Madhav Institute of Technology & Science Gwalior (M.P.)

Deemed to be University

(Declared under Distinct Category by Ministry of Education, Government of India)

NAAC ACCREDITED WITH A++ GRADE

DEPARTMENT OF ELECTRONICS ENGINEERING

2024 – 2025

A Skill-Based Mini Project Report

On

“Write a program for Customer Segmentation”

Artificial Inteligence and Machine Learning (2140617)

Bachelor of Technology

In

Electronics Engineering

Submitted By:

Suryansh Dixit
0901EC221130

Submitted To:

Dr. Varun Sharma
Assistant Proffesor

CERTIFICATE

This is certified that Suryansh Dixit (0901EC221130) has submitted the project report titled

Write a program for customer segmentation.

under the mentorship of **Dr. Varun Sharma**, in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in Electronics Engineering from Madhav Institute of Technology and Science, Gwalior.

Suryanshdixit

Suryansh Dixit
0901EC221130

[illegible]

Dr. Varun Sharma
Assistant Professor
Department of Electronics Engineering
MITS, Gwalior

DECLARATION

I at this moment declare that the work presented in this project report, for the partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in Electronics Engineering at Madhav Institute of Technology & Science, Gwalior is an authenticated and original record of my work under the mentorship of **Dr. Varun Sharma,** Assistant Professor, **Department of Electronics Engineering, MITS Gwalior.**

I declare that I have not submitted the matter embodied in this report for the award of any degree or diploma anywhere else.

Suryansh Dixit
0901EC221130

ACKNOWLEDGEMENTS

The full semester project has proved to be pivotal to my career. We are thankful to our institute, **Madhav Institute of Technology and Science** to allow us to continue our disciplinary/interdisciplinary project as a curriculum requirement, under the provisions of the Flexible Curriculum Scheme (based on the AICTE Model Curriculum 2018), approved by the Academic Council of the institute.

I would like to express my sincere gratitude to my mentor Dr. Varun Sharma for providing her invaluable guidance, comments and suggestions throughout the course of the project and for constantly motivating me to work harder.

Suryansh Dixit
0901EC221130

CONTENT

- Certificate 2
- Declaration 3
- Acknowledgements 4

1. Introduction 6
   1.1 What is customer segmentation? 6
   1.2 What is RMF analysis? 6
   1.3 K-Mean algorithm 7

2. Code 9
   2.1 Break-down of Logic 14

3. Output 19
   3.1 Visualisation 22

4. Conclusion 26

References 26

Introduction

Artificial Intelligence and Machine Learning (AIML) offer powerful tools for uncovering patterns and insights within large datasets, significantly impacting business strategy. One critical application lies in customer analytics, particularly customer segmentation. By grouping customers with similar traits, businesses can optimize marketing efforts and enhance customer engagement. This report focuses on a Python program created to perform customer segmentation using AIML techniques. The program analyzes historical transaction data [Optional: mention the source] to compute Recency, Frequency, and Monetary (RFM) scores for each customer. Subsequently, the unsupervised K-Means clustering algorithm is applied to these RFM features to partition the customer base into distinct, data-driven segments.

1.1 What is customer segmentation?

Customer segmentation is the process of dividing a customer base into distinct groups based on shared characteristics like demographics, behavior, or preferences. This allows businesses to understand their customers better and tailor their marketing, sales, and product offerings to meet specific needs, ultimately increasing the effectiveness of their efforts.

Benefits of customer segmentation:

- **Enhanced Customer Understanding:** Gain deeper insights into different customer groups and their needs.
- **More Effective Marketing:** Reach the right customers with the right message at the right time.
- **Improved Sales:** Tailor sales strategies to specific customer segments, increasing conversion rates.
- **Better Product Development:** Identify unmet customer needs and develop products that better meet those needs.
- **Increased Customer Loyalty:** Personalized experiences and targeted offers can foster stronger relationships with customers.

1.2 What is RMF analysis?

RFM analysis is a marketing technique used to quantitatively rank and group customers based on the recency, frequency and monetary total of their recent transactions to identify the best customers and perform targeted marketing campaigns. The system assigns each customer numerical scores based on these factors to provide an objective analysis.

Recency: Days since last purchase
Frequency: Count of unique invoices
Monetary: Sum of total purchases

RFM analysis ranks each customer on the following factors:

- **Recency**: How recent was the customer's last purchase? Customers who recently made a purchase will still have the product on their mind and are more likely to purchase or use the product again. Businesses often measure recency in days. But, depending on the product, they may measure it in years, weeks or even hours.
- **Frequency**: How often did this customer make a purchase in a given period? Customers who purchased once are often more likely to purchase again. Additionally, first time customers may be good targets for follow-up advertising to convert them into more frequent customers.
- **Monetary**: How much money did the customer spend in a given period? Customers who spend a lot of money are more likely to spend money in the future and have a high value to a business.

1.3 K – Means Algorithm

K-means clustering is an unsupervised machine learning algorithm used to group similar data points into clusters based on their features. It aims to minimize the distance between each data point and the centroid of its assigned cluster. The algorithm iteratively assigns data points to clusters and updates the cluster centroids until the cluster assignments stabilize.

Key Concepts:

- **Unsupervised Learning:** K-means is an unsupervised algorithm, meaning it doesn't require labeled data. It discovers patterns and groups within the data itself.
- **Clustering:** K-means groups similar data points together into clusters.
- **Centroids:** Each cluster is represented by a centroid, which is the center of the cluster.
- **Iteration:** The algorithm works iteratively, assigning data points to clusters and updating centroids until convergence.
- **Distance Metric:** K-means uses a distance metric (e.g., Euclidean distance) to measure the similarity between data points and centroids.
- **K (Number of Clusters):** A user-defined parameter that specifies the desired number of clusters.

How it Works:

1. **Initialization:** Randomly choose K initial centroids.
2. **Assignment:** Assign each data point to the closest cluster based on its distance to the centroids.
3. **Update**: Calculate the new centroids by taking the mean of all data points assigned to each cluster.
4. **Repeat**: Repeat steps 2 and 3 until the cluster assignments no longer change (convergence).

Applications:

- **Market Segmentation**: Group customers based on their purchasing behavior.
- **Image Compression:** Reduce image size by clustering pixels.
- **Document Clustering:** Organize documents into topics.
- **Anomaly Detection:** Identify unusual data points that don't belong to any cluster.

Limitations:

- **Sensitivity to Initial Centroid Placement**: K-means can be sensitive to the initial placement of centroids, which may lead to different clusterings.
- **Assumes Spherical Clusters**: K-means assumes that clusters are roughly spherical or compact, which may not always be the case.
- **Requires Pre-defined K**: The number of clusters (K) must be specified in advance, which can be challenging.

