "-"))

age_sales <- mdg %>%
  group_by(Age_Group) %>%
  summarize(Total_Value = sum(Total.value.of.sale), .groups = 'drop')
ggplot(age_sales, aes(x = Age_Group, y = Total_Value, fill = Age_Group)) +
  geom_bar(stat = "identity") +
  labs(title = "Total Sales by Customer Age Group", x = "Age Group", y = "Total Sales Value") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

#3.2.5) Sales Distribution by Marital Status
marital_sales <- mdg %>%
  group_by(Marital.status) %>%
  summarize(Total_Value = sum(Total.value.of.sale), .groups = 'drop')
ggplot(marital_sales, aes(x = Marital.status, y = Total_Value, fill = Marital.status)) +
  geom_bar(stat = "identity") +
  labs(title = "Total Sales Distribution by Marital Status", x = "Marital Status", y = "Total Sales Value") +
  theme_minimal()

#3.2.6) Top 10 Products by Total Quantity Sold
quantity_sold <- mdg %>%
  group_by(Product) %>%
  summarize(Total_Quantity = sum(Quantity.Sold), .groups = 'drop') %>%
  top_n(10, Total_Quantity) %>%
  arrange(desc(Total_Quantity))

ggplot(quantity_sold, aes(x = reorder(Product, Total_Quantity), y = Total_Quantity, fill = Product)) +
  geom_bar(stat = "identity", width = 0.7) +
  geom_text(aes(label = Total_Quantity),
            hjust = -0.2,
            color = "black",
            size = 4,
            fontface = "bold") +
  labs(title = "Top 10 Products by Total Quantity Sold",
       x = NULL,
       y = "Total Quantity Sold") +
  theme_minimal(base_size = 12) +
  theme(axis.text.x = element_blank(), # Remove labels from x-axis
        axis.ticks.x = element_blank(), # Remove ticks from x-axis
        axis.title.x = element_text(size = 14, face = "bold"),
        axis.title.y = element_blank(), # No title on y-axis
        axis.text.y = element_text(size = 10, face = "bold"), # Adjust left-side product names
        legend.position = "none", # Remove legend
        plot.title = element_text(hjust = 0.5, face = "bold", size = 16)) +
  scale_fill_brewer(palette = "Paired") +
```

Figure 3: Trend and demographic analysis 1.

```r
#3.3.7) Total Sales by Location - Top 10 Locations
location_sales <- mdg %>%
  group_by(Location) %>%
  summarize(Total_Value = sum(Total.value.of.sale), .groups = 'drop') %>%
  slice_max(order_by = Total_Value, n = 10)

ggplot(location_sales, aes(x = reorder(Location, -Total_Value), y = Total_Value, fill = Location)) +
  geom_bar(stat = "identity") +
  labs(title = "Top 10 Total Sales by Location", x = "Location", y = "Total Sales Value") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

#3.3.8) Sales Value by Customer Type
customer_type_sales <- mdg %>%
  group_by(Type.of.Customer) %>%
  summarize(Total_Value = sum(Total.value.of.sale), .groups = 'drop')

ggplot(customer_type_sales, aes(x = reorder(Type.of.Customer, -Total_Value), y = Total_Value, fill = Type.of.Customer
  geom_bar(stat = "identity") +
  labs(title = "Total Sales Value by Type of Customer", x = "Type of Customer", y = "Total Sales Value") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

#3.3.9) Boxplot of Sales Value by Age Group
ggplot(mdg, aes(x = Age_Group, y = Total.value.of.sale, fill = Age_Group)) +
  geom_boxplot() +
  labs(title = "Sales Value Distribution by Age Group", x = "Age Group", y = "Total Sales Value") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Figure 4: Trend and demographic analysis 2.

3.3 Variables conversion and duplicates removal

Most of the algorithms does not work well on the categorical variables, hence they are converted into factorials or numerical for better implementation and maintaining consistency as shown in Figure 5.

```r
## 3.2) Convert categorical variables to factors
New_mdg <- mdg %>%
  mutate(
    Marital.status = factor(Marital.status),
    Location = factor(Location),
    Type.of.Customer = factor(Type.of.Customer),
    Product = factor(Product),
    Preferred.Category = factor(Preferred.Category)
  )

# Check the structure of the dataset to confirm the changes
str(New_mdg)
```

Figure 5: Categorical variables conversion.

In the complete dataset, we try to identify the missing values and luckily there was no missing values found. But eight duplicate rows were removed from the dataset as shown in Figure 6.

```r
## 3.3) Check for missing values
colSums(is.na(New_mdg))

## 3.4) Check for Duplicates
# Check for duplicate rows in the entire dataset
duplicates <- New_mdg[duplicated(New_mdg), ]
print(duplicates)

# Count the total number of duplicate rows
num_duplicates <- sum(duplicated(New_mdg))
print(num_duplicates) #8 rows removed

# Remove duplicate rows
New_mdg_unique <- New_mdg %>%
  distinct() #6569 observations to 6561 obervations
```

Figure 6: Duplicates removal.

