CUSTOMER SEGMENTATION ANALYIS (RFM ANALYSIS)

IE6400 – FOUNDATIONS OF DATA ANALYTICS (FALL 2024)

PROJECT REPORT - 2

GROUP NUMBER 14

NIRMALKUMAR THIRUPALLIKRISHNAN KESAVAN (002334868)

MOHAMMED IBRAHIM LNU (002319064)

Introduction

In today's competitive eCommerce landscape, understanding customer behavior is essential for enhancing customer experience, improving retention, and driving sales. Businesses rely on data-driven strategies to identify customer segments and tailor their marketing efforts effectively. One such widely used technique is RFM (Recency, Frequency, Monetary) analysis, which evaluates customer purchasing patterns based on three key metrics:

Recency (R): How recently a customer has made a purchase.

Frequency (F): How often a customer has made purchases over a specific period.

Monetary Value (M): The total monetary value of a customer's transactions.

This project leverages RFM analysis to segment customers from an eCommerce dataset containing transactional data from 2010 to 2011 for a UK-based online retail company. The dataset includes details such as invoice numbers, stock codes, purchase dates, quantities, and customer IDs.

The objective of this project is to identify distinct customer groups by performing RFM analysis and applying KMeans clustering. By understanding the unique characteristics of each customer segment, businesses can craft targeted marketing strategies and personalized engagement initiatives to maximize customer lifetime value and foster long-term loyalty.

Through this project, we aim to highlight the power of RFM-based segmentation in extracting actionable insights and its pivotal role in modern customer relationship management.

Task 1. Data Preprocessing:

During the data preparation stage, we loaded the dataset data.csv into a pandas DataFrame using the Python programming language. To ensure compatibility with various character encodings, we specified the latin-1 encoding while reading the file. We then checked for missing values in the dataset to identify potential data cleaning requirements. The total number of rows was displayed to understand the dataset size, and the data types of each column were inspected for consistency and correctness. This step served as the foundation for subsequent data preprocessing and analysis.

```python
import pandas as pd

# Load dataset with specified encoding
data = pd.read_csv('data.csv', encoding='latin-1')

# Check missing values
missing_values = data.isnull().sum()
print("Missing Values:")
print(missing_values)

# Display total rows
total_rows = data.shape[0]
print(f"Total number of rows: {total_rows}")

# Display data types
print("Data Types:")
print(data.dtypes)
```

Results:

```
Missing Values:
InvoiceNo          0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID    135080
Country            0
dtype: int64
```

```
Data Types:
InvoiceNo      object
StockCode      object
Description    object
Quantity        int64
InvoiceDate    object
UnitPrice     float64
CustomerID    float64
Country        object
dtype: object
```

```
Total number of rows: 541909
```

The output shows that the CustomerID column has a high number of missing values (135,080), making it unsuitable for analysis, so it will be removed. The Description column has 1,454 missing values, which will also be addressed during preprocessing, while other columns have no missing values and are ready for analysis.

```python
# Drop rows with missing 'CustomerID'
data = data.dropna(subset=['CustomerID'])

# Check missing values after dropping
missing_values = data.isnull().sum()
print("Missing Values after dropping rows with missing CustomerID:")
print(missing_values)
```

```
Missing Values after dropping rows with missing CustomerID:
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

Task 2. RFM Calculation:

RFM (Recency, Frequency, Monetary) analysis is a customer segmentation technique used in marketing to identify a company's most valuable customers. It categorizes customers based on three key factors:
Recency: How recently a customer made a purchase.
Frequency: How often a customer makes purchases.
Monetary: How much money a customer spends.

This technique helps businesses target specific customer segments with personalized marketing strategies to improve engagement, retention, and revenue.

```
# Convert 'InvoiceDate' to datetime
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

# Calculate Recency
current_date = data['InvoiceDate'].max()
recency = current_date - data.groupby('CustomerID')['InvoiceDate'].max()
recency = recency.dt.days  # Extract the number of days

# Calculate Frequency
frequency = data.groupby('CustomerID')['InvoiceNo'].nunique()

# Calculate Monetary Value
monetary = data.groupby('CustomerID').agg({'Quantity': 'sum', 'UnitPrice': 'sum'})
monetary['Monetary'] = monetary['Quantity'] * monetary['UnitPrice']

# Combine into RFM DataFrame
rfm_data = pd.DataFrame({
    'Recency': recency,
    'Frequency': frequency,
    'Monetary': monetary['Monetary']
})