CODE

```
# -----------------------------------------------
# Customer Segmentation using K-Means and RFM
# Based on the provided transaction data (01.csv)
# -----------------------------------------------

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import datetime as dt
import warnings

warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=UserWarning)

print("--- Customer Segmentation Program (RFM Analysis) ---")

# --- 1. Load Data ---
file_path = "D:\DEVELOPER\PYTHON\Code\SBMP AIML\Retail_dataset.csv"
print(f"\n[1] Loading Data from {file_path}...")

try:
    # Read the CSV, handling potential encoding issues (like BOM) and parsing dates
    # Specify the date format explicitly if pandas struggles to infer it
    data = pd.read_csv(file_path, encoding='utf-8-sig')
    # Try parsing the date column - adjust format if necessary
    try:
        data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'], format='%d-%m-%Y %H:%M')
    except ValueError:
        print("Warning: Could not parse 'InvoiceDate' with format '%d-%m-%Y %H:%M'. Attempting automatic parsing.")
        data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

    print("Data loaded successfully.")
    print("\nOriginal Data Info:")
    data.info()
    print("\nSample Data Head:")
    print(data.head())

except FileNotFoundError:
    print(f"Error: File not found at {file_path}. Please check the path and filename.")
    exit() # Stop execution if file not found
except Exception as e:
    print(f"An error occurred during data loading: {e}")
    exit()

# --- 2. Data Cleaning & Preparation ---
print("\n[2] Cleaning and Preparing Data...")

# Check for missing values
print("\nMissing values before cleaning:")
print(data.isnull().sum())

# Handle missing CustomerID - Segmentation requires customer identification
initial_rows = data.shape[0]
data.dropna(subset=['CustomerID'], inplace=True)
print(f"\nRemoved {initial_rows - data.shape[0]} rows with missing CustomerID.")

# Convert CustomerID to integer
data['CustomerID'] = data['CustomerID'].astype(int)

# Remove cancelled orders (InvoiceNo starts with 'C') or negative quantities
# Also remove transactions with zero or negative unit price as they don't contribute to monetary value meaningfully
original_rows = data.shape[0]
data = data[~data['InvoiceNo'].astype(str).str.startswith('C')]
data = data[data['Quantity'] > 0]
data = data[data['UnitPrice'] > 0]
print(f"Removed {original_rows - data.shape[0]} rows corresponding to cancellations, negative/zero quantity or price.")

# Calculate Total Price for each item
data['TotalPrice'] = data['Quantity'] * data['UnitPrice']

print("\nData after initial cleaning:")
print(data.head())
data.info() # Check data types again

# --- 3. Calculate RFM Features ---
print("\n[3] Calculating RFM Features...")

# Find the most recent date in the dataset to use as a reference point (snapshot date)
# Add one day to make sure even the latest transaction has a recency > 0
snapshot_date = data['InvoiceDate'].max() + dt.timedelta(days=1)
print(f"Snapshot date for Recency calculation: {snapshot_date.strftime('%Y-%m-%d')}")

# Aggregate data by CustomerID
rfm_df = data.groupby('CustomerID').agg({
    'InvoiceDate': lambda date: (snapshot_date - date.max()).days, # Recency: Days since last purchase
    'InvoiceNo': 'nunique',                                         # Frequency: Count of unique invoices
    'TotalPrice': 'sum'                                             # Monetary: Sum of total purchases
})
# Rename columns for clarity
rfm_df.rename(columns={'InvoiceDate': 'Recency',
                 'InvoiceNo': 'Frequency',
                 'TotalPrice': 'MonetaryValue'}, inplace=True)

print("\nRFM Dataframe Head:")
print(rfm_df.head())
print("\nDescriptive Statistics for RFM:")
print(rfm_df.describe())

# --- Handle Skewness ---
# RFM features often have skewed distributions. Log transformation can help.
# Add 1 to avoid log(0) issues.
print("\nApplying log transformation to handle skewness in F and M (and potentially R)...")
rfm_log = rfm_df.copy()
# Check for non-positive values before log transform (should be okay after cleaning, but good practice)
if (rfm_log['Recency'] <= 0).any() or (rfm_log['Frequency'] <= 0).any() or (rfm_log['MonetaryValue'] <= 0).any():
    print("Warning: Found non-positive values before log transform. This might indicate data issues.")
    rfm_log['Recency'] = rfm_log['Recency'].apply(lambda x: max(x, 1)) # Ensure Recency >= 1
    rfm_log['Frequency'] = rfm_log['Frequency'].apply(lambda x: max(x, 1)) # Ensure Frequency >= 1
    rfm_log['MonetaryValue'] = rfm_log['MonetaryValue'].apply(lambda x: max(x, 0.01))

rfm_log['Recency'] = np.log1p(rfm_log['Recency']) # log1p calculates log(1 + x)
rfm_log['Frequency'] = np.log1p(rfm_log['Frequency'])
rfm_log['MonetaryValue'] = np.log1p(rfm_log['MonetaryValue'])

print("\nRFM Dataframe Head (Log Transformed):")
print(rfm_log.head())

# Visualize distributions after log transform (optional but recommended)
plt.figure(figsize=(15, 5))
for i, feature in enumerate(['Recency', 'Frequency', 'MonetaryValue']):
    plt.subplot(1, 3, i + 1)
    sns.histplot(rfm_log[feature], kde=True) # type: ignore
    plt.title(f'Distribution of Log({feature})')
plt.tight_layout()
plt.suptitle('RFM Feature Distributions (Log Transformed)', y=1.02)
plt.show()

# --- 4. Preprocessing RFM Data ---
print("\n[4] Preprocessing RFM Data (Scaling)...")
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm_log) # Scale the log-transformed data

print("RFM data scaled using StandardScaler.")
# print("\nScaled RFM Data Sample:\n", rfm_scaled[:5]) # Optional

# --- 5. Determine Optimal Number of Clusters (K) ---
print("\n[5] Determining Optimal Number of Clusters (K)...")

wcss = [] # Within-Cluster Sum of Squares (Inertia)
silhouette_scores = []
k_range = range(2, 11) # Test K from 2 to 10

for k in k_range:
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=10, random_state=42)
    kmeans.fit(rfm_scaled)
    wcss.append(kmeans.inertia_)
    score = silhouette_score(rfm_scaled, kmeans.labels_)
    silhouette_scores.append(score)
    print(f" K={k}, WCSS={kmeans.inertia_:.2f}, Silhouette Score={score:.3f}")

# Plotting the results
plt.figure(figsize=(12, 5))

# Elbow Method Plot
plt.subplot(1, 2, 1)
plt.plot(k_range, wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS (Inertia)')
plt.grid(True)
```

```
# Silhouette Score Plot
plt.subplot(1, 2, 2)
plt.plot(k_range, silhouette_scores, marker='o', linestyle='--')
plt.title('Silhouette Scores for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.tight_layout()
plt.show()

# --- Choosing K ---
# Analyze the plots. Look for the "elbow" in WCSS and the peak in Silhouette Score.
# For RFM data, 3-5 clusters are often meaningful. Let's start with 4, but adjust based on your plots.
optimal_k = 4 # <<< ADJUST THIS BASED ON YOUR PLOTS <<<
print(f"\nBased on Elbow Method and Silhouette Score analysis, choosing K = {optimal_k}")

# --- 6. Apply K-Means Clustering ---
print(f"\n[6] Applying K-Means with K = {optimal_k}...")
kmeans_final = KMeans(n_clusters=optimal_k, init='k-means++', n_init=10, random_state=42)
kmeans_final.fit(rfm_scaled)

# Add the cluster labels back to the ORIGINAL RFM dataframe (before log/scaling) for interpretation
rfm_df['Cluster'] = kmeans_final.labels_
print("Cluster labels assigned to RFM data.")
print("\nRFM Data Head with Cluster Labels:")
print(rfm_df.head())

# --- 7. Analyze and Visualize Clusters ---
print("\n[7] Analyzing and Visualizing Clusters...")

# Calculate average RFM values for each cluster (using the original, non-logged values)
cluster_summary = rfm_df.groupby('Cluster').agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'MonetaryValue': 'mean',
    'Cluster': 'size' # Get cluster size
}).rename(columns={'Cluster': 'Cluster Size'})

print("\nCluster Summary (Mean RFM Values & Size):")
print(cluster_summary.round(2)) # Round for readability

# --- Visualizations (using original RFM values for interpretability) ---

# Scatter plot: Recency vs MonetaryValue
plt.figure(figsize=(10, 7))
sns.scatterplot(data=rfm_df, x='Recency', y='MonetaryValue', hue='Cluster', palette='viridis', s=50, alpha=0.7)
plt.title(f'Customer Segments by Recency vs Monetary Value (K={optimal_k})')
plt.xlabel('Recency (Days since last purchase)')
plt.ylabel('Monetary Value (Total Spent)')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()

# Scatter plot: Frequency vs MonetaryValue
plt.figure(figsize=(10, 7))
sns.scatterplot(data=rfm_df, x='Frequency', y='MonetaryValue', hue='Cluster', palette='viridis', s=50,
alpha=0.7)
```

```
plt.title(f'Customer Segments by Frequency vs Monetary Value (K={optimal_k})')
plt.xlabel('Frequency (Number of Purchases)')
plt.ylabel('Monetary Value (Total Spent)')
plt.legend(title='Cluster')
# Use log scale for axes if distribution is very skewed, even after clustering
# plt.xscale('log')
# plt.yscale('log')
plt.grid(True)
plt.show()

# Scatter plot: Recency vs Frequency
plt.figure(figsize=(10, 7))
sns.scatterplot(data=rfm_df, x='Recency', y='Frequency', hue='Cluster', palette='viridis', s=50, alpha=0.7)
plt.title(f'Customer Segments by Recency vs Frequency (K={optimal_k})')
plt.xlabel('Recency (Days since last purchase)')
plt.ylabel('Frequency (Number of Purchases)')
plt.legend(title='Cluster')
# plt.yscale('log') # Frequency might benefit from log scale for visualization
plt.grid(True)
plt.show()

# Box plots to see RFM distributions per cluster
for feature in ['Recency', 'Frequency', 'MonetaryValue']:
    plt.figure(figsize=(8, 5))
    sns.boxplot(data=rfm_df, x='Cluster', y=feature, palette='viridis', showfliers=False) # Hide outliers for clarity
    plt.title(f'{feature} Distribution per Cluster (K={optimal_k})')
    # if feature in ['Frequency', 'MonetaryValue']: # Use log scale if needed for visualization
    #     plt.yscale('log')
    plt.show()

# --- 8. Interpretation ---
print("\n--- [8] Interpretation Guide ---")
print("Analyze the 'Cluster Summary (Mean RFM Values & Size)' and the plots.")
print("Assign descriptive names to each cluster based on their RFM characteristics:")
print(" - Low Recency = Recent customer")
print(" - High Frequency = Buys often")
print(" - High MonetaryValue = Spends a lot")
print("\nExample Interpretations (modify based on YOUR results):")
print(" * Cluster with Low R, High F, High M: Best Customers / Champions")
print(" * Cluster with High R, Low F, Low M: Lost Customers / At Risk (Churned)")
print(" * Cluster with Low R, High F, Mid/Low M: Loyal Customers (Frequent but lower spend)")
print(" * Cluster with Low R, Low F, Low M: New Customers")
print(" * Cluster with High R, High F, High M: Can't Lose Them (High value, but becoming inactive)")
print(" * Cluster with Mid R, Mid F, Mid M: Potential Loyalists / Needs Attention")
print("\nLook at your specific 'Cluster Summary' table to name your segments accurately.")

# --- Saving Results (Optional) ---
output_filename = 'customer_segments_rfm.csv'
# Save the RFM dataframe with cluster labels
try:
    rfm_df.reset_index().to_csv(output_filename, index=False) # Reset index to include CustomerID as a column
    print(f"\nSegmented RFM data saved to {output_filename}")
except Exception as e:
    print(f"\nError saving results to CSV: {e}")
print("\n--- Program End ---")
```

