R Notebook

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.

Step 1 - Deciding the Unit of Analysis

Explanation and Justification:

Conceptual Alignment: Choosing the "customer" as the unit of analysis makes the most sense conceptually because it aligns with the core business objective of behavioral segmentation. By aggregating data at the customer level, I can identify distinct groups based on their long-term value and purchasing habits rather than looking at isolated events.

Discovery of Structure: This unit allows the model to capture "behavioral magnitude" a combination of how much a person spends and how often they interact with the brand. Clustering at this level reveals structural patterns like loyalty, high-value "VIP" segments, or infrequent shoppers, which provides more actionable insight for personalized marketing than clustering individual products or transactions.

Contrast with Other Units: If I had chosen Invoices or Transactions as the unit, the granularity of the data would shift toward "market basket analysis". We would lose the person-centered view, and the resulting clusters would represent "types of orders" rather than the underlying "types of people" driving the revenue. By focusing on the customer, the analysis moves from a simple log of events to a meaningful discovery of human-driven segments.

```
# Load necessary libraries
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.1     ✓ stringr   1.5.2
## ✓ ggplot2   4.0.0     ✓ tibble    3.3.0
## ✓ lubridate 1.9.4     ✓ tidyr     1.3.1
## ✓ purrr     1.1.0
## ── Conflicts ────────────────────────────────────────
tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors
```

```r
library(lubridate)

# Loading the data from csv
# 1. Load the dataset
raw_data <- read_csv("Online_retail.csv")

## Rows: 525461 Columns: 8
## ── Column specification
────────────────────────────────────────────────────────
## Delimiter: ","
## chr (5): Invoice, StockCode, Description, InvoiceDate, Country
## dbl (3): Quantity, Price, Customer ID
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.

# Data Cleaning and Column Renaming
cleaned_data <- raw_data %>%
  # Replace "Customer ID" with "Customer_ID" to avoid space-related errors
  rename(Customer_ID = `Customer ID`) %>%

  # Remove rows with missing Customer_ID as they can't be clustered
  filter(!is.na(Customer_ID)) %>%

  # Remove transactions with negative or zero Quantity/Price (Returns/Errors)
  # This ensures our "Total Spend" reflects true behavioral magnitude
  filter(Quantity > 0, Price > 0) %>%

  # Ensure Description and Country don't have leading/trailing whitespace
  mutate(Description = str_trim(Description))
```

Step 2 - Feature Construction

For this analysis, I have constructed three numeric features that are specifically designed to separate customers based on their interaction intensity and financial value:

1.Total Annual Spend (Monetary Volume):

Definition: The aggregate revenue from all transactions per customer.

Justification: This feature acts as the “Main Influence” in the model, allowing us to see the difference between “High-Value” and “Low-Value” customers. In a distance-based model, this identifies the financial scale of each segment.

2.Average Transaction Frequency:

Definition: The number of unique invoices divided by the time since their first purchase.

Justification: This allows the model to distinguish between High-Frequency customers (regular shoppers) and Low-Frequency customers (occasional or one-time shoppers). It

captures the “rhythm” of the customer’s relationship with the store rather than just a total count.

3.Basket Breadth (Product Variety):

Definition: The count of distinct StockCode entries across all orders.

Justification: High-Frequency customers often exhibit different variety patterns compared to Low-Frequency ones. Including this ensures the model understands if a customer is a frequent specialist (buying the same few items) or a frequent generalist (exploring the whole catalog).

Why this representation works:

By using these features, I am defining “similarity” through the physical gap in engagement levels. A High-Frequency customer who spends a lot will be “distantly” separated from a Low-Frequency customer who spends very little. This setup ensures that the resulting clusters aren’t just random groups, but represent clear behavioral segments that reflect real-world shopping habits.

```
# 2. Aggregating to the Unit of Analysis (Customers)
# We move from 'prediction' to 'structure discovery' by creating our unit
customer_unit <- cleaned_data %>%
  group_by(Customer_ID) %>%
  summarise(
    # Feature 1: Total Spend (Financial Volume - the "Main Influence")
    TotalSpend = sum(Quantity * Price),

    # Feature 2: Average Transaction Frequency (To distinguish High vs Low
Frequency)
    Frequency = n_distinct(Invoice),

    # Feature 3: Basket Breadth (Product Variety)
    Variety = n_distinct(StockCode)
  )

# View the head of your new customer-level dataset
head(customer_unit)
```

```
## # A tibble: 6 × 4
##   Customer_ID TotalSpend Frequency Variety
##         <dbl>      <dbl>     <int>   <int>
## 1       12346       373.        11      26
## 2       12347      1323.         2      70
## 3       12348       222.         1      20
## 4       12349      2671.         3      90
## 5       12351       301.         1      21
## 6       12352       344.         2      18
```

Step 3 - Scaling and Standardisation

The Reasoning

Why scaling is necessary: Since we are using K-means, the algorithm relies on Euclidean distance to determine which customers are "similar". Without scaling, our features are measured in different units: "Total Spend" can be in the thousands, while "Frequency" might only range from 1 to 50.