3.4 Outlier detection and removal

Boxplot graphs helps to identify the outliers present in the dataset during the visualisation step. We have also utilised techniques like Z-score and IQR-based thresholds to identify and eliminate outliers as shown in Figure 7.

```r
# 4) Outlier Detection and Removal
## Step 1: Calculate Z-scores
New_mdg_unique$Z_Score <- scale(New_mdg_unique$Total.value.of.sale)

## Identify outliers using Z-score (threshold > 3 or < -3)
mdg_no_z_outliers <- New_mdg_unique %>%
  dplyr::filter(abs(Z_Score) <= 3)

## Step 2: Calculate IQR
Q1 <- quantile(mdg_no_z_outliers$Total.value.of.sale, 0.25)
Q3 <- quantile(mdg_no_z_outliers$Total.value.of.sale, 0.75)
IQR_value <- IQR(mdg_no_z_outliers$Total.value.of.sale)

## Define outlier thresholds using IQR
lower_threshold <- Q1 - 1.5 * IQR_value
upper_threshold <- Q3 + 1.5 * IQR_value

## Step 3: Remove outliers using IQR
mdg_final <- mdg_no_z_outliers %>%
  dplyr::filter(Total.value.of.sale >= lower_threshold & Total.value.of.sale <= upper_threshold)

## Remove the Z_Score column as it's no longer needed
mdg_final <- mdg_final %>%
  select(-Z_Score)

# Print original and final row counts
original_row_count <- nrow(New_mdg_unique)
cat("Original row count before outlier detection: ", original_row_count, "\n")

row_count_after_z_score <- nrow(mdg_no_z_outliers)
cat("Row count after Z-score outlier detection: ", row_count_after_z_score, "\n")

final_row_count <- nrow(mdg_final)
cat("Row count after IQR-based outlier detection: ", final_row_count, "\n")

# Step 4: Save the cleaned dataset to a CSV file
output_file <- "cleaned_mdg_data.csv"
write.csv(mdg_final, file = output_file, row.names = FALSE)
```

Figure 7: Outlier detection and removal.

3.5 Feature Normalisation

Min-Max scaling is used in our research to rescale the features of our dataset to range between [0,1] or [-1,1], making sure that all the features will contribute equally to the model implementation as in Figure 8.

```
# 5) Feature Normalization (Min-Max Scaling)
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Apply normalization to relevant numeric columns
mdg_final <- mdg_final %>%
  mutate(
    Total.value.of.sale = normalize(Total.value.of.sale),
    Price = normalize(Price),
    Quantity.Sold = normalize(Quantity.Sold)
  )
```

Figure 8: Feature normalisation.

3.6 RFM analysis

In this RFM analysis phase, customers are evaluated based on three metrics like Recency, Frequency, and Monetary. Each customer is assigned scores for the three metrics and then overall sum is computed as RFM scores. Based on their RFM scores, customers are segmented into categories such as Best Customers, Loyal Customers, Potential Customers, and At-Risk Customers to understand their value and to target them strategically as mentioned in Figure 9.

```
# 6) RFM Analysis
#6.1) RFM calculation
rfm_data <- mdg_final %>%
  group_by(Customer.ID) %>%
  summarise(
    Recency = as.numeric(difftime(Sys.Date(), max(Date), units = "days")),
    Frequency = n(),
    Monetary = sum(Total.value.of.sale)
  )

# Define the output file name for RFM data
output_file_rfm <- "rfm_data.csv"
write.csv(rfm_data, file = output_file_rfm, row.names = FALSE)

#6.2)Score Customers based on RFM metrics
rfm_data_score <- rfm_data %>%
  mutate(
    R_Score = ntile(-Recency, 5),  # Lower recency is better (hence negative)
    F_Score = ntile(Frequency, 5),
    M_Score = ntile(Monetary, 5),
    RFM_Score = R_Score + F_Score + M_Score  # Total RFM score
  )

# Save the RFM data with scores
output_file_rfm_score <- "rfm_data_score.csv"
write.csv(rfm_data_score, file = output_file_rfm_score, row.names = FALSE)

#6.3) Segment Customers
rfm_data_score_segments <- rfm_data_score %>%
  mutate(
    Segment = case_when(
      RFM_Score >= 9 ~ "Best Customers",
      RFM_Score >= 6 ~ "Loyal Customers",
      RFM_Score >= 4 ~ "Potential Customers",
      TRUE ~ "At Risk Customers"
    )
  )
```

Figure 9: RFM score on the three metrics.