2.1 Break-down of logic of the code

1. **Import Libraries:**
    - **pandas:** Essential for data manipulation. Used for reading the CSV, creating DataFrames, cleaning data, grouping, and aggregating.
    - **numpy:** Used for numerical operations, particularly the logarithm (np.log1p) for handling data skewness.
    - **matplotlib.pyplot & seaborn:** Used for creating visualizations (histograms, scatter plots, box plots) to understand the data and the resulting clusters.
    - **sklearn.cluster.KMeans:** The implementation of the K-Means clustering algorithm.
    - **sklearn.preprocessing.StandardScaler:** Used to scale the data before applying K-Means, making sure all features contribute equally to the distance calculations.
    - **sklearn.metrics.silhouette_score:** A metric used to help evaluate the quality of the clustering and determine a good number of clusters (K).
    - **datetime (dt):** Used for date and time operations, specifically to calculate Recency.
    - **warnings:** Used to suppress specific warning messages that might clutter the output but don't necessarily indicate a problem (like future warnings from libraries).

2. **Load Data ([1] Loading Data...)**
    - **What:** Reads the 01.csv file into a pandas DataFrame.
    - **Why:** To get the raw transactional data into memory for processing.
    - **How:** Uses pd.read_csv(). It specifies encoding='utf-8-sig' to handle potential Byte Order Marks (like the ufeff sometimes seen at the start of CSVs). It attempts to parse the InvoiceDate column into proper datetime objects, trying a specific format first (%d-%m-%Y %H:%M) and falling back to automatic parsing if that fails. Includes error handling for FileNotFoundError.

3. **Data Cleaning & Preparation ([2] Cleaning and Preparing Data...)**
    - **What:** Prepares the raw data for RFM analysis. This involves several crucial steps:
        - **Handle Missing CustomerID:** Removes rows where CustomerID is missing.
        - **Remove Cancellations/Returns:** Filters out transactions where InvoiceNo starts with 'C' (indicating cancellations) or where Quantity is zero or negative.
        - **Remove Zero/Negative Prices**: Filters out transactions where UnitPrice is zero or negative.

        - **Calculate TotalPrice:** Creates a new column by multiplying Quantity and UnitPrice.
    - **Why:**
        - RFM segmentation is customer-centric, so we need CustomerID for every relevant transaction.
        - Cancellations/returns would artificially deflate Monetary value and potentially affect Frequency counts negatively.
        - Meaningful Monetary value cannot be calculated with zero or negative prices/quantities.
        - TotalPrice is needed to calculate the Monetary value for each customer.
    - **How:** Uses pandas filtering (dropna, boolean indexing []), string methods (.str.startswith()), and basic arithmetic operations.

4. **Calculate RFM Features ([3] Calculating RFM Features...)**
    - **What**: Transforms the cleaned transactional data into a customer-centric view with Recency, Frequency, and Monetary value for each unique CustomerID.
    - **Why:** RFM provides standard, interpretable metrics for customer behavior. K-Means will be applied to these derived features, not the raw transactions.
    - **How:**
        - **Snapshot Date:** Determines the most recent date in the dataset and adds one day. This date acts as a reference point ("today") to calculate how *recent* the last purchase was.
        - **Grouping:** Uses data.groupby('CustomerID') to group all transactions belonging to the same customer.
        - **Aggregation (.agg()):** Calculates the three RFM metrics for each customer group:
            - **Recency:** Finds the maximum InvoiceDate for the customer and subtracts it from the snapshot_date to get the number of days since their last purchase. (lambda date: (snapshot_date - date.max()).days)
            - **Frequency:** Counts the number of *unique* InvoiceNo values for that customer. ('nunique')
            - **MonetaryValue:** Sums up the TotalPrice of all transactions for that customer. ('sum')
        - **Renaming**: Renames the aggregated columns to 'Recency', 'Frequency', 'MonetaryValue'.

5. **Handle Skewness (Log Transformation) (Inside [3]...)**
    - **What**: Applies a logarithmic transformation (specifically log(1 + x)) to the Recency, Frequency, and MonetaryValue columns.

    - **Why**: RFM features often have highly skewed distributions (many customers with low values, few with very high values). K-Means works best with roughly normally distributed data because it relies on Euclidean distance. Log transformation compresses the range of large values and expands the range of small values, making the distribution more symmetrical. log1p is used to handle potential zero values gracefully (since log(0) is undefined).
    - **How**: Uses np.log1p() on the RFM columns. Visualizes the transformed distributions using histograms.