print(rfm_data.head())
```

This code performs RFM analysis by first converting the InvoiceDate column to a datetime format. It calculates Recency as the number of days since the customer's last purchase, Frequency as the count of unique invoices, and Monetary as the total revenue generated by each customer (calculated by multiplying quantity and unit price). These values are combined into a new rfm_data DataFrame containing Recency, Frequency, and Monetary metrics for each customer. This DataFrame serves as the basis for customer segmentation.

Results:

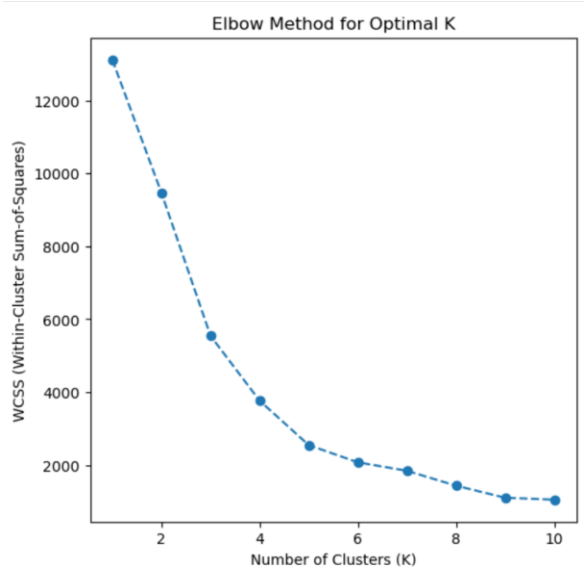
CustomerID	Recency	Frequency	Monetary
12346.0	325	2	0.00
12347.0	1	7	1182814.18
12348.0	74	4	418360.11
12349.0	18	1	381818.10
12350.0	309	1	12864.10

Task 3. RFM Segmentation, Customer Segmentation and Visualization

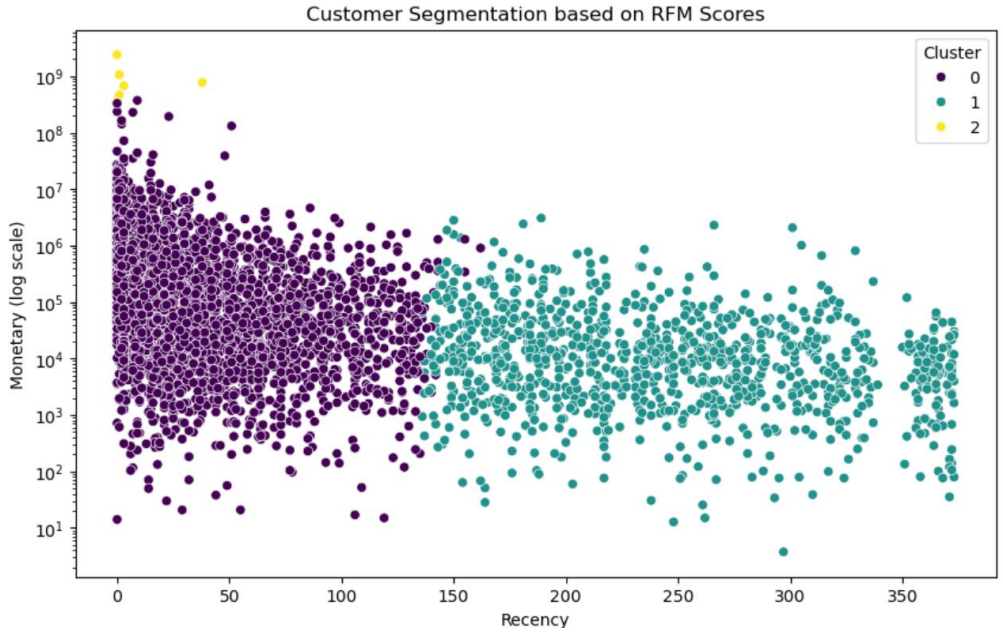
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

rfm_for_clustering = rfm_data[['Recency', 'Frequency', 'Monetary']]

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm_for_clustering)

wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(rfm_scaled)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS (Within-Cluster-Sum-of-Squares)')
plt.show()

optimal_k = 3

kmeans = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42)
rfm_data['Cluster'] = kmeans.fit_predict(rfm_scaled)

plt.figure(figsize=(10, 8))
sns.scatterplot(x='Recency', y='Monetary', hue='Cluster', data=rfm_data, palette='viridis', legend='full')

plt.yscale('log')

plt.title('Customer Segmentation based on RFM Scores')
plt.xlabel('Recency')
plt.ylabel('Monetary (log scale)')
plt.show()
```

Here we used the Elbow Method to determine the ideal number of clusters, which turns out to be three, then used KMeans clustering to the RFM analysis data, scaling the features. After that, the data was clustered, given cluster names, and the outcomes were displayed in a scatter plot for the "Monetary" value using logarithmic scale.

Results:



To find the ideal number of clusters for KMeans clustering, an elbow method plot is utilized. The graph displays the number of clusters (K) on the x-axis and the within-cluster sum of squares (WCSS) on the y-axis. The plot shows a steep drop in WCSS as K rises from 1 to 3, after which the reduction slows. Since the elbow of the curve is at 3, this suggests that the ideal K is probably about 3.



This scatterplot visualizes customer segmentation based on RFM (Recency, Frequency, Monetary) scores, with customers grouped into three clusters. The x-axis represents Recency (days since last purchase), and the y-axis (log scale) represents Monetary (total spending). Each color corresponds to a cluster, indicating distinct customer groups. For example, Cluster 0 (purple) may include less valuable customers with low spending and longer recency, while Cluster 2 (yellow) represents highly valuable customers with high spending and recent activity. This segmentation helps identify and target different customer groups effectively.

Task 4. Segment Profiling:

```
# Segment profiling
segment_profiles = rfm_data.groupby('Cluster').agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'Monetary': 'mean'
}).reset_index()

segment_profiles['Number of Customers'] = rfm_data['Cluster'].value_counts().sort_index().values

segment_profiles = segment_profiles.rename(columns={
    'Recency': 'Average Recency',
    'Frequency': 'Average Frequency',
    'Monetary': 'Average Monetary'
})

print(segment_profiles)
```

This code calculates segment profiles for customer clusters. It groups the rfm_data by Cluster and computes the mean values of Recency, Frequency, and Monetary for each cluster. The number of customers in each cluster is added as a new column. Column names are then renamed to provide more descriptive labels like "Average Recency," "Average Frequency," and "Average Monetary." Finally, the segment profiles are printed, summarizing customer behavior for each cluster.

Results:

	Cluster	Average Recency	Average Frequency	Average Monetary	\
0	0	38.815838	5.927563	1.396348e+06	
1	1	245.083935	1.850181	5.193095e+04	
2	2	7.166667	138.000000	9.569100e+08	

	Number of Customers
0	3258
1	1108
2	6

The output summarizes the profiles of three customer clusters based on RFM analysis. Cluster 0 has 3,258 customers with moderate recency (39 days), low frequency (~6 purchases), and average monetary value (~1.4 million). Cluster 1 includes 1,108 customers with high recency (245 days), low frequency (~2 purchases), and low monetary value (~51,930). Cluster 2 represents only 6 high-value customers with very low recency (~7 days), very high frequency (138 purchases), and extremely high monetary value (~95.7 million). These profiles highlight the behavioral differences among the clusters, enabling targeted strategies.

Task 5. Marketing Recommendations:

Actionable marketing recommendations for each segment:

Cluster 0 (Moderate Value Customers):

1. **Characteristics:** Moderate recency (~39 days), moderate frequency (~6 purchases), and moderate monetary value (~1.4M). Largest segment with 3,258 customers.
2. **Recommendations**: Launch loyalty programs to encourage repeat purchases and build brand loyalty. Use personalized marketing campaigns highlighting discounts and product recommendations based on past purchases. Promote time-sensitive offers to create urgency and maintain engagement.