Also, visualisations are done to view the RFM metrics and their scores to understand the distribution as shown in Figure 10.

```r
#7.4) RFM visualisations
#7.4.1) Visualize RFM Segments
rfm_segment_counts <- rfm_data_score_segments %>%
  group_by(Segment) %>%
  summarise(Total_Customers = n(), .groups = 'drop')

# Bar plot for number of customers in each segment
ggplot(rfm_segment_counts, aes(x = Segment, y = Total_Customers, fill = Segment)) +
  geom_bar(stat = "identity") +
  labs(title = "Number of Customers in Each RFM Segment", x = "Customer Segment", y = "Total Customers") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

#7.4.2) Visualize Recency, Frequency, and Monetary by Segment
# Load the tidyr package
library(tidyr)

rfm_segment_metrics <- rfm_data_score_segments %>%
  group_by(Segment) %>%
  summarise(
    Avg_Recency = mean(Recency),
    Avg_Frequency = mean(Frequency),
    Avg_Monetary = mean(Monetary),
    .groups = 'drop'
  ) %>%
  pivot_longer(cols = starts_with("Avg"), names_to = "Metric", values_to = "Value")

# Bar plot for average Recency, Frequency, and Monetary by Segment
ggplot(rfm_segment_metrics, aes(x = Segment, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Average RFM Metrics by Customer Segment", x = "Customer Segment", y = "Average Value") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```r
#7.4.3) Visualize RFM Score Distribution
ggplot(rfm_data_score, aes(x = RFM_Score)) +
  geom_histogram(binwidth = 1, fill = "lightblue", color = "black") +
  labs(title = "Distribution of RFM Scores", x = "RFM Score", y = "Number of Customers") +
  theme_minimal()

#7.4.4) Visualize Average Monetary Value by Segment
ggplot(rfm_segment_metrics %>% filter(Metric == "Avg_Monetary"), aes(x = Segment, y = Value, fill = Segment)) +
  geom_bar(stat = "identity") +
  labs(title = "Average Monetary Value by Customer Segment", x = "Customer Segment", y = "Average Monetary Value") +
  theme_minimal()
```

Figure 10: RFM metrics visualisation.

3.7 Dimensionality Reduction

Principle Component Analysis is done to reduce the dimensionality and to retain the principle components which contribute maximum of the variance in this dataset as shown in Figure 11.

```r
#8) Feature selection for clustering process
# Perform PCA on the numeric features

numeric_features <- mdg_final %>% select(where(is.numeric))

pca_result <- prcomp(numeric_features, scale. = TRUE)

# Plot variance explained by each principal component
fviz_eig(pca_result)

# Choose components that capture at least 90% of the variance
pca_data <- as.data.frame(pca_result$x[, 1:which(cumsum(pca_result$sdev^2 / sum(pca_result$sdev^2)) >= 0.9)[1]])
```

Figure 11: PCA.

4 Model Application

After all the initial pre-processing and RFM analysis which is required to implement the algorithms to segments the customers are done through all the above steps. In this section we proceed to implement the actual algorithms necessary for research. Silhouette score method is a measure of the quality of a cluster used to find the mean silhouette co-efficient for different

number of clusters. The highest silhouette score indicates the optimal number of clusters as highlighted by (Shahapure et al.; 2020).

4.1 Determining optimal number of clusters before algorithm implementation

Before implementing the clustering algorithms, we need to identify the ideal number of clusters. In this research we have used Elbow method and Silhouette analysis. Once the value of k is identified, the same value will be utilised across all the algorithms to maintain uniformity and not to bias on the results as shown in Figure 12.

```
#9.1) Determine Optimal Number of Clusters

#9.1.a)Elbow method(The Elbow method evaluates the total within-cluster
#sum of squares (WSS) for different values of k)

# Calculate WSS for a range of k values
wss <- sapply(1:10, function(k) {
  kmeans(pca_data, centers = k, nstart = 25)$tot.withinss
})

# Plot the elbow curve
plot(1:10, wss, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of Clusters K", ylab = "Total Within-Cluster Sum of Squares",
     main = "Elbow Method for Optimal K")


#9.1.b)Silhouette Analysis (The silhouette width measures how similar each point is to
#its own cluster compared to other clusters.)

# Calculate average silhouette width for different values of k
library(cluster)
avg_sil_width <- sapply(2:10, function(k) {
  km_res <- kmeans(pca_data, centers = k, nstart = 25)
  silhouette_score <- silhouette(km_res$cluster, dist(pca_data))
  mean(silhouette_score[, 3])
})

# Plot silhouette widths
plot(2:10, avg_sil_width, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of Clusters K", ylab = "Average Silhouette Width",
     main = "Silhouette Method for Optimal K")
```

Figure 12: Determine optimal number of clusters.