6. **Preprocessing RFM Data (Scaling) ([4] Preprocessing RFM Data...)**
    - **What:** Scales the log-transformed RFM features.
    - **Why**: K-Means calculates distances between data points. If features have vastly different scales (e.g., Recency in days, MonetaryValue potentially in thousands), the feature with the larger range will dominate the distance calculation, potentially leading to poor clustering. Scaling (like Standardization) transforms the data so that all features have a similar scale (mean=0, std dev=1).
    - **How**: Uses StandardScaler from scikit-learn. It first fits the scaler to the log-transformed RFM data (to learn the mean and standard deviation) and then transforms the data.

7. **Determine Optimal Number of Clusters (K) ([5] Determining Optimal K...)**
    - **What**: Tries different numbers of clusters (K) and evaluates the results using the Elbow Method and Silhouette Score.
    - **Why**: K-Means requires you to specify the number of clusters beforehand. These methods provide quantitative and visual aids to help choose a K that represents a good trade-off between capturing the structure in the data and avoiding overfitting (too many clusters).
    - **How**:
        - **Loop**: Iterates through a range of potential K values (e.g., 2 to 10).
        - **K-Means Fit**: For each K, fits a K-Means model to the *scaled* RFM data. init='k-means++' provides smart centroid initialization, and n_init=10 runs the algorithm multiple times with different starting centroids and picks the best result to avoid poor local optima. random_state ensures reproducibility.
        - **WCSS (Inertia):** Calculates the Within-Cluster Sum of Squares (kmeans.inertia_). This measures how compact the clusters are. Lower is generally better.
        - **Silhouette Score**: Calculates the silhouette score, which measures how similar a point is to its own cluster compared to other clusters. Values range from -1 to 1; higher values (closer to 1) indicate better-defined clusters.

        - **Plotting**: Visualizes WCSS vs. K (looking for an "elbow" point where the rate of decrease slows) and Silhouette Score vs. K (looking for a peak).
        - **Selection**: Based on the plots, a value for optimal_k is chosen (this often requires human judgment).

8. **Apply K-Means Clustering ([6] Applying K-Means...)**
    - **What**: Runs the K-Means algorithm one final time using the chosen optimal_k. Assigns each customer (each row in the RFM data) to a cluster.
    - **Why**: To perform the actual segmentation based on the chosen number of clusters.
    - **How**: Creates a KMeans object with n_clusters=optimal_k and fits it to the *scaled* RFM data. The resulting cluster assignments (kmeans_final.labels_) are added as a new 'Cluster' column to the *original* (non-logged, non-scaled) rfm_df for easier interpretation later.

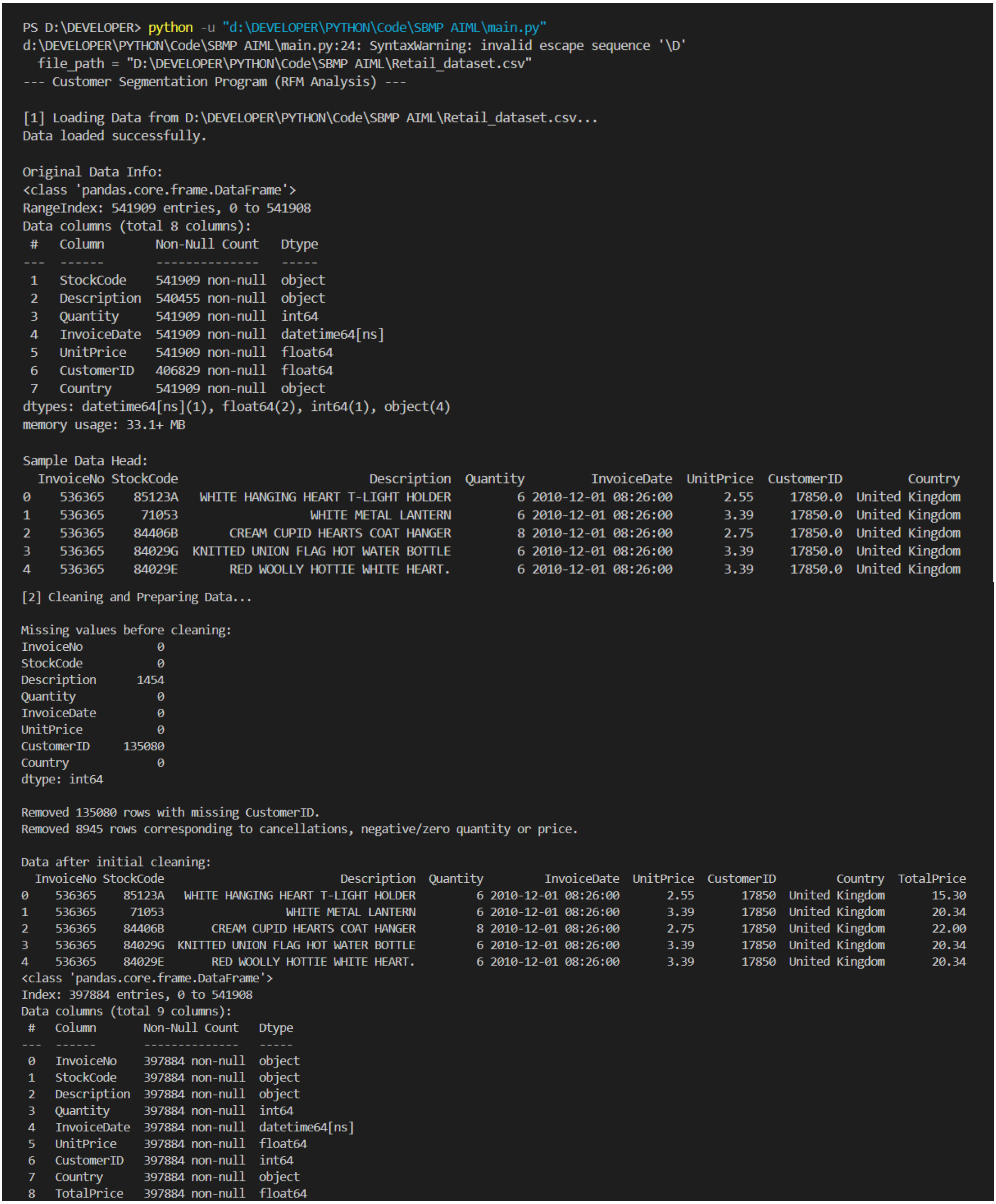
9. **Analyze and Visualize Clusters ([7] Analyzing and Visualizing...)**
    - **What**: Examines the characteristics of each generated cluster and visualizes them.
    - **Why**: To understand what defines each segment. Simply having cluster numbers isn't useful; we need to know *what* kind of customers are in each cluster.
    - **How**:
        - **Cluster Summary**: Groups the rfm_df (which now includes the 'Cluster' column) by cluster and calculates the *mean* Recency, Frequency, and MonetaryValue for each cluster, along with the cluster size (number of customers). This table is crucial for interpretation.
        - **Visualization:** Creates scatter plots (e.g., Recency vs. Monetary, Frequency vs. Monetary) and box plots using the *original* (non-logged, non-scaled) RFM values, colored by cluster. This helps visually grasp the differences between segments. Using original values makes the axes interpretable (e.g., days, counts, currency amounts).