Cluster 1 (Low-Value Customers):

1. **Characteristics:** High recency (~245 days), low frequency (~2 purchases), and low monetary value (~51,930). Includes 1,108 customers.
2. **Recommendations:** Re-engage these dormant customers with targeted win-back campaigns offering discounts, free shipping, or exclusive deals. Send email reminders or push notifications

showcasing new products and promotions. Introduce low-risk subscription models or free trials to rekindle interest.

Cluster 2 (High-Value Customers):

1. **Characteristics:** Very low recency (~7 days), very high frequency (~138 purchases), and extremely high monetary value (~95.7M). Smallest segment with 6 customers.
2. **Recommendations:** Focus on retaining these VIP customers with premium loyalty programs, exclusive perks, and early access to products. Offer personalized thank-you messages or gifts for their loyalty. Engage them in referral programs to attract similar high-value customers.

General Recommendations:

1. **Cross-Selling:** Identify complementary products and promote bundle offers for all clusters to maximize cart value.
2. **Personalized Communication:** Use customer behavior data to send tailored messages, such as purchase recommendations or restock alerts.
3. **Retention Strategies:** Focus on retention campaigns for Cluster 0 and Cluster 2 to ensure consistent engagement and long-term loyalty.
4. **Feedback Collection:** Conduct surveys or interviews with Cluster 2 customers to understand their preferences and improve offerings. Use insights for other clusters.
5. **Social Media Engagement:** Utilize social media campaigns to engage Clusters 0 and 1 with promotions, while showcasing VIP benefits for Cluster 2.

These strategies aim to maximize customer lifetime value and optimize engagement for all segments.

Questions:

1. Data Overview:

a. What is the size of the dataset in terms of the number of rows and columns?

```
# Display dataset size
print(f"Dataset Size: {data.shape}")
```

Results:

```
Dataset Size: (406829, 8)
```

b. Can you provide a brief description of each column in the dataset?

```
# Dataset info
print("\nColumn Descriptions:")
print(data.info())
```

```
# Dataset summary statistics
print("\nSummary Statistics for Numerical Columns:")
print(data.describe())
```

This code first prints concise information about the dataset's structure using `data.info()`. Then, it displays summary statistics for numerical columns with `data.describe()`.

Results:

```
Summary Statistics for Numerical Columns:
            Quantity                    InvoiceDate      UnitPrice  \
count  406829.000000                         406829  406829.000000
mean       12.061303  2011-07-10 16:30:57.879207424       3.460471
min    -80995.000000            2010-12-01 08:26:00       0.000000
25%         2.000000            2011-04-06 15:02:00       1.250000
50%         5.000000            2011-07-31 11:48:00       1.950000
75%        12.000000            2011-10-20 13:06:00       3.750000
max     80995.000000            2011-12-09 12:50:00   38970.000000
std       248.693370                            NaN      69.315162

          CustomerID
count  406829.000000
mean    15287.690570
min     12346.000000
25%     13953.000000
50%     15152.000000
75%     16791.000000
max     18287.000000
std      1713.600303
```

```
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    406829 non-null  object
 1   StockCode    406829 non-null  object
 2   Description  406829 non-null  object
 3   Quantity     406829 non-null  int64
 4   InvoiceDate  406829 non-null  datetime64[ns]
 5   UnitPrice    406829 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      406829 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 27.9+ MB
None
```

c. What is the period covered by this dataset?

```
# Time period covered
print("\nTime Period Covered:")
print(f"Minimum Invoice Date: {data['InvoiceDate'].min()}")
print(f"Maximum Invoice Date: {data['InvoiceDate'].max()}")
```

This code offers information about the time frame that the dataset covers. It displays the range of transactions by printing the minimum and maximum invoice dates.

Results:

```
Time Period Covered:
Minimum Invoice Date: 2010-12-01 08:26:00
Maximum Invoice Date: 2011-12-09 12:50:00
```

2. Customer Analysis:

a. How many unique customers are there in the dataset?

```
import matplotlib.pyplot as plt
import seaborn as sns

# Unique customers
unique_customers = data['CustomerID'].nunique()
print(f"Number of Unique Customers: {unique_customers}")
```

This code calculates the total number of unique customers in the dataset using the nunique() function on the CustomerID column. It then prints the result in a formatted string to display the count of unique customers.

Results:

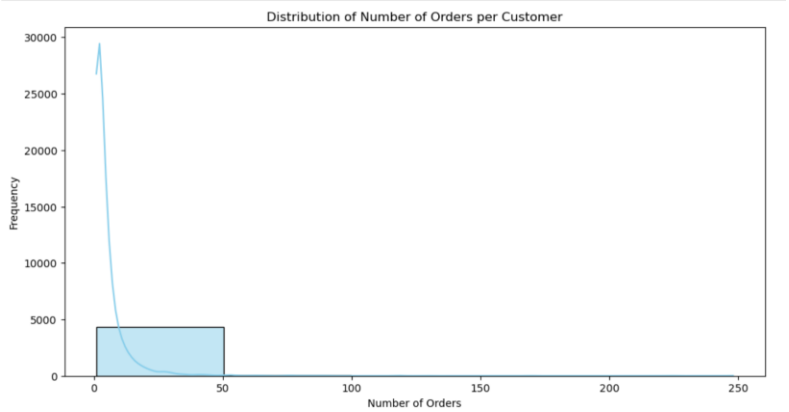
```
Number of Unique Customers: 4372
```

b. What is the distribution of the number of orders per customer?

```
# Distribution of number of orders per customer
orders_per_customer = data.groupby('CustomerID')['InvoiceNo'].nunique()

plt.figure(figsize=(12, 6))
sns.histplot(orders_per_customer, bins=5, kde=True, color='skyblue')
plt.title("Distribution of Number of Orders per Customer")
plt.xlabel("Number of Orders")
plt.ylabel("Frequency")
plt.show()
```

This code calculates the number of unique orders per customer by grouping the data by CustomerID and counting unique InvoiceNo values. It then creates a histogram using Seaborn to visualize the distribution of order counts per customer, with a KDE (Kernel Density Estimate) overlay for smoothness, labeled axes, and a title.