4.2 Clustering algorithms

We have made use of algorithms like K-means, DBSCAN, Hierarchical Clustering, and GMM integrated with RFM analysis to segment the customers related to the garden business and are described in below sections. All the libraries required to implement these algorithms are loaded.

4.2.1 K-means clustering

This code performs K-means clustering on the data using a range of k values (2,3,4,6,7) to identify the most suitable k based on the highest average Silhouette score, which evaluates the quality of clustering. For each of the k values, clustering results and its scores are

calculated and stored in a data frame. Visualization is also done to check how the clusters are separated with the different values of k as shown in Figure 13.

```
#9.2) Run K-Means Clustering
# List to store clustering results and silhouette scores
cluster_results <- list()
silhouette_scores <- c()

# Range of K values to try based on Elbow and Silhouette analysis
k_values <- c(2, 3, 4, 6, 7)

# Loop over each K value and run K-means
for (k in k_values) {
  set.seed(123)  # for reproducibility
  kmeans_result <- kmeans(pca_data, centers = k, nstart = 25)

  # Store clustering result
  cluster_results[[as.character(k)]] <- kmeans_result

  # Calculate silhouette score
  library(cluster)
  silhouette_score <- silhouette(kmeans_result$cluster, dist(pca_data))
  avg_silhouette <- mean(silhouette_score[, 3])
  silhouette_scores <- c(silhouette_scores, avg_silhouette)
}

# Combine K values and silhouette scores into a data frame for easy viewing
results_df <- data.frame(
  K = k_values,
  Silhouette_Score = silhouette_scores
)
# Print results
print(results_df)

# Find the optimal K based on the highest silhouette score
optimal_k <- k_values[which.max(silhouette_scores)]
cat("Optimal K based on silhouette scores:", optimal_k, "\n")
```

```
# Run final K-means clustering with the optimal K
set.seed(123)
final_kmeans_result <- kmeans(pca_data, centers = optimal_k, nstart = 25)

# clusters if you want to visualize
library(factoextra)
fviz_cluster(final_kmeans_result, data = pca_data,
             geom = "point", ellipse.type = "convex",
             main = paste("K-means Clustering with K =", optimal_k))
```

Figure 13: K-means clustering.

4.2.1.1 Using Davies-Bouldin function

Davies-Bouldin Index (DBI) is used to evaluate the cluster quality with different values of k. This function helps to measure the compactness and separation of clusters, where a lower DBI indicates better-defined clusters. For each value of k, K-means clustering is done and the DBI values are calculated as shown in Figure 14. Also, WSS is recorded to check the quality of the cluster in further. The summary of all the results including WSS, k values and DBI values is done. The ideal number of clusters is selected as the k=6 which has the lowest DBI

value to run the K-means clustering. Finally, the results are visualised to determine the points.

```r
#9.3) Davies-Bouldin
# Load necessary libraries
library(cluster)   # For silhouette calculation if needed
library(factoextra) # For WSS calculation

# Define custom Davies-Bouldin function
davies.bouldin <- function(data, cluster_labels) {
  clusters <- unique(cluster_labels)
  n_clusters <- length(clusters)

  # Calculate centroids for each cluster
  centroids <- lapply(clusters, function(c) colMeans(data[cluster_labels == c, , drop=FALSE]))
  # Calculate average distance within each cluster (cluster compactness)
  s <- sapply(clusters, function(c) {
    cluster_points <- data[cluster_labels == c, , drop=FALSE]
    mean(dist(cluster_points, method = "euclidean"))
  })

  db_index <- 0
  for (i in 1:n_clusters) {
    max_r <- 0
    for (j in 1:n_clusters) {
      if (i != j) {
        # Distance between centroids of clusters i and j
        d_ij <- dist(rbind(centroids[[i]], centroids[[j]]), method = "euclidean")[1]
        # Ratio of average distances within clusters to distance between clusters
        r_ij <- (s[i] + s[j]) / d_ij
        if (r_ij > max_r) max_r <- r_ij
      }
    }
    db_index <- db_index + max_r
  }
  db_index / n_clusters
}

# Initialize lists to store results
davies_bouldin_scores <- c()
wss_scores <- c()

# Range of K values to try based on analysis (Elbow and Silhouette)
k_values <- c(2, 3, 4, 6, 7)

# Loop over each K value and run K-means
for (k in k_values) {
  set.seed(123)  # for reproducibility
  kmeans_result <- kmeans(pca_data, centers = k, nstart = 25)

  # Calculate WSS for current K (within-cluster sum of squares)
  wss_scores <- c(wss_scores, kmeans_result$tot.withinss)

  # Calculate Davies-Bouldin index for current K
  db_index <- davies.bouldin(pca_data, kmeans_result$cluster)
  davies_bouldin_scores <- c(davies_bouldin_scores, db_index)
}

# Create the evaluation data frame with K, WSS, and DBI
evaluation_df <- data.frame(
  K = k_values,
  WSS = wss_scores,
  Davies_Bouldin_Index = davies_bouldin_scores
)

# Print the evaluation data frame
print(evaluation_df)

# Identify the optimal K based on the lowest Davies-Bouldin index
optimal_k_db <- k_values[which.min(davies_bouldin_scores)]
cat("Optimal K based on Davies-Bouldin index:", optimal_k_db, "\n")

#Optimal K based on Davies-Bouldin index: 6

# Set the optimal K based on Davies-Bouldin index
optimal_k <- 6

# Run K-means clustering with the optimal K
set.seed(123)  # for reproducibility
final_kmeans_result <- kmeans(pca_data, centers = optimal_k, nstart = 25)

# Visualize the clustering result
library(factoextra)
fviz_cluster(final_kmeans_result, data = pca_data,
             geom = "point", ellipse.type = "convex",
             main = paste("K-means Clustering with K =", optimal_k))
```