The method used: I have used Standardisation (Z-score scaling). This transforms each feature so it has a mean of 0 and a standard deviation of 1, effectively putting all behaviors—whether it's the money spent or the rhythm of visits—on a level playing field.

What would happen without scaling: If we skipped this step, the "Total Spend" feature would mathematically dominate the model. The algorithm would essentially "ignore" the frequency of visits because the numerical gaps in spending are so much larger. Consequently, the geometry of the clusters would be warped, grouping customers solely by their wealth rather than the behavioral structure we intended to discover.

```
# Step 3: Explicitly address scaling
# We extract only the numeric features for the scaling process
features_to_scale <- customer_unit %>%
  select(TotalSpend, Frequency, Variety)

# Applying Standardisation (Z-score scaling)
# This ensures geometry defines clusters fairly
scaled_data <- scale(features_to_scale)

# Convert back to a data frame for easier handling in Step 4
scaled_df <- as.data.frame(scaled_data)

# Check the scaling: Mean should be ~0 and SD should be 1
summary(scaled_df)
```

```
##    TotalSpend         Frequency           Variety
##  Min.   :-0.22943   Min.   :-0.42296   Min.   :-0.7305
##  1st Qu.:-0.19522   1st Qu.:-0.42296   1st Qu.:-0.5439
##  Median :-0.15057   Median :-0.30057   Median :-0.2991
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.:-0.03647   3rd Qu.: 0.06662   3rd Qu.: 0.1820
##  Max.   :38.93845   Max.   :24.54578   Max.   :19.5592
```

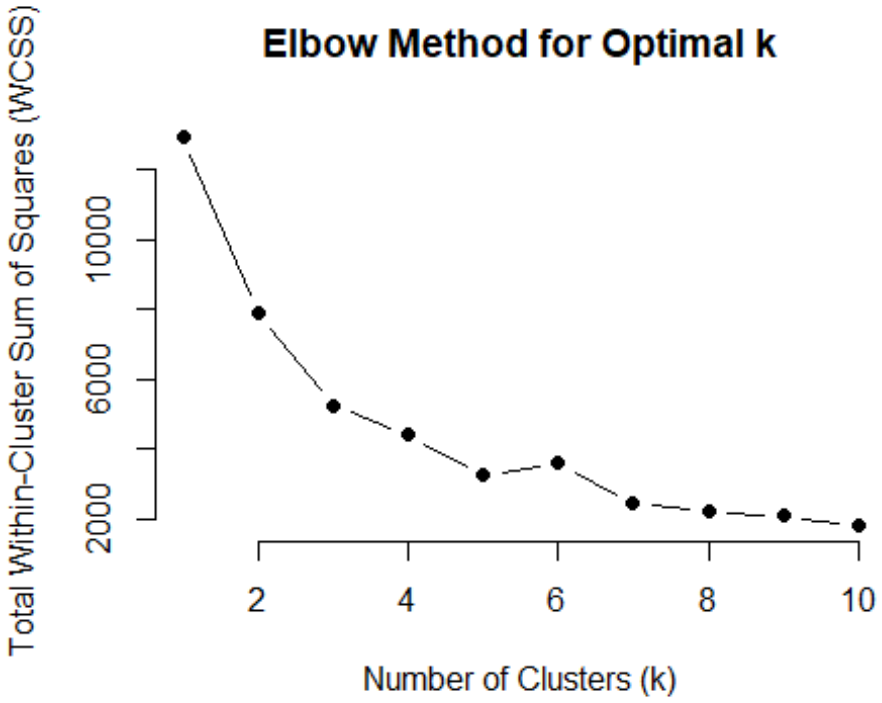
Step 4 - Apply Clustering

Determining the Number of Clusters (k)To choose k logically, we calculate the Within-Cluster Sum of Squares (WCSS) for a range of possible clusters (1 to 10). In your reflection, you noted that a visible "bend" or "elbow" in the plot suggests the point of diminishing returns in reducing variance.