10. **Interpretation ([8] Interpretation Guide...)**
    - **What**: Provides guidance on how to interpret the cluster summary table and plots.
    - **Why**: This is the final, crucial step where data science translates into business insight. The numerical clusters need meaningful labels.
    - **How**: Explains how to look at the average R, F, and M values for each cluster and assign descriptive names (like "Champions", "Loyal

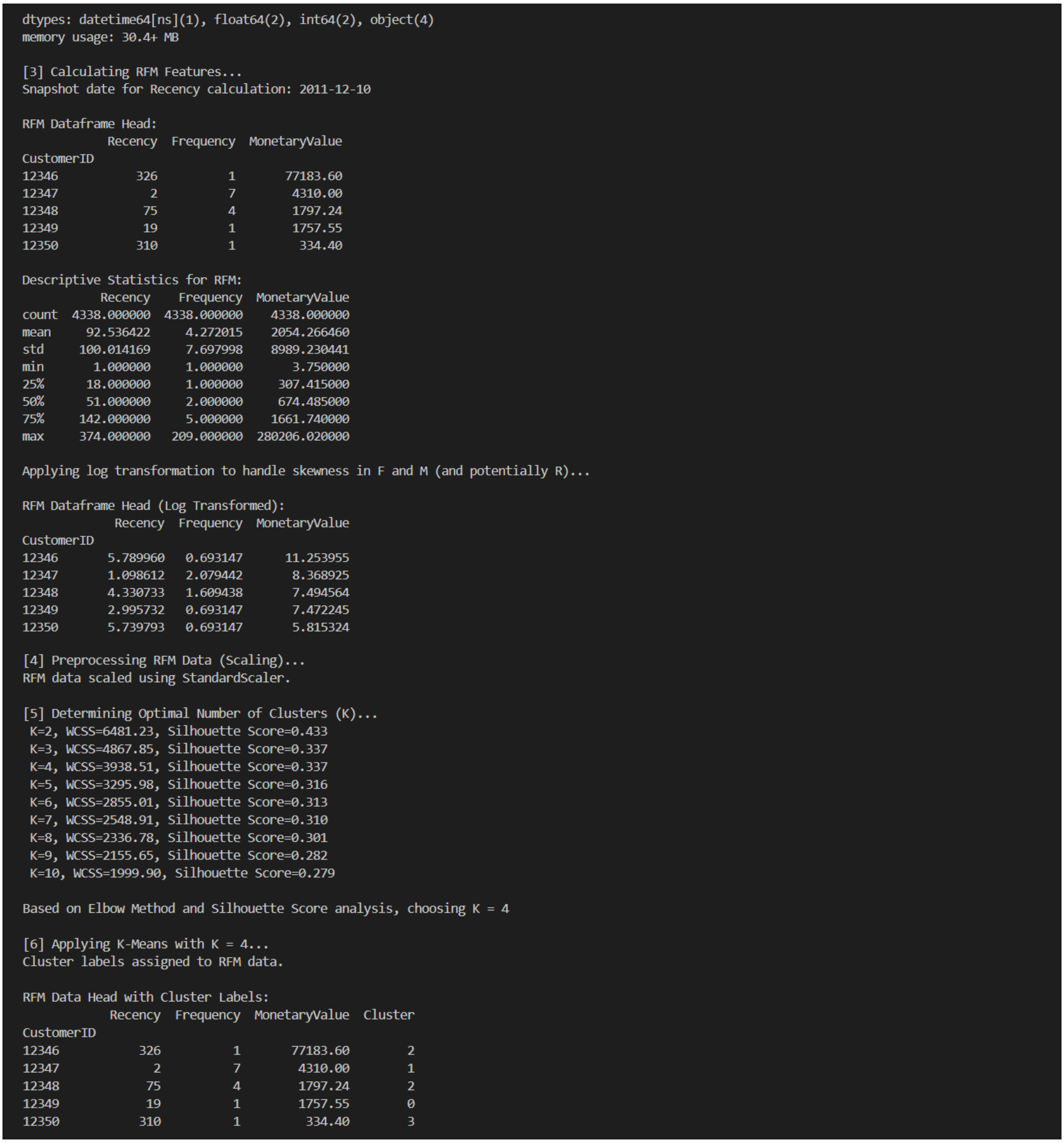
Customers", "At Risk", "New Customers") based on whether the R/F/M values are high or low relative to other clusters.

11. **Saving Results (Optional step)**
    - **What**: Saves the rfm_df (containing CustomerID, Recency, Frequency, MonetaryValue, and Cluster) to a new CSV file.
    - **Why**: To persist the segmentation results for further analysis or use in marketing tools, reporting, etc.
    - **How**: Uses rfm_df.to_csv(). reset_index() is used before saving to turn the CustomerID (which was the index of rfm_df) back into a regular column in the output CSV.