Figure 14: DBI calculation.

4.2.1.2 Evaluating the cluster quality

As highlighted in Figure 15, this piece of code calculates the quality of the clusters with k=6 which is obtained from final K-means clustering. Firstly, Silhouette score is calculated for each of the clusters, which explains how the data points are fitting in comparison with each other. The average Silhouette score is calculated, with higher values indicating better cluster cohesion and separation. Next the plot is used to visualise the quality of the clustering. In addition, WSS is calculated to measure the compactness. These metrics overall give a detailed explanation on the quality of the clusters validating the value of k chosen as 6.

```r
#9.4) Evaluate Cluster Quality
# Assuming `final_kmeans_result` is the final clustering result with K = 6

# 1. Calculate Silhouette Scores for Each Cluster
library(cluster)

# Compute silhouette scores
silhouette_values <- silhouette(final_kmeans_result$cluster, dist(pca_data))
avg_silhouette_score <- mean(silhouette_values[, 3])  # Average silhouette score for all clusters

# Print the average silhouette score for overall evaluation
cat("Average Silhouette Score for K =", optimal_k, ":", avg_silhouette_score, "\n")

# 2. Plot Silhouette Width for Each Cluster
fviz_silhouette(silhouette_values) +
  labs(title = paste("Silhouette Plot for K =", optimal_k))

# This plot helps in visually assessing the cohesion and separation of each cluster. A higher silhouett

# 3. Per-Cluster WSS (Within-Cluster Sum of Squares)
# Calculate WSS for each cluster
wss_per_cluster <- sapply(1:optimal_k, function(i) {
  sum(dist(pca_data[final_kmeans_result$cluster == i, ])^2) /
    nrow(pca_data[final_kmeans_result$cluster == i, ])
})

# Print WSS for each cluster
cat("WSS for each cluster with K =", optimal_k, ":\n")
print(wss_per_cluster)

# 4. Recalculate Davies-Bouldin Index for Final Clustering (K = 6)
db_index_final <- davies.bouldin(pca_data, final_kmeans_result$cluster)
cat("Davies-Bouldin Index for final clustering (K =", optimal_k, "):", db_index_final, "\n")

# Summary of cluster quality metrics
quality_summary <- data.frame(
  Cluster = 1:optimal_k,
  WSS_Per_Cluster = wss_per_cluster,
  Silhouette_Width = silhouette_values[, 3][match(1:optimal_k, final_kmeans_result$cluster)]
)
```

Figure 15: Evaluating cluster quality.

4.2.1.3 Rerun K-means with k=4

Although k value is identified as 6, and cluster quality also is checked, K-means clustering is rerun by taking values as 4 and 5. It proceeds to recalculate the evaluation metrics with k=4 like Silhouette score which helps to measure the cohesion of clusters as shown in Figure 16 and, how they are separated and their DBI values to identify the compactness by calculating the WSS. Finally, the score and the graph are plotted with value of k=4 to further analyse the quality and providing insights for companies.

```r
##1)Re-run K-means clustering for a different K (e.g., K = 4 or K = 5)
alternative_k <- 4

# Run K-means clustering
set.seed(123)
alternative_kmeans_result <- kmeans(pca_data, centers = alternative_k, nstart = 25)

# Calculate Silhouette Score for new K
silhouette_values_alt <- silhouette(alternative_kmeans_result$cluster, dist(pca_data))
avg_silhouette_score_alt <- mean(silhouette_values_alt[, 3])
cat("Average Silhouette Score for K =", alternative_k, ":", avg_silhouette_score_alt, "\n")

# Calculate Davies-Bouldin Index for new K
db_index_alt <- davies.bouldin(pca_data, alternative_kmeans_result$cluster)
cat("Davies-Bouldin Index for final clustering (K =", alternative_k, "):", db_index_alt, "\n")

# Plot silhouette for new K
fviz_silhouette(silhouette_values_alt) +
  labs(title = paste("Silhouette Plot for K =", alternative_k))

# Calculate WSS for each cluster
wss_per_cluster_alt <- sapply(1:alternative_k, function(i) {
  sum(dist(pca_data[alternative_kmeans_result$cluster == i, ])^2) /
    nrow(pca_data[alternative_kmeans_result$cluster == i, ])
})

# Print WSS per cluster
cat("WSS for each cluster with K =", alternative_k, ":\n")
print(wss_per_cluster_alt)
```