Results:



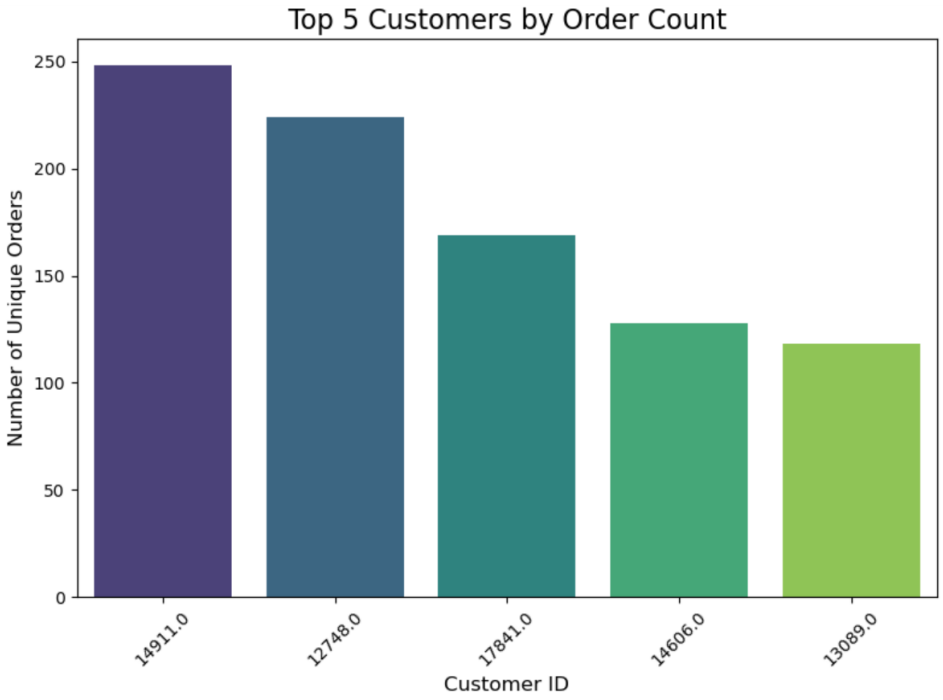
The histogram shows the distribution of the number of orders per customer. Most customers place a small number of orders, as indicated by the steep drop-off near the start of the chart. The frequency significantly decreases as the number of orders increases, highlighting a long-tail distribution.

c. Can you identify the top 5 customers who have made the most purchases by order count?

```
import matplotlib.pyplot as plt
import seaborn as sns

# Identify top 5 customers by order count
top_customers_by_order_count = data.groupby('CustomerID')['InvoiceNo'].nunique().sort_values(ascending=False).head(5)
print("Top 5 Customers and Order Count:")
print(top_customers_by_order_count)

# Visualization
plt.figure(figsize=(10, 6))
sns.barplot(x=top_customers_by_order_count.index.astype(str), y=top_customers_by_order_count.values, palette='viridis')
plt.title("Top 5 Customers by Order Count", fontsize=16)
plt.xlabel("Customer ID", fontsize=12)
plt.ylabel("Number of Unique Orders", fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Results:

```
Top 5 Customers and Order Count:
CustomerID
14911.0    248
12748.0    224
17841.0    169
14606.0    128
13089.0    118
Name: InvoiceNo, dtype: int64
```



3. Product Analysis:

a. What are the top 10 most frequently purchased products?

```
top_products_by_frequency = data['StockCode'].value_counts().head(10)
print("Top 10 Most Frequently Purchased Products:")
print(top_products_by_frequency)
```

Using value_counts(), this code counts the instances of each distinct StockCode in the dataset to determine the top ten most popular goods. The most popular goods and their purchase frequencies are displayed when the result is saved in top_products_by_frequency and printed.

Results:

```
Top 10 Most Frequently Purchased Products
StockCode
85123A    2077
22423     1905
85099B    1662
84879     1418
47566     1416
20725     1359
22720     1232
POST      1196
20727     1126
22197     1118
Name: count, dtype: int64
```

b. What is the average price of products in the dataset?

```
average_price = data['UnitPrice'].mean()
print("Average Price of Products:", average_price)
```

By utilizing the.mean() function to take the mean of the UnitPrice column, this code determines the average unit price of the products in the dataset. The result is then displayed by printing the average price that was determined.

Results:

```
Average Price of Products: 3.460471018536043
```

c. Can you find out which product category generates the highest?

```
data['Revenue'] = data['Quantity'] * data['UnitPrice']
top_revenue_category = data.groupby('Description')['Revenue'].sum().idxmax()
print("Product Category Generating the Highest Revenue:", top_revenue_category)
```

The Quantity and UnitPrice columns are multiplied in this code to determine the total income for each product, which is then stored in a new income column. The product category with the highest total revenue is then determined by grouping data according to the Description column, adding up the revenue values for each group, and using.idxmax() to determine which category has the most revenue. Lastly, it prints the product category that generates the most income.

Results:

```
Product Category Generating the Highest Revenue: REGENCY CAKESTAND 3 TIER
```

4. Time Analysis:

a. Is there a specific day of the week or time of day when most orders are placed?

```
import matplotlib.pyplot as plt
import seaborn as sns

data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
data['DayOfWeek'] = data['InvoiceDate'].dt.day_name()
data['HourOfDay'] = data['InvoiceDate'].dt.hour

# Orders by Day of the Week
plt.figure(figsize=(12, 6))
sns.countplot(x='DayOfWeek', data=data, palette='viridis',
              order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.title("Distribution of Orders Over Days of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Number of Orders")
plt.show()