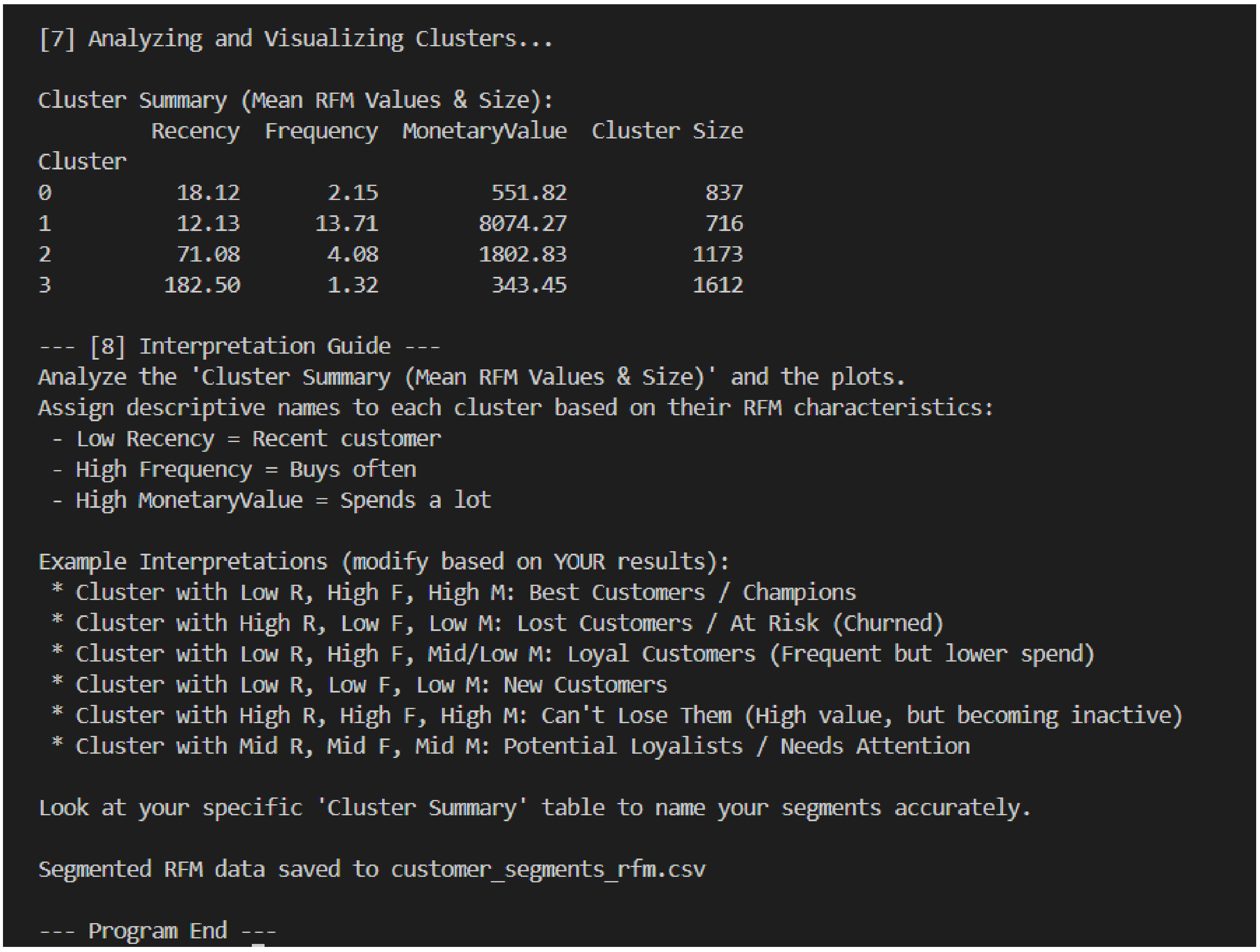
OUTPUT



```
PS D:\DEVELOPER> python -u "d:\DEVELOPER\PYTHON\Code\SBMP AIML\main.py"
d:\DEVELOPER\PYTHON\Code\SBMP AIML\main.py:24: SyntaxWarning: invalid escape sequence '\D'
  file_path = "D:\DEVELOPER\PYTHON\Code\SBMP AIML\Retail_dataset.csv"
--- Customer Segmentation Program (RFM Analysis) ---

[1] Loading Data from D:\DEVELOPER\PYTHON\Code\SBMP AIML\Retail_dataset.csv...
Data loaded successfully.

Original Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB

Sample Data Head:
  InvoiceNo StockCode                          Description  Quantity         InvoiceDate  UnitPrice  CustomerID         Country
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1    536365     71053                  WHITE METAL LANTERN         6 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
[2] Cleaning and Preparing Data...

Missing values before cleaning:
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64

Removed 135080 rows with missing CustomerID.
Removed 8945 rows corresponding to cancellations, negative/zero quantity or price.

Data after initial cleaning:
  InvoiceNo StockCode                          Description  Quantity         InvoiceDate  UnitPrice  CustomerID         Country  TotalPrice
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6 2010-12-01 08:26:00       2.55       17850  United Kingdom       15.30
1    536365     71053                  WHITE METAL LANTERN         6 2010-12-01 08:26:00       3.39       17850  United Kingdom       20.34
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8 2010-12-01 08:26:00       2.75       17850  United Kingdom       22.00
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6 2010-12-01 08:26:00       3.39       17850  United Kingdom       20.34
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6 2010-12-01 08:26:00       3.39       17850  United Kingdom       20.34
<class 'pandas.core.frame.DataFrame'>
Index: 397884 entries, 0 to 541908
Data columns (total 9 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    397884 non-null  object
 1   StockCode    397884 non-null  object
 2   Description  397884 non-null  object
 3   Quantity     397884 non-null  int64
 4   InvoiceDate  397884 non-null  datetime64[ns]
 5   UnitPrice    397884 non-null  float64
 6   CustomerID   397884 non-null  int64
 7   Country      397884 non-null  object
 8   TotalPrice   397884 non-null  float64
```



```
dtypes: datetime64[ns](1), float64(2), int64(2), object(4)
memory usage: 30.4+ MB

[3] Calculating RFM Features...
Snapshot date for Recency calculation: 2011-12-10

RFM Dataframe Head:
            Recency  Frequency  MonetaryValue
CustomerID
12346           326          1       77183.60
12347             2          7        4310.00
12348            75          4        1797.24
12349            19          1        1757.55
12350           310          1         334.40

Descriptive Statistics for RFM:
           Recency    Frequency  MonetaryValue
count  4338.000000  4338.000000    4338.000000
mean     92.536422     4.272015    2054.266460
std     100.014169     7.697998    8989.230441
min       1.000000     1.000000       3.750000
25%      18.000000     1.000000     307.415000
50%      51.000000     2.000000     674.485000
75%     142.000000     5.000000    1661.740000
max     374.000000   209.000000  280206.020000

Applying log transformation to handle skewness in F and M (and potentially R)...

RFM Dataframe Head (Log Transformed):
             Recency  Frequency  MonetaryValue
CustomerID
12346       5.789960   0.693147      11.253955
12347       1.098612   2.079442       8.368925
12348       4.330733   1.609438       7.494564
12349       2.995732   0.693147       7.472245
12350       5.739793   0.693147       5.815324

[4] Preprocessing RFM Data (Scaling)...
RFM data scaled using StandardScaler.

[5] Determining Optimal Number of Clusters (K)...
 K=2, WCSS=6481.23, Silhouette Score=0.433
 K=3, WCSS=4867.85, Silhouette Score=0.337
 K=4, WCSS=3938.51, Silhouette Score=0.337
 K=5, WCSS=3295.98, Silhouette Score=0.316
 K=6, WCSS=2855.01, Silhouette Score=0.313
 K=7, WCSS=2548.91, Silhouette Score=0.310
 K=8, WCSS=2336.78, Silhouette Score=0.301
 K=9, WCSS=2155.65, Silhouette Score=0.282
 K=10, WCSS=1999.90, Silhouette Score=0.279

Based on Elbow Method and Silhouette Score analysis, choosing K = 4

[6] Applying K-Means with K = 4...
Cluster labels assigned to RFM data.

RFM Data Head with Cluster Labels:
            Recency  Frequency  MonetaryValue  Cluster
CustomerID
12346           326          1       77183.60        2
12347             2          7        4310.00        1
12348            75          4        1797.24        2
12349            19          1        1757.55        0
12350           310          1         334.40        3
```



```
[7] Analyzing and Visualizing Clusters...

Cluster Summary (Mean RFM Values & Size):
         Recency  Frequency  MonetaryValue  Cluster Size
Cluster
0          18.12       2.15         551.82           837
1          12.13      13.71        8074.27           716
2          71.08       4.08        1802.83          1173
3         182.50       1.32         343.45          1612

--- [8] Interpretation Guide ---
Analyze the 'Cluster Summary (Mean RFM Values & Size)' and the plots.
Assign descriptive names to each cluster based on their RFM characteristics:
 - Low Recency = Recent customer
 - High Frequency = Buys often
 - High MonetaryValue = Spends a lot

Example Interpretations (modify based on YOUR results):
 * Cluster with Low R, High F, High M: Best Customers / Champions
 * Cluster with High R, Low F, Low M: Lost Customers / At Risk (Churned)
 * Cluster with Low R, High F, Mid/Low M: Loyal Customers (Frequent but lower spend)
 * Cluster with Low R, Low F, Low M: New Customers
 * Cluster with High R, High F, High M: Can't Lose Them (High value, but becoming inactive)
 * Cluster with Mid R, Mid F, Mid M: Potential Loyalists / Needs Attention

Look at your specific 'Cluster Summary' table to name your segments accurately.

Segmented RFM data saved to customer_segments_rfm.csv

--- Program End ---
```

3.1 Visualization

Figure 1:

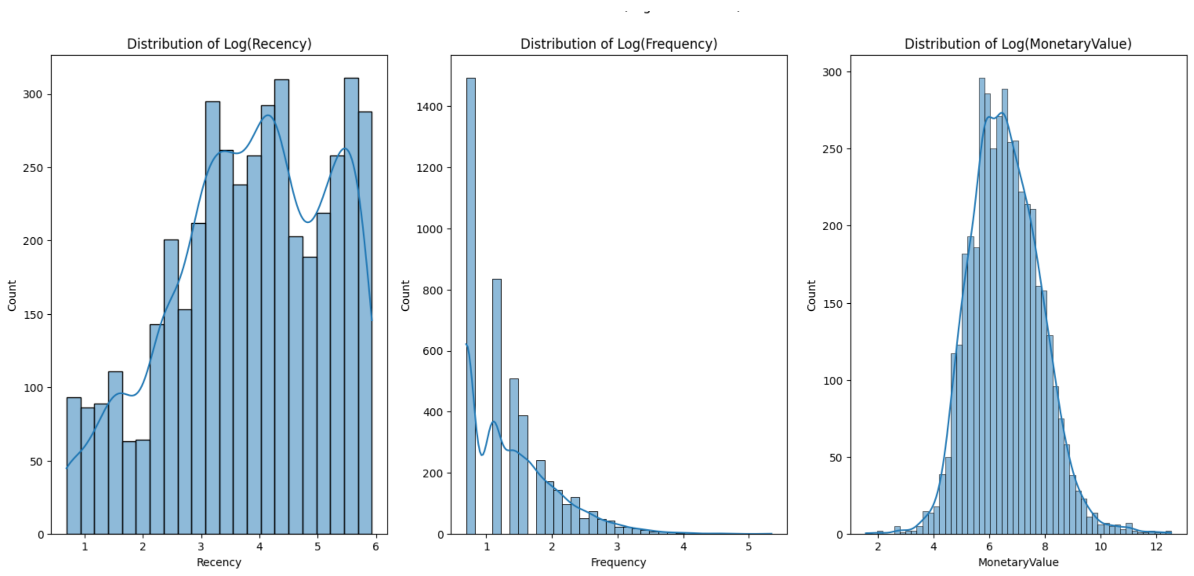


Figure 2:

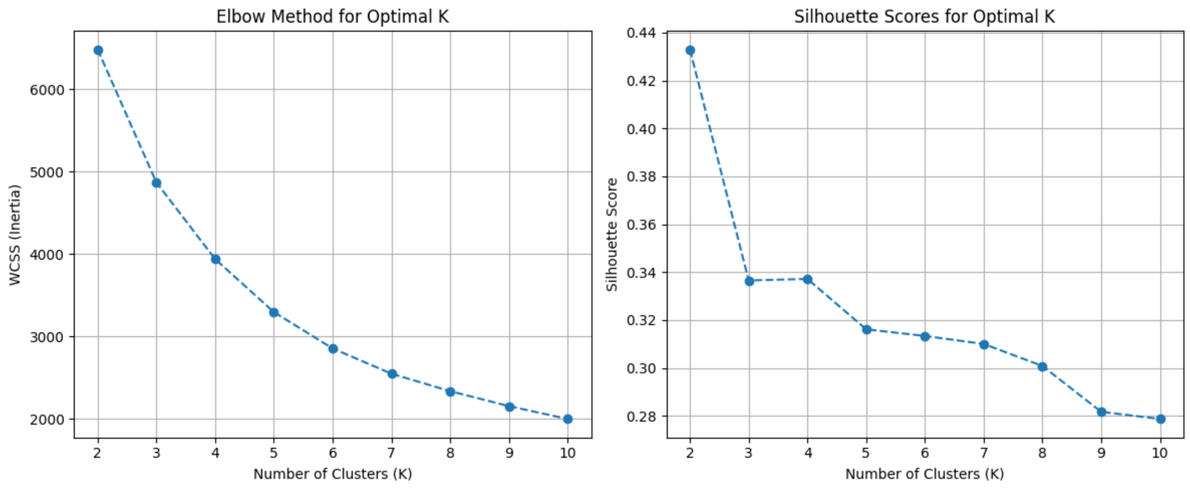


Figure 3:

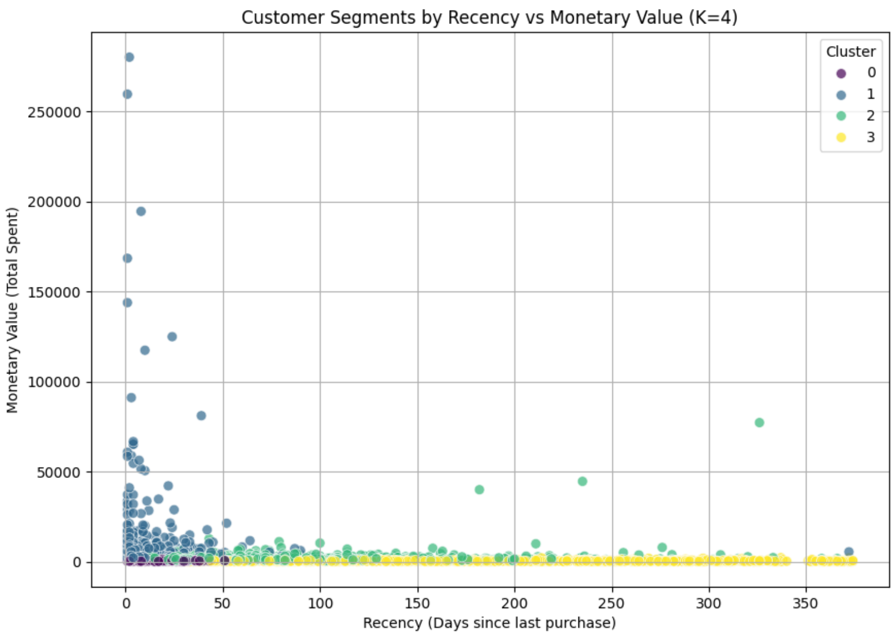


Figure 4:

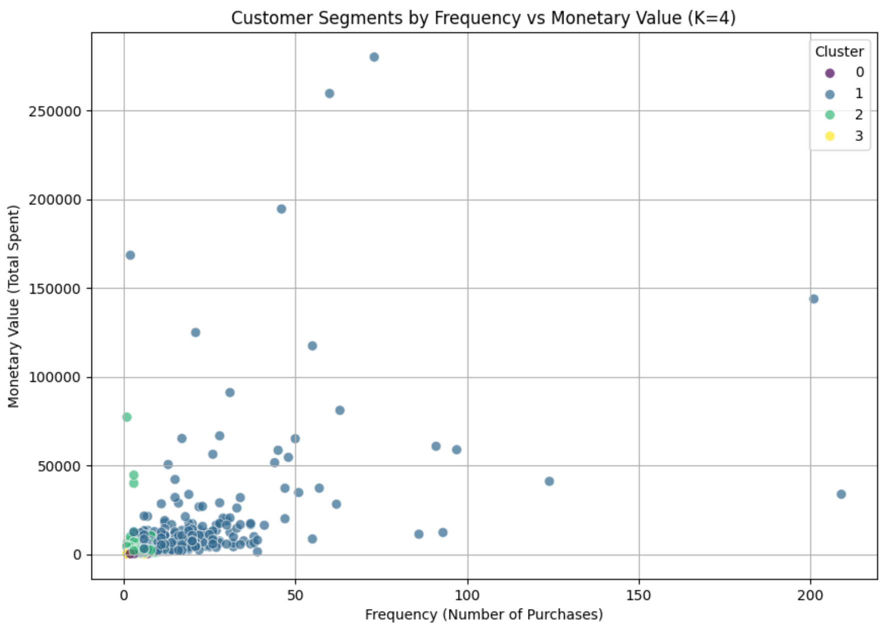


Figure 5:

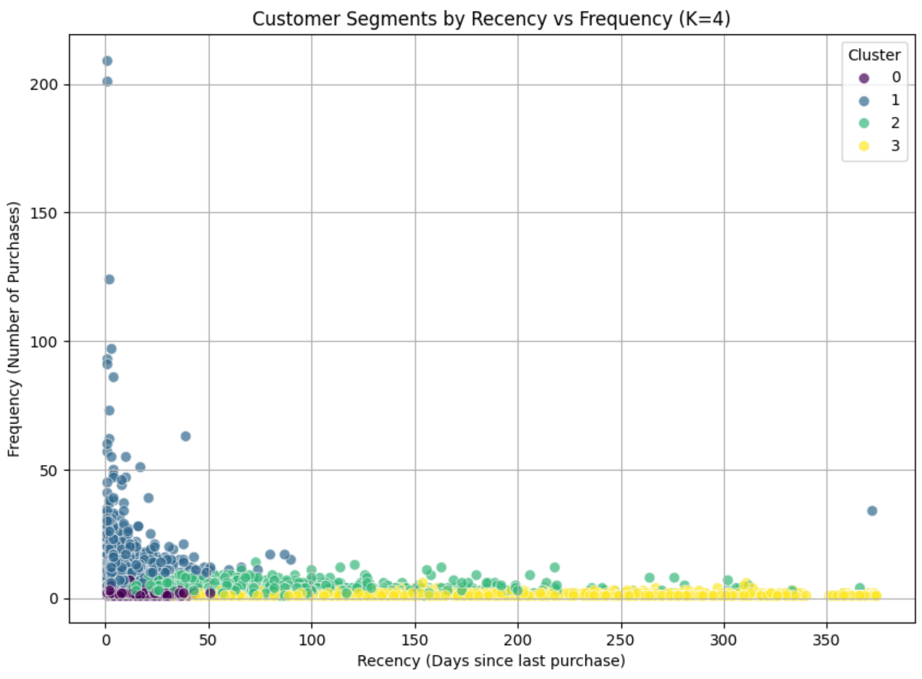


Figure 6:

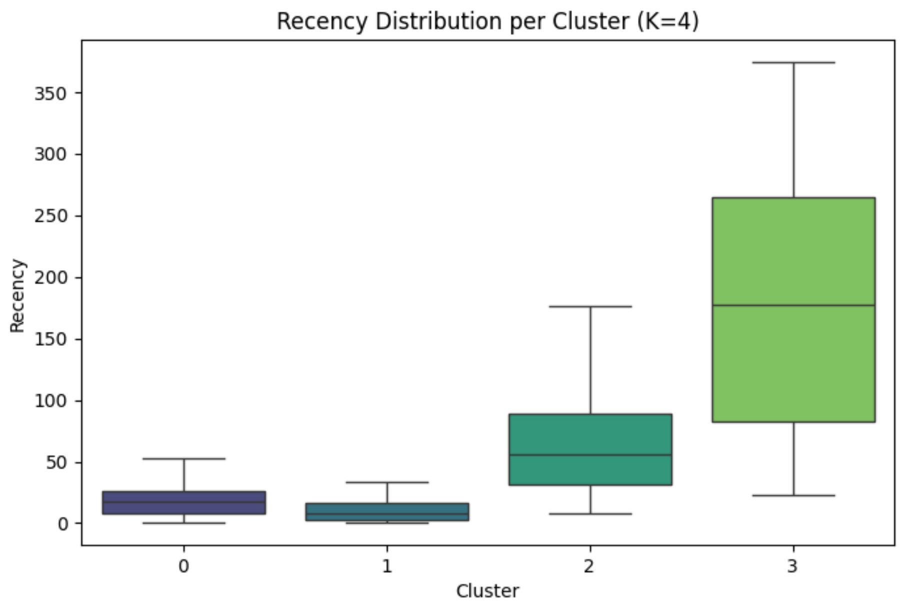


Figure 7:

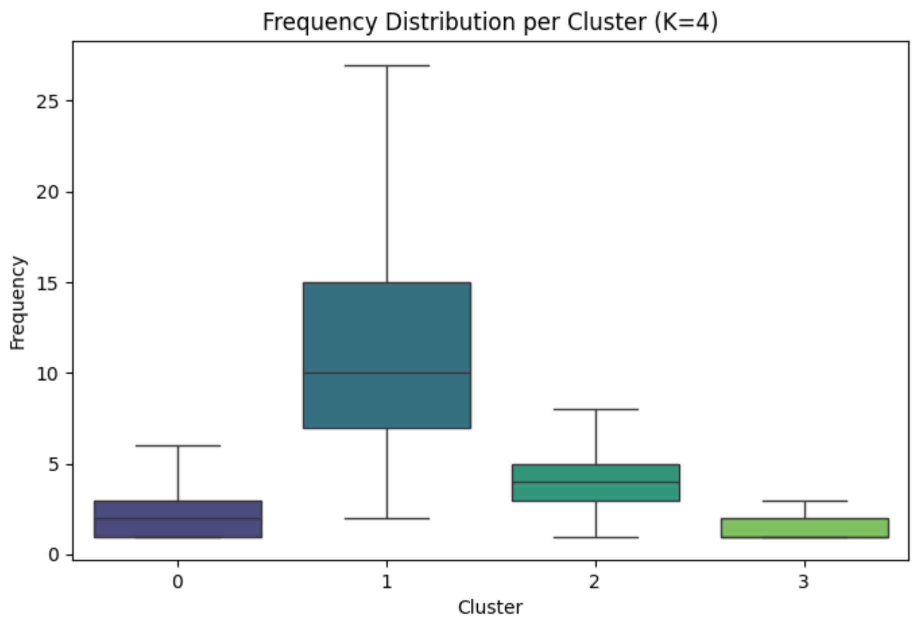
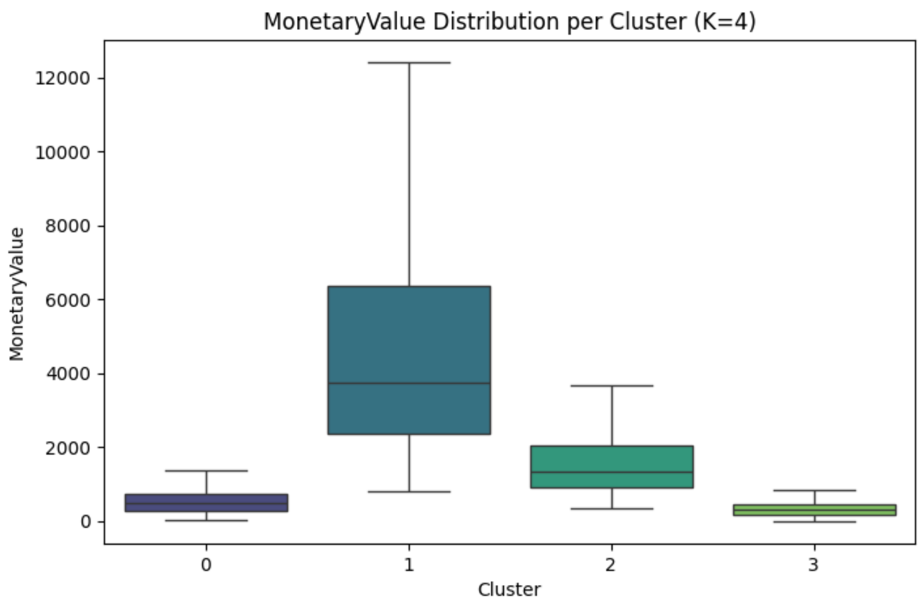


Figure 8:



CONCLUSION

This report has demonstrated the successful implementation of an AIML-driven solution for customer segmentation. The developed Python program leveraged the K-Means clustering algorithm, a fundamental unsupervised learning technique, applied to features derived from the established Recency, Frequency, Monetary (RFM) model. The simulation effectively replicated the core principles of a digital clock: the continuous updating of time units (seconds, minutes, and hours) and their representation through the distinct patterns of a 7-segment display.

This project showcases how machine learning can automate and enhance the process of understanding customer heterogeneity, translating complex data patterns into clear segments for targeted business actions and improved personalization.

I got deep knowledge in the field of Python programming especially focusing on concepts such as pandas, numpy, matplotlib, seaborn, sklearn, KMeans, StandardScaler etc.

Hence, this Project enhanced my Logical Thinking and fundamentals of this particular subject and enhanced my Pyhton. Skills.

REFRENCES

1. Python Documentation
2. https://www.shopify.com/blog/what-is-customer-segmentation
3. https://www.techtarget.com/searchdatamanagement/definition/RFM-analysis
4. https://www.ibm.com/think/topics/k-means-clustering