Figure 16: Rerun K-means with k=4.

4.2.1.4 Rerun K-means with k=5

Again, k value is taken as 5 and the process is repeated to calculate the Silhouette score, print the WSS and the plot the scores within the clusters as showcased in Figure 17.

```r
##2)Re-run K-means clustering for a different K (e.g., K = 4 or K = 5)
alternative_k <- 5

# Run K-means clustering
set.seed(123)
alternative_kmeans_result <- kmeans(pca_data, centers = alternative_k, nstart = 25)

# Calculate Silhouette Score for new K
silhouette_values_alt <- silhouette(alternative_kmeans_result$cluster, dist(pca_data))
avg_silhouette_score_alt <- mean(silhouette_values_alt[, 3])
cat("Average Silhouette Score for K =", alternative_k, ":", avg_silhouette_score_alt, "\n")

# Calculate Davies-Bouldin Index for new K
db_index_alt <- davies.bouldin(pca_data, alternative_kmeans_result$cluster)
cat("Davies-Bouldin Index for final clustering (K =", alternative_k, "):", db_index_alt, "\n")

# Plot silhouette for new K
fviz_silhouette(silhouette_values_alt) +
  labs(title = paste("Silhouette Plot for K =", alternative_k))

# Calculate WSS for each cluster
wss_per_cluster_alt <- sapply(1:alternative_k, function(i) {
  sum(dist(pca_data[alternative_kmeans_result$cluster == i, ])^2) /
    nrow(pca_data[alternative_kmeans_result$cluster == i, ])
})

# Print WSS per cluster
cat("WSS for each cluster with K =", alternative_k, ":\n")
print(wss_per_cluster_alt)

#Code to Save and Use Final Clustering for k=6
#Add final cluster labels to the preprocessed data
final_data_with_clusters <- cbind(mdg_final, Cluster = final_kmeans_result$cluster)

# Save as a CSV for further analysis
write.csv(final_data_with_clusters, "final_clustering_results_K6.csv", row.names = FALSE)
```

Figure 17: Rerun K-means with k=5.

4.2.1.5 Profiling the clusters

Figure 18 illustrates the code to profile the clusters once the clustering process is completed. Calculation of RFM metrics is done firstly for each of the customers if and only if they are not done in the initial stages. These metrics will be joined with the clustered dataset and mean, median values are calculated with key attributes like average age, sales value, quantity sold, price, and RFM metrics, as well as the total number of customers in each cluster. Lastly the summary is calculated inside the cluster profile which aims to highlight the characteristics of each cluster. This step can help to tailor the marketing strategies.

```r
#9.5.2)Profiling Each Cluster
# Calculate RFM metrics if not already present
rfm_data <- mdg_final %>%
  group_by(Customer.ID) %>%
  summarise(
    Recency = as.numeric(difftime(Sys.Date(), max(Date), units = "days")),
    Frequency = n(),
    Monetary = sum(Total.value.of.sale)
  )

# Merge RFM metrics with final_data_with_clusters
final_data_with_clusters <- final_data_with_clusters %>%
  left_join(rfm_data, by = "Customer.ID")

# Now, calculate the mean or median for key features within each cluster
cluster_profile <- final_data_with_clusters %>%
  group_by(Cluster) %>%
  summarise(
    Avg_Age = mean(Age, na.rm = TRUE),
    Avg_Total_Sales_Value = mean(Total.value.of.sale, na.rm = TRUE),
    Avg_Quantity_Sold = mean(Quantity.Sold, na.rm = TRUE),
    Avg_Price = mean(Price, na.rm = TRUE),
    Avg_Recency = mean(Recency, na.rm = TRUE),
    Avg_Frequency = mean(Frequency, na.rm = TRUE),
    Avg_Monetary = mean(Monetary, na.rm = TRUE),
    Total_Customers = n()
  )

# Display the cluster profile
print(cluster_profile)
```

Figure 18: Profiling the clusters.

