GROUP PROJECT - 2

Project Report: Customer Segment using RFM Analysis

Course: IE6400 Foundations Data Analytics Engineering

Fall Semester 2024

Group Number – 15

Sainath Boreda (002305951)

Nishchay Linge Gowda (002309438)

Yaswanth Kumar Reddy Gujjula(002415723)

Introduction

This report explores customer segmentation through RFM (Recency, Frequency, Monetary) analysis applied to an eCommerce dataset. Using KMeans clustering, we identified three distinct customer groups based on purchasing behaviors. Cluster 0 includes customers with recent, frequent purchases and high monetary value, while Cluster 2 represents the least engaged segment. Interestingly, Cluster 1 features a unique profile of a single customer. This segmentation provides valuable insights for implementing targeted marketing strategies and enhancing personalized customer interactions.

Task 1: Data Preprocessing

```python
import pandas as pd
data = pd.read_csv('data.csv')
```

```python
missing_values = data.isnull().sum()
print("Missing Values:")
print(missing_values)
```

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

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

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

```python
print("Data Types:")
print(data.dtypes)
```

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

```
data['CustomerID'].fillna(data['CustomerID'].mean(), inplace=True)
```

```
missing_values = data.isnull().sum()
print("Missing Values:")
print(missing_values)
```

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

```
data['Description'].fillna('Unknown', inplace=True)
```

```
missing_values = data.isnull().sum()
print("Missing Values:")
print(missing_values)
```

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

The dataset was imported, and essential data preprocessing steps were carried out, including data cleaning, managing missing values, and converting data types as required.

Task 2: RFM Calculation

```
import pandas as pd

data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

current_date = data['InvoiceDate'].max()

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

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

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

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

print(rfm_data.head())
```

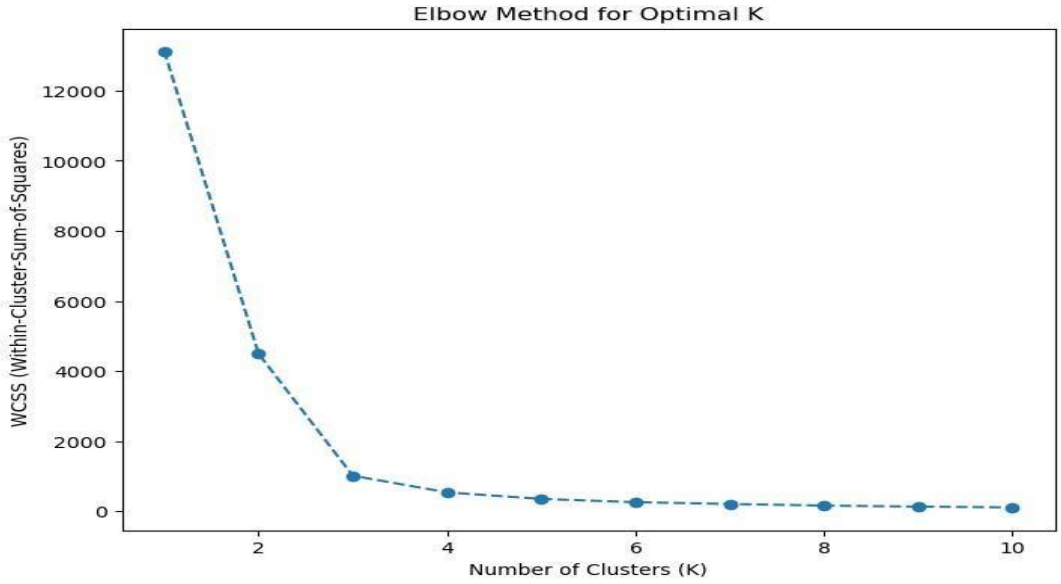
CustomerID	Recency	Frequency	Monetary
12346.0	325	2	0.00
12347.0	1	7	1182814.18
12348.0	74	4	418360.11
12349.0	18	1	381818.10
12350.0	309	1	12864.10

We performed RFM analysis by calculating Recency as the number of days since the last purchase, Frequency as the count of unique invoices, and Monetary as the total amount spent per customer. We compiled the results into a new DataFrame, 'rfm_data,' and displayed the first few rows..