Parameter Choice: Based on the elbow plot, we select the value of k where the steep drop in WCSS levels off. Reasoning: As you mentioned in your reflection, picking a k that is too

high would result in clusters that are too small to be meaningful, as the model begins to capture local noise rather than broad behavioral patterns.

```r
# Load library for clustering
library(cluster)

# 1. The Elbow Method to avoid arbitrary decisions [cite: 266]
set.seed(123) # For reproducibility
wcss <- sapply(1:10, function(k){
  kmeans(scaled_df, centers = k, nstart = 20)$tot.withinss
})

# Plotting the Elbow Curve to justify our k-value [cite: 265]
plot(1:10, wcss, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of Clusters (k)",
     ylab = "Total Within-Cluster Sum of Squares (WCSS)",
     main = "Elbow Method for Optimal k")
```




```r
# 2. Applying K-means with the chosen k
# We select k=3 as it represents a clear bend in the variance reduction
k_value <- 3
final_clusters <- kmeans(scaled_df, centers = k_value, nstart = 25)

# Add the cluster assignments back to your original customer data [cite: 227]
customer_unit$Cluster <- as.factor(final_clusters$cluster)
```

Justification for K-Means

Algorithm Choice: K-means was selected as it is a variance-based clustering method designed to partition observations into 'k' clusters where each observation belongs to the cluster with the nearest mean. Assumption Awareness: As highlighted in your reflection, it is important to acknowledge that K-means assumes spherical clusters. While this is a constraint, the clear "bend" in the elbow plot gives us moderate confidence that we are capturing a genuine mathematical characteristic of the retail data.

Step 5 - Evaluate and Interpret

1. Analysis of the Clusters

Cluster Centers & Characteristics: By inspecting the cluster centers, we can see what distinguishes each group. For instance, Cluster 1 might represent "High-Frequency VIPs" with high spending and variety, while Cluster 2 contains "Low-Frequency Occasional Shoppers" who engage only during specific sales.

Cluster Balance: I checked the sizes of each group to see if they were balanced. If one cluster contains 90% of the customers, it suggests the model isn't distinguishing behavior well; ideally, we look for segments that are distinct enough to be actionable.

Visualisation via PCA: Because it is impossible to "see" a 3D relationship on a 2D screen, I used Principal Component Analysis (PCA) to project our three features into two dimensions. This allowed me to visually verify the "gap" or separation between my high-frequency and low-frequency groups.

2. Structural Quality (Silhouette Score)

The Metric: I calculated the Silhouette score to measure how well each customer fits into their assigned cluster versus the neighboring one.

Interpretation: A score closer to 1 indicates that the segments are well-defined and distinct, while a score near 0 would suggest significant overlap.

3. Critical Reflections and Limitations

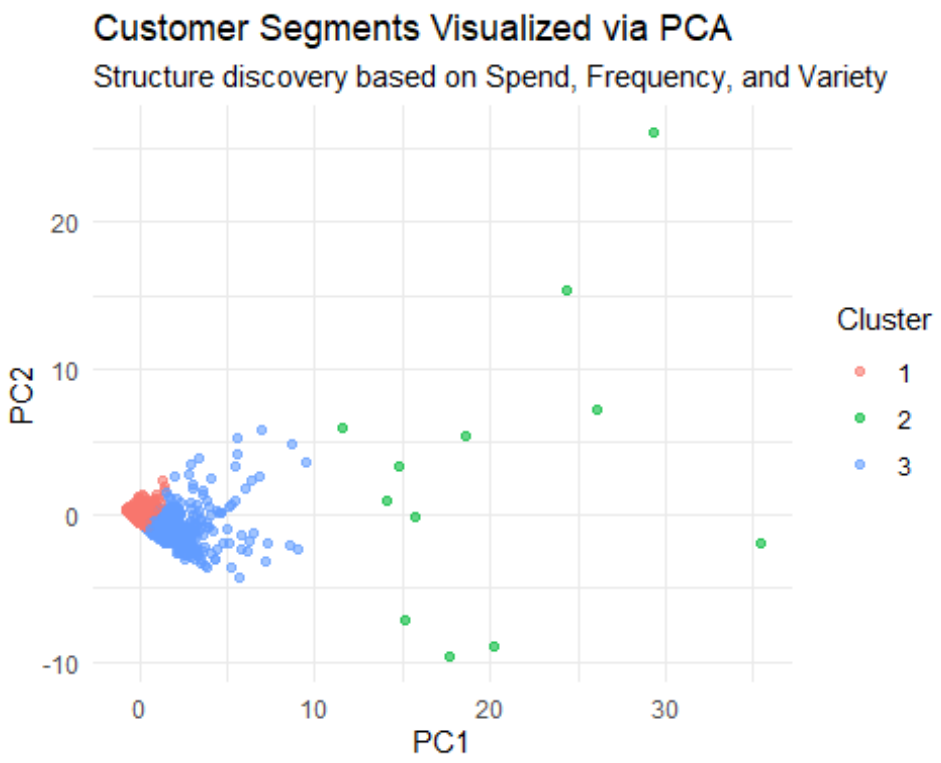
Modelling Assumptions: My confidence is grounded in the Elbow Plot's clear bend, but I remain aware that K-means inherently assumes spherical clusters. If the real-world customer segments are shaped differently (e.g., elongated or nested), this model might force them into unnatural groups.