4.2.1.6 Visualisation of clusters

This code visualises the cluster profile for better interpretation by creating bar charts to identify the key metrics across the clusters. Four bar chars are used to visualise the average total sales by cluster, average age by cluster, average frequency by cluster and average monetary value by cluster as shown in Figure 19.

```
# Visualize cluster profiles for better interpretation
# Average Total Sales Value by Cluster
ggplot(cluster_profile, aes(x = as.factor(Cluster), y = Avg_Total_Sales_Value, fill = as.factor(Cluster))) +
  geom_bar(stat = "identity") +
  labs(title = "Average Total Sales Value by Cluster", x = "Cluster", y = "Average Sales Value") +
  theme_minimal()

# Average Age by Cluster
ggplot(cluster_profile, aes(x = as.factor(Cluster), y = Avg_Age, fill = as.factor(Cluster))) +
  geom_bar(stat = "identity") +
  labs(title = "Average Age by Cluster", x = "Cluster", y = "Average Age") +
  theme_minimal()

# Average Frequency by Cluster
ggplot(cluster_profile, aes(x = as.factor(Cluster), y = Avg_Frequency, fill = as.factor(Cluster))) +
  geom_bar(stat = "identity") +
  labs(title = "Average Frequency by Cluster", x = "Cluster", y = "Average Frequency") +
  theme_minimal()

# Average Monetary Value by Cluster
ggplot(cluster_profile, aes(x = as.factor(Cluster), y = Avg_Monetary, fill = as.factor(Cluster))) +
  geom_bar(stat = "identity") +
  labs(title = "Average Monetary Value by Cluster", x = "Cluster", y = "Average Monetary Value") +
  theme_minimal()
```

Figure 19: Visualising the clusters.

4.2.1 Hierarchical clustering

Hierarchical clustering method is chosen as one among the advanced clustering method to compliment K-means and provide deeper insights. A dissimilarity matrix is evaluated using the Euclidean distance, which forms the foundation for this clustering. Then Wards method is used to perform hierarchical clustering and k value is taken as 6 to maintain uniformity. The complete clustering is visualized using dendrogram with red rectangle separating the 6 clusters as shown in Figure 20. This whole approach is done on the PCA transformed data and this increases the robustness of the analysis.

```
#10) Enhanced Clustering Analysis
#10.1) Experiment with Different Algorithms
# Load required libraries
library(cluster)      # For hierarchical clustering and silhouette calculation
library(dbscan)       # For DBSCAN
library(mclust)       # For Gaussian Mixture Models (GMM)
library(factoextra)   # For visualization
library(tidyverse)    # For data manipulation and visualization

#10.1.1) Hierarchical Clustering
# Compute dissimilarity matrix
dist_matrix <- dist(pca_data)

# Perform hierarchical clustering with Ward's method
hc_result <- hclust(dist_matrix, method = "ward.D2")

# Cut tree into 6 clusters (for consistency with K-means)
hc_clusters <- cutree(hc_result, k = 6)

# Visualize dendrogram
plot(hc_result, main = "Dendrogram of Hierarchical Clustering", xlab = "", sub = "")
rect.hclust(hc_result, k = 6, border = "red")

# Add cluster labels to pca_data for analysis
pca_data$hc_cluster <- hc_clusters
```

Figure 20: Hierarchical clustering.

4.2.1 DBSCAN clustering

DBSCAN is a density-based clustering algorithm which identifies the clusters based on the density of data points. The parameters eps and minPts are chosen based on the dataset distribution. They identify the core points, border points and their noises making highly effective for the datasets who have outliers and variety of cluster shapes. The clustering result is visualized using a scatter plot, with points grouped by clusters, and noise points left isolated. Finally, the cluster labels assigned by DBSCAN are added to the PCA transformed data for further analysis as shown in Figure 21.

```
#10.1.2) DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
# Choose epsilon and minPts based on your data distribution (you may need to experiment with values)
dbscan_result <- dbscan(pca_data, eps = 0.5, minPts = 5)

# Plot DBSCAN clustering result
fviz_cluster(dbscan_result, data = pca_data, geom = "point", ellipse = FALSE, main = "DBSCAN Clustering")

# Add DBSCAN cluster labels to pca_data for analysis
pca_data$dbscan_cluster <- dbscan_result$cluster
```

Figure 21: DBSCAN clustering.

4.2.1 GMM clustering

As shown in Figure 22, GMM clustering is implemented using the Mclust package to fit the 6 components and this method is a probabilistic soft clustering approach. The results are visualized with fviz_mclust to display the classification, and the generated cluster labels are added to the dataset for further analysis.

```
#10.1.3) Gaussian Mixture Models (GMM)
# Fit GMM using Mclust package
gmm_result <- Mclust(pca_data, G = 6)  # G = 6 means we are trying to fit 6 components

# Plot GMM clustering result
fviz_mclust(gmm_result, "classification", geom = "point", main = "GMM Clustering")

# Add GMM cluster labels to pca_data for analysis
pca_data$gmm_cluster <- gmm_result$classification
```

Figure 22: GMM clustering.