# Orders by Hour of the Day
plt.figure(figsize=(12, 6))
sns.countplot(x='HourOfDay', data=data, palette='viridis')
plt.title("Distribution of Orders Over Hours of the Day")
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Orders")
plt.show()
```

- **Handling Date and Time:**

Convert the 'InvoiceDate' column to datetime format.
Create new columns for the day of the week ('DayOfWeek') and the hour of the day ('HourOfDay').

- **Plotting the Distribution of Orders over Days of the Week:**

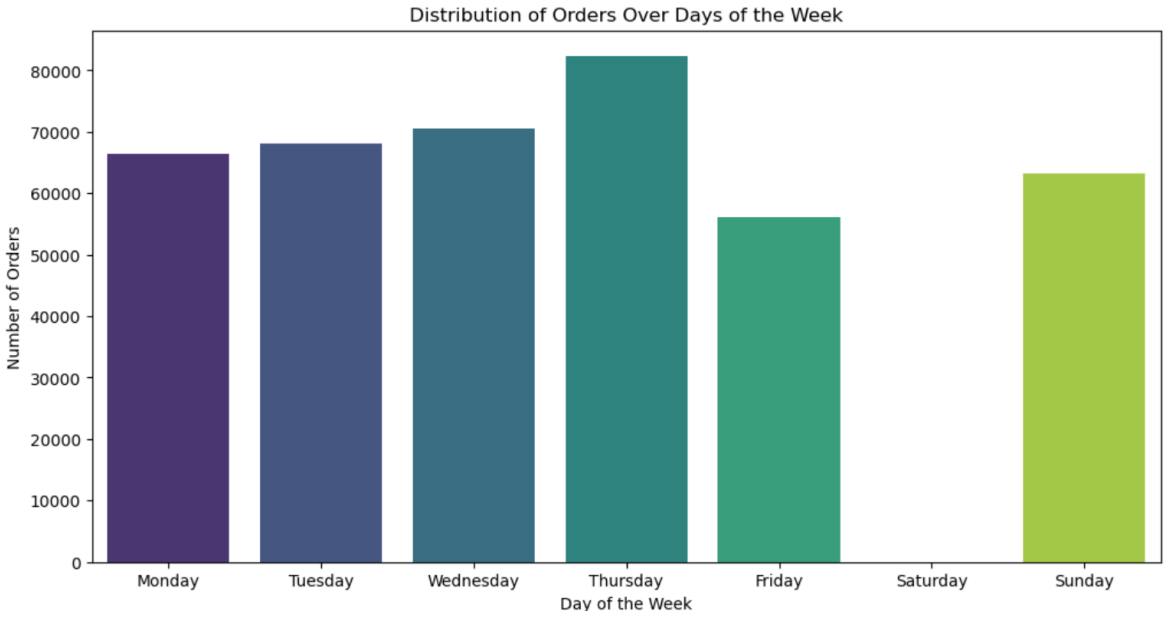
Creates a bar plot with Seaborn (sns.countplot) to show the number of orders for each day of the week. The x-axis represents the days of the week, and the y-axis represents the number of orders.

- **Plotting the Distribution of Orders over Hours of the Day:**

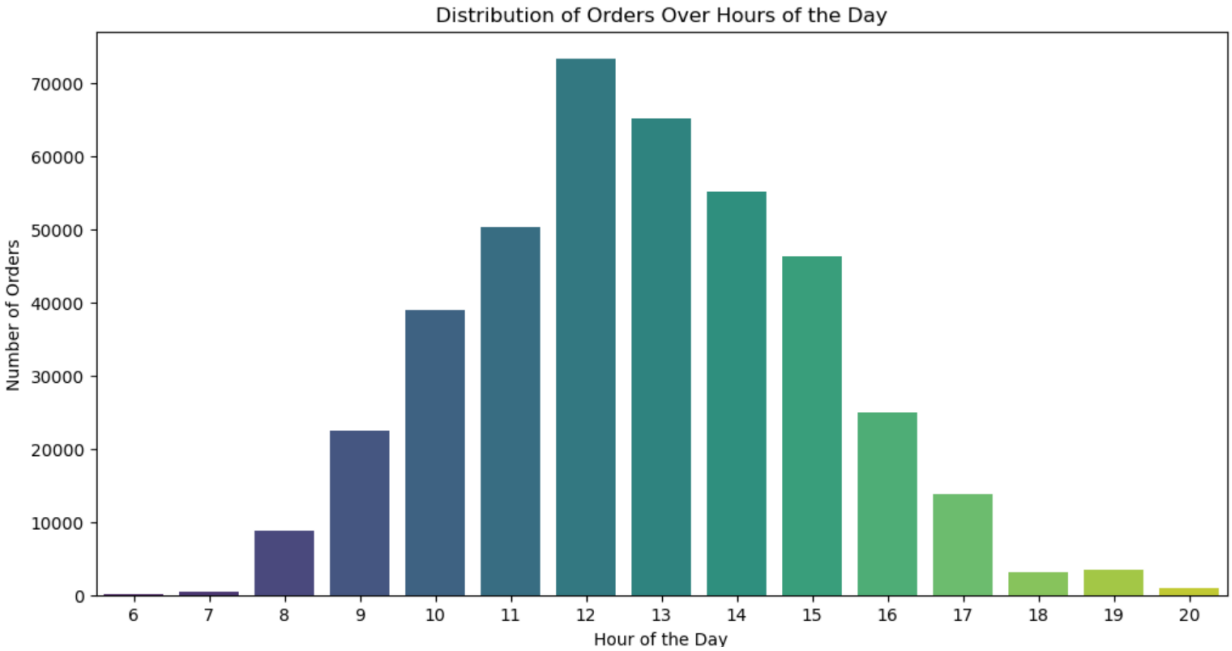
Creates another bar plot to visualize the number of orders during different hours of the day.
The x-axis represents hours, and the y-axis represents the number of orders.

Results:

The bar chart illustrates the distribution of orders across different days of the week. It shows that Thursday experiences the highest number of orders, suggesting peak customer activity on this day. Monday, Tuesday, and Wednesday also have high order volumes, indicating consistent engagement early in the week. In contrast, Friday and Sunday observe relatively lower order counts, potentially signaling a drop-in customer activity. This pattern highlights Thursday as an optimal day for promotional



campaigns or special discounts to maximize sales. Businesses could also target Monday through Wednesday for steady engagement initiatives.



The bar chart shows the distribution of orders by the hour, revealing customer activity trends throughout the day. Most orders occur between 11:00 AM and 2:00 PM, with a clear peak at noon. This suggests that customers are most active during mid-day hours. After 3:00 PM, there is a noticeable decline in order activity, with minimal engagement during early morning and late evening hours. These insights suggest businesses should focus marketing efforts, flash sales, or customer outreach during the mid-day peak hours to maximize customer response and sales.

b. What is the average order processing time?