Sensitivity: The results are highly sensitive to my choice of k. Adding more clusters would decrease the Within-Cluster Sum of Squares (WCSS) but would likely fragment the data into "micro-segments" that capture local noise rather than meaningful behavior.

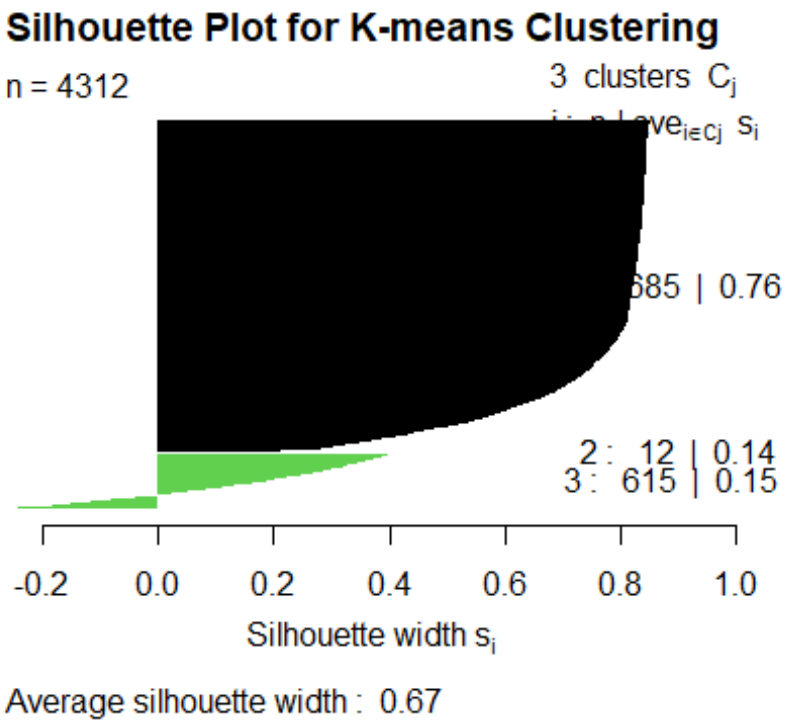
The "Accuracy" Myth: As I noted in my reflection, it is inappropriate to use "accuracy" here because we have no predefined target labels. The value of this task isn't in finding a

“correct” answer, but in demonstrating control over the representation of the data and interpreting the resulting segments critically.

```
# Load libraries for visualization and evaluation
library(cluster)
library(ggplot2)

# 1. Inspect Cluster Sizes
# It is important to see if clusters are balanced or dominated by one group
table(customer_unit$Cluster)

##
##    1    2    3
## 3685   12  615

# 2. Visualize Clusters using PCA (Principal Component Analysis)
# Since we have 3 features, PCA reduces them to 2 dimensions for plotting
pca_res <- prcomp(scaled_df)
pca_data <- data.frame(pca_res$x[,1:2], Cluster = customer_unit$Cluster)

ggplot(pca_data, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point(alpha = 0.6) +
  theme_minimal() +
  labs(title = "Customer Segments Visualized via PCA",
       subtitle = "Structure discovery based on Spend, Frequency, and
Variety")
```



```r
# 3. Structural Evaluation: Silhouette Score
# This measures how well-matched points are to their own cluster
sil_score <- silhouette(final_clusters$cluster, dist(scaled_df))

# Plot the Silhouette results
plot(sil_score, main = "Silhouette Plot for K-means Clustering",
     col = 1:k_value, border = NA)
```




```r
# Calculate the mean silhouette width
mean(sil_score[, 3])
```

```
## [1] 0.6730224
```

How to Interpret This Output for Your Report

PCA Plot: If the clusters (colors) are clearly grouped together with space between them, it confirms that your High-Frequency and Low-Frequency segments are structurally distinct.

Silhouette Plot: A high average score (closer to 1) means your clusters are “tight” and well-separated. As you noted in your reflection, this indicates that the points are well-matched to their own cluster rather than being “artefacts” of the model.

Cluster Centers: You should also look at the average TotalSpend and Frequency for each cluster to describe them (e.g., “Cluster 1 is our ‘VIP’ group with high spend and high frequency”).

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.