5 Model Evaluation

Taking the base as Silhouette score, the quality all the four clustering algorithm is calculated. Each score provides insight into the effectiveness of the respective algorithm in segmenting the data as shown in Figure 23.

```r
#11) Evalation
#11.1) Calculation
# K-means Silhouette Score
kmeans_silhouette <- silhouette(kmeans_result$cluster, dist(pca_data))
avg_kmeans_sil <- mean(kmeans_silhouette[, 3])
cat("Average Silhouette Score for K-means:", avg_kmeans_sil, "\n")

# Hierarchical Clustering Silhouette Score
hc_silhouette <- silhouette(hc_clusters, dist(pca_data))
avg_hc_sil <- mean(hc_silhouette[, 3])
cat("Average Silhouette Score for Hierarchical Clustering:", avg_hc_sil, "\n")

# DBSCAN does not provide a silhouette score directly due to noise points; however, you can calculate it on clusters only
dbscan_silhouette <- silhouette(dbscan_result$cluster[dbscan_result$cluster > 0], dist(pca_data[dbscan_result$cluster > 0, ]))
avg_dbscan_sil <- mean(dbscan_silhouette[, 3], na.rm = TRUE)
cat("Average Silhouette Score for DBSCAN (excluding noise):", avg_dbscan_sil, "\n")

# Gaussian Mixture Models (GMM) Silhouette Score
gmm_silhouette <- silhouette(gmm_result$classification, dist(pca_data))
avg_gmm_sil <- mean(gmm_silhouette[, 3])
cat("Average Silhouette Score for GMM:", avg_gmm_sil, "\n")
```

Figure 23: Model evaluation.

5.1 Visualisation of the SIL scores with red dashed line

Once the Silhouette score is calculated for all the four algorithms, they are visualised so that a comparison is made as shown in Figure 24. Each plot will show how well the clusters are separated for the respective method and thus helping which algorithm performs best in defining distinct and meaningful clusters.

```r
#11.2)Visualise the SIl score
# Required library
library(factoextra)

# 1. K-means Silhouette Plot
fviz_silhouette(kmeans_silhouette) +
  labs(title = "Silhouette Plot for K-means Clustering",
       x = "Clusters", y = "Silhouette Width")

# 2. Hierarchical Clustering Silhouette Plot
fviz_silhouette(hc_silhouette) +
  labs(title = "Silhouette Plot for Hierarchical Clustering",
       x = "Clusters", y = "Silhouette Width")

# 3. DBSCAN Silhouette Plot (Excluding Noise Points)
fviz_silhouette(dbscan_silhouette) +
  labs(title = "Silhouette Plot for DBSCAN Clustering (Excluding Noise)",
       x = "Clusters", y = "Silhouette Width")

# 4. Gaussian Mixture Models (GMM) Silhouette Plot
fviz_silhouette(gmm_silhouette) +
  labs(title = "Silhouette Plot for Gaussian Mixture Models",
       x = "Clusters", y = "Silhouette Width")
```

Figure 24: Visualization of the SIL scores with red dashed line.

5.2 Plotting the SIL scores

The Silhouette scores calculated in Figure 23 is plotted as shown in the Figure 25 so that we finally conclude which algorithm performed well to segment the customers related to garden business.

```
# 11.3) Plotting
# Create a data frame with the clustering algorithms and their silhouette scores
silhouette_data <- data.frame(
  Algorithm = c("K-means", "Hierarchical Clustering", "DBSCAN", "GMM"),
  Silhouette_Score = c(0.2784522, 0.4766735, 0.5881333, 0.5021256)
)

# Plot the silhouette scores
library(ggplot2)
ggplot(silhouette_data, aes(x = Algorithm, y = Silhouette_Score, fill = Algorithm)) +
  geom_bar(stat = "identity", color = "black", show.legend = FALSE) +
  geom_text(aes(label = round(Silhouette_Score, 3)), vjust = -0.3, size = 4) +
  labs(
    title = "Comparison of Silhouette Scores for Clustering Algorithms",
    x = "Clustering Algorithm",
    y = "Average Silhouette Score"
  ) +
  ylim(0, 0.7) +
  theme_minimal()
```

Figure 25: Plotting the SIL scores.

References

Shahapure, K. R. and Nicholas, C. (2020). Cluster Quality Analysis Using Silhouette Score, *IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA),* Sydney, NSW, Australia, 2020, pp. 747-748.