```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
data['OrderProcessingTime'] = data.groupby('InvoiceNo')['InvoiceDate'].transform(lambda x: x.max() - x.min())
average_processing_time = data['OrderProcessingTime'].mean()
print("Average Order Processing Time:", average_processing_time)
```

This code converts the InvoiceDate field into a datetime format in order to get the average order processing time. It computes the difference between the maximum and minimum InvoiceDate for each invoice, groups the data by InvoiceNo, and stores the result in a new column named OrderProcessingTime. The average processing time for each order is then calculated using the mean() method. To provide insight into operational efficiency, the average order processing time is printed at the end.

Results:

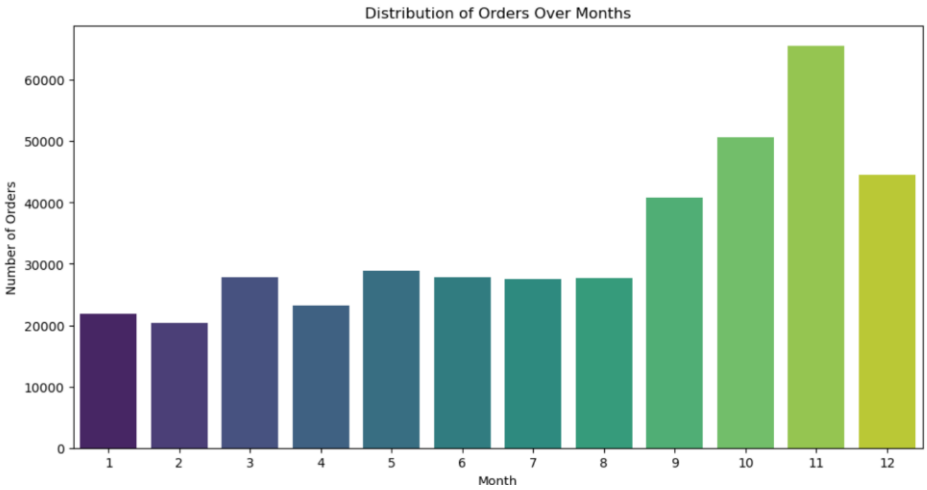
```
Average Order Processing Time: 0 days 00:00:00.238331092
```

c. Are there any seasonal trends in the dataset?

```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
data['Year'] = data['InvoiceDate'].dt.year
data['Month'] = data['InvoiceDate'].dt.month
data['DayOfWeek'] = data['InvoiceDate'].dt.day_name()

# Orders by Month
plt.figure(figsize=(12, 6))
sns.countplot(x='Month', data=data, palette='viridis')
plt.title("Distribution of Orders Over Months")
plt.xlabel("Month")
plt.ylabel("Number of Orders")
plt.show()

# Orders by Year
plt.figure(figsize=(12, 6))
sns.countplot(x='Year', data=data, palette='viridis')
plt.title("Distribution of Orders Over Years")
plt.xlabel("Year")
plt.ylabel("Number of Orders")
plt.show()
```

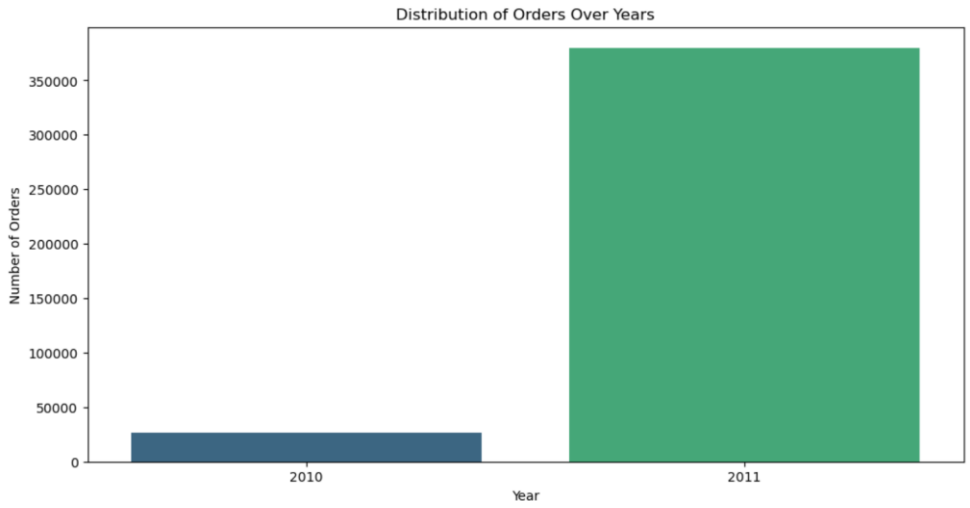
This code creates new columns for Year, Month, and DayOfWeek and converts the InvoiceDate column to datetime format in order to extract time-based attributes from it. Seaborn's countplot is then used to visualize the distribution of orders over months and years. Using a viridis color scheme, the first plot shows the number of orders for each month, with the x-axis representing the months and the y-axis representing the order counts. The second figure highlights patterns over time by displaying the distribution of orders for each year in a similar manner. The dataset's seasonal and annual order patterns can be found with the use of these visualizations.

Results:



Distribution of Orders Over Months:

- November has the highest number of orders, followed by December, indicating peak activity during the holiday season.
- The first quarter (January and February) shows significantly lower order activity, suggesting slower sales periods.
- There is a gradual increase in orders leading up to the last quarter of the year.
- Businesses can leverage this trend to plan targeted marketing campaigns and manage inventory effectively during peak months.



Distribution of Orders Over Years:

- The year 2011 accounts for the majority of orders, significantly outperforming 2010.
- The increase in orders in 2011 may indicate a growth phase or improved business reach.
- The stark difference between the two years emphasizes the importance of understanding growth trends.
- These insights can help guide strategies to maintain or accelerate growth in subsequent years.

5. Geographical Analysis:

a. Can you determine the top 5 countries with the highest number of orders?

```
import numpy as np

# Group data by 'Country'
grouped_data = data.groupby('Country')

# Aggregate stats
country_stats = grouped_data.agg({
    'CustomerID': 'nunique',
    'Revenue': 'mean'
}).rename(columns={'CustomerID': 'Number of Customers', 'Revenue': 'Average Order Value'})

# Correlation
correlation = np.corrcoef(country_stats['Number of Customers'], country_stats['Average Order Value'])[0, 1]
print("Correlation between Number of Customers and Average Order Value:", correlation)

# Top 5 Countries by Number of Orders
top_countries_by_orders = data['Country'].value_counts().head(5)
print("Top 5 Countries with the Highest Number of Orders:")
print(top_countries_by_orders)
```

By classifying client data by nation, this code conducts a geographical analysis and computes important metrics, such as the average order value by nation and the number of unique customers. In order to find such connections, it then calculates the correlation between the average order value and the number of customers. Using the value_counts() method, the code also determines the top 5 nations with the most orders. The calculated correlation and a list of the best-performing nations are included in the results, which provide information on regional sales distribution and consumer behavior.

Results:

```
Correlation between Number of Customers and Average Order Value: -0.10390345388863093
Top 5 Countries with the Highest Number of Orders:
Country
United Kingdom    361878
Germany             9495
France              8491
EIRE                7485
Spain               2533
Name: count, dtype: int64
```

The output shows the correlation between the number of customers and the average order value as -0.1039, indicating a weak negative relationship, suggesting that an increase in the number of customers is slightly associated with a decrease in average order value. The top 5 countries with the highest number of orders are listed, with the United Kingdom significantly leading with 361,878 orders, followed by Germany (9,495), France (8,491), EIRE (7,485), and Spain (2,533). This highlights the dominance of the UK market and potential opportunities for growth in other regions.

b. Is there a correlation between the customer's country and the average order value?

```
# Group data by country
grouped_data = data.groupby('Country')

# Aggregate statistics for each country
country_stats = grouped_data.agg({
    'CustomerID': 'nunique',
    'Revenue': 'mean'
}).rename(columns={
    'CustomerID': 'Number of Customers',
    'Revenue': 'Average Order Value'
})

# Calculate correlation between the number of customers and the average order value
correlation = np.corrcoef(country_stats['Number of Customers'], country_stats['Average Order Value'])[0, 1]
print("Correlation between Number of Customers and Average Order Value:", correlation)

# Print average order value by country
print("Average Order Value by Country:")
print(country_stats)
```

This code examines how each country's average order value and client count relate to one another. First, the "Country" column is used to group the dataset. Each country's mean revenue and total unique customers are then calculated, and the resulting columns are renamed "Number of Customers" and "Average Order Value." The relationship between these two measurements is determined by calculating a correlation coefficient using NumPy's corrcoef function. Lastly, each country's average order value and associated statistics are provided to offer information on local revenue trends and consumer behavior.

Results:

```
Correlation between Number of Customers and Average Order Value: ·
Average Order Value by Country:
                 Number of Customers  Average Order Value
Country
Australia                          9           108.877895
Austria                           11            25.322494
Bahrain                            2            32.258824
Belgium                           25            19.773301
Brazil                             1            35.737500
Canada                             4            24.280662
Channel Islands                    9            26.499063
Cyprus                             8            20.813971
Czech Republic                     1            23.590667
Denmark                            9            48.247147
EIRE                               3            33.438239
European Community                 1            21.176230
Finland                           12            32.124806
France                            87            23.167217
Germany                           95            23.348943
Greece                             4            32.263836
Iceland                            1            23.681319
Israel                             4            27.977000
Italy                             15            21.034259
Japan                              8            98.716816
Lebanon                            1            37.641778
Lithuania                          1            47.458857
Malta                              2            19.728110
Netherlands                        9           120.059696
Norway                            10            32.378877
Poland                             6            21.152903
Portugal                          19            19.635007
RSA                                1            17.281207
Saudi Arabia                       1            13.117000
```

- **Top Average Order Value Countries:** Netherlands leads with the highest average order value of 120.06, followed by Australia (108.88) and Japan (98.72), indicating higher spending per customer in these regions.

- **Low Average Order Value:** The United Kingdom, despite having the highest number of customers (3950), has a low average order value of 18.70, suggesting a high volume of small purchases.

- **Countries with Few Customers:** Countries like RSA (South Africa) and Saudi Arabia have minimal customer representation with lower average order values of 17.28 and 13.12, respectively.

- **Moderate Spenders:** Countries such as Sweden (79.21) and Denmark (48.25) show moderate spending patterns, indicating mid-level engagement.

- **Unspecified Data:** The "Unspecified" category has a notably low average order value (10.93), possibly due to incomplete or ambiguous customer information.

6. Payment Analysis:

a. What are the most common payment methods used by customers?

NO DATA ABOUT PAYMENT METHODS.

b. Is there a relationship between the payment method and the order amount?

NO INFORMATION ABOUT PAYMENT METHOD AND ORDER AMOUNT IS AVAILABLE FOR ANALYSIS.

7. Customer Behavior:

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Ensure 'InvoiceDate' is in datetime format
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

# Define a reference date (e.g., the maximum invoice date in the dataset)
reference_date = data['InvoiceDate'].max()

# Calculate Recency (days since the last purchase)
recency = (reference_date - data.groupby('CustomerID')['InvoiceDate'].max()).dt.days

# Calculate Frequency (number of unique invoices per customer)
frequency = data.groupby('CustomerID')['InvoiceNo'].nunique()

# Calculate Monetary value (total revenue per customer)
monetary = data.groupby('CustomerID')['Revenue'].sum()

# Combine RFM data into a single DataFrame
rfm_data = pd.DataFrame({'Recency': recency, 'Frequency': frequency, 'Monetary': monetary}).reset_index()

# Scale the RFM values
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm_data[['Recency', 'Frequency', 'Monetary']])
```

```
# Perform KMeans clustering
num_clusters = 4
kmeans = KMeans(n_clusters=num_clusters, init='k-means++', random_state=42)
rfm_data['Cluster'] = kmeans.fit_predict(rfm_scaled)

# Cluster Profiles
cluster_profiles = rfm_data.groupby('Cluster').agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'Monetary': 'mean',
    'CustomerID': 'count'
}).rename(columns={'CustomerID': 'Number of Customers'})

print("Cluster Profiles:")
print(cluster_profiles)
```

RFM (Recency, Frequency, Monetary) analysis and client segmentation using KMeans clustering are carried out by this code. Initially, the InvoiceDate column is transformed into a datetime format, and the number of days since each customer's most recent purchase in relation to the dataset's most recent date is used to determine the recency. Monetary value is the total amount of money each customer generates, while frequency is the number of distinct invoices per customer. The values are scaled using StandardScaler once these metrics are merged into a single DataFrame. Customers are grouped into four categories according to their RFM scores using KMeans clustering. Lastly, each cluster's average frequency, monetary value, number of clients, and recency are summarized in the cluster profiles.

Results:

```
Cluster Profiles:
             Recency   Frequency       Monetary  Number of Customers
Cluster
0           9.752577   28.510309   12168.264691                  194
1         247.927577    1.805942     455.110716                 1077
2           4.090909  109.909091  124312.306364                   11
3          41.780906    4.370550    1320.981506                 3090
```

Based on their RFM scores, the profiles of four client clusters are compiled in this output. Customers in Cluster 0 have moderate frequency, average monetary value, and low recency (recent purchases). Clients with low frequency, modest expenditure, and high recency (having not made a purchase recently) are represented by Cluster 1, which denotes less active clients. Cluster 2, which probably consists of top buyers, has highly engaged clients with the highest monetary value, high frequency, and very low recency. Cluster 3 indicates sporadic buyers due to its somewhat recent consumers, low frequency, and small spending. The distribution of clients among various clusters is displayed in the "Number of Customers" column.

8. Returns and Refunds:

a. What is the percentage of orders that have experienced returns or refunds?

NO DATA ABOUT RETURN AND REFUNDS IS AVAILABLE

b. Is there a correlation between the product category and the likelihood of returns?

NO RELEVANT DATA AVAILABLE FOR ANALYSIS.

The information needed to determine the percentage of orders with returns or refunds is not available

9. Profitability Analysis:

a. Can you calculate the total profit generated by the company during the dataset's period?

```
# Convert 'InvoiceDate' to datetime
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

# Calculate Revenue
data['Revenue'] = data['Quantity'] * data['UnitPrice']

# Calculate Total Profit
total_profit = data['Revenue'].sum()
print("Total Profit:", total_profit)
```

This code calculates the total profit from a dataset of sales transactions. It first converts the InvoiceDate column to a datetime format for proper date handling. Next, it computes the Revenue for each transaction by multiplying the Quantity of items sold with their UnitPrice. Finally, the Revenue values are summed up to determine the total profit, which is displayed as the output.

Results:

```
Total Profit: 8300065.814000001
```

b. What are the top 5 products with the highest profit margins?

```
# Calculate Revenue per Product
data['Revenue'] = data['Quantity'] * data['UnitPrice']

# Group by 'Description' and calculate Profit and Revenue
product_profit = data.groupby('Description')['Revenue'].sum()
product_revenue = data.groupby('Description')['Revenue'].sum()

# Calculate Profit Margin
profit_margin = (product_profit / product_revenue) * 100

# Get Top 5 Products with Highest Profit Margins
top_products = profit_margin.nlargest(5)
print("Top 5 Products with Highest Profit Margins:")
print(top_products)
```

This code determines the top 5 products with the largest profit margins. First, it multiplies the quantity sold by the unit price to determine the revenue for each product. The total revenue (product_revenue) and profit (product_profit) are then determined by grouping the data according to the product description. These are used to divide profit by revenue and determine the profit_margin as a percentage. Lastly, a selection and display of the top 5 products with the best profit margins are made.

Results:

```
Top 5 Products with Highest Profit Margins:
Description
4 PURPLE FLOCK DINNER CANDLES    100.0
50'S CHRISTMAS GIFT BAG LARGE    100.0
DOLLY GIRL BEAKER                100.0
I LOVE LONDON MINI BACKPACK      100.0
I LOVE LONDON MINI RUCKSACK      100.0
Name: Revenue, dtype: float64
```

The output lists the top 5 products with the highest profit margins, all achieving a 100% margin. This implies that for these products, the revenue and profit values are equal, possibly due to specific pricing or cost structures. The listed items include products such as "4 PURPLE FLOCK DINNER CANDLES" and "I LOVE LONDON MINI BACKPACK," among others. These products stand out as the most profitable in the dataset based on margin, indicating their significance for maximizing profitability.

10. Customer Satisfaction:

a. Is there any data available on customer feedback or ratings for products or services?

NO DATA IS AVAILABLE ON CUSTOMER FEEDBACK OR RATINGS FOR PRODUCTS OR SERVICE.

b. Can you analyze the sentiment or feedback trends, if available?

NO DATA AVAILABLE TO ANALYZE THE SENTIMENT OR FEEDBACK TRENDS.

Conclusion

This project successfully demonstrated the use of RFM (Recency, Frequency, Monetary) analysis for customer segmentation, leveraging KMeans clustering to group customers into distinct behavioral categories. The analysis provided insights into customer engagement, spending patterns, and recency of purchases, which can be used to develop targeted marketing strategies and improve business outcomes

Key outcomes from this project include a comprehensive analysis of customer behavior, sales trends, and product performance that drives actionable insights. The segmentation of customers through RFM analysis and clustering revealed distinct purchasing patterns, enabling tailored marketing strategies for retention, loyalty, and re-engagement. High-profit margin products were identified, highlighting opportunities for maximizing revenue through targeted promotions and strategic cross-selling or up-selling initiatives.

The analysis also uncovered peak sales periods by time, day, month, and year, providing valuable insights for inventory management and campaign planning. Additionally, the correlation between customer metrics and order values was explored, offering a deeper understanding of revenue contributions by geographic regions and customer groups. While this report establishes a strong analytical foundation, it also emphasizes the potential for further refinement through expanded datasets, predictive modeling, and integration of additional customer feedback metrics to enrich business decision-making and long-term strategic growth.