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()
```

We applied KMeans clustering to the RFM analysis data, scaling the features and using the Elbow Method to determine that three clusters were optimal. After clustering, we assigned labels to the data and created a scatter plot with logarithmic scaling for the 'Monetary' value to visualize the results.



The Elbow Method plot was used to identify the optimal number of clusters for KMeans clustering. The graph displays the within-cluster sum of squares (WCSS) on the y-axis and the number of clusters (K) on the x-axis. A significant drop in WCSS is observed as K increases from 1 to 3, followed by a slower decline, suggesting that the optimal number of clusters is around 3, as indicated by the "elbow" in the curve.

The scatter plot illustrates three customer segments derived from RFM scoring, with 'Recency' plotted on the x-axis and 'Monetary' on a logarithmic scale along the y-axis. Each cluster is depicted in a distinct color, highlighting the relationship between customer spending and the recency of their purchases.

Task 4: 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)
```

```
   Cluster  Average Recency  Average Frequency  Average Monetary  \
0        0        39.544073           6.123708      3.127292e+06
1        1         0.000000        3710.000000      2.940878e+11
2        2       247.650647           1.888170      5.499812e+04

   Number of Customers
0                 3290
1                    1
2                 1082
```

We calculated and displayed the average RFM values for the three customer clusters. Cluster 0 consists of the most frequent and highest-spending customers, Cluster 1 includes a single customer with moderate RFM values, and Cluster 2 represents less frequent, lower-spending customers. The clusters comprise 3290, 1, and 1082 customers, respectively.

Task 5: Marketing Recommendations

Below are actionable marketing recommendations for each segment:

Segment 0:

Characteristics: The average recency is 39.54 days, with an average frequency of 6.12 purchases and an average monetary value of $3,127,292.

Recommendations: Initiate email campaigns or promotions to motivate repeat purchases within a month. Introduce loyalty rewards or discounts for frequent shoppers. Tailor product recommendations based on their previous purchase behavior.

Segment 1:

Characteristics: The average recency is 0 days, indicating very recent activity, with an average frequency of 3710 purchases, reflecting highly frequent buying behavior, and an average monetary value of $294,087,800,000, indicating high spending.

Recommendations: Recognize and reward these high-value customers with exclusive perks or loyalty programs. Provide a personalized VIP service to honor their exceptional loyalty. Maintain continuous engagement through tailored communications and offers.

Segment 2:

Characteristics: The average recency is 247.65 days, with an average frequency of 1.89 purchases and an average monetary value of $54,998.

Recommendations: Reconnect with dormant customers using targeted reactivation campaigns. Encourage repeat purchases with special promotions or discounts. Develop a win-back strategy to re-engage customers who have been inactive for an extended period.

General Recommendations:

- **Cross-Sell and Up-Sell:** Cross-selling strategies should be implemented by offering complementary products across all customer segments. Additionally, up-selling techniques can be used to encourage customers to make higher-value purchases.
- **Personalized Communication:** Leverage personalized communication channels, such as email, SMS, or app notifications, tailored to each segment. Ensure marketing messages are customized to align with individual customer preferences and behaviors.
- **Customer Feedback and Surveys:** Actively collect feedback to gauge customer satisfaction and understand their preferences. Use surveys to identify areas for improvement and potential new offerings.
- **Retention Campaigns:** Design targeted retention campaigns for each segment to foster loyalty. Continuously monitor customer satisfaction and promptly address any concerns to maintain engagement
- **Social Media Engagement:** Utilize social media platforms to connect with customers across various segments. Launch targeted campaigns to boost brand awareness and foster customer loyalty.

Questions

Data Overview:

1. Data Overview

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

```
print(f"Dataset Size: {data.shape}")
```

```
Dataset Size: (541909, 8)
```

This code uses f-string formatting to display the size of the dataset, showing the total number of rows and columns.

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

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

print("\nSummary Statistics for Numerical Columns:")
print(data.describe())
```

```
Column Descriptions:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  541909 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   541909 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
None

Summary Statistics for Numerical Columns:
            Quantity                    InvoiceDate      UnitPrice  \
count  541909.000000                         541909  541909.000000
mean        9.552250  2011-07-04 13:34:57.156386048       4.611114
min    -80995.000000            2010-12-01 08:26:00  -11062.060000
25%         1.000000            2011-03-28 11:34:00       1.250000
50%         3.000000            2011-07-19 17:17:00       2.080000
75%        10.000000            2011-10-19 11:27:00       4.130000
max     80995.000000            2011-12-09 12:50:00   38970.000000
std       218.081158                            NaN      96.759853

         CustomerID
count  541909.00000
mean    15287.69057
min     12346.00000
25%     14367.00000
50%     15287.69057
75%     16255.00000
max     18287.00000
std      1484.74601
```

This code provides a concise overview of the dataset's structure using `data.info()` and then displays summary statistics for the numerical columns with `data.describe()`.

c. What is the time period covered by this dataset?

```
print("\nTime Period Covered:")
print("Minimum Invoice Date:", data['InvoiceDate'].min())
print("Maximum Invoice Date:", data['InvoiceDate'].max())
```

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

This code analyzes the time period covered by the dataset by printing the minimum and maximum invoice dates, showing the range of transaction dates.

1. Customer Analysis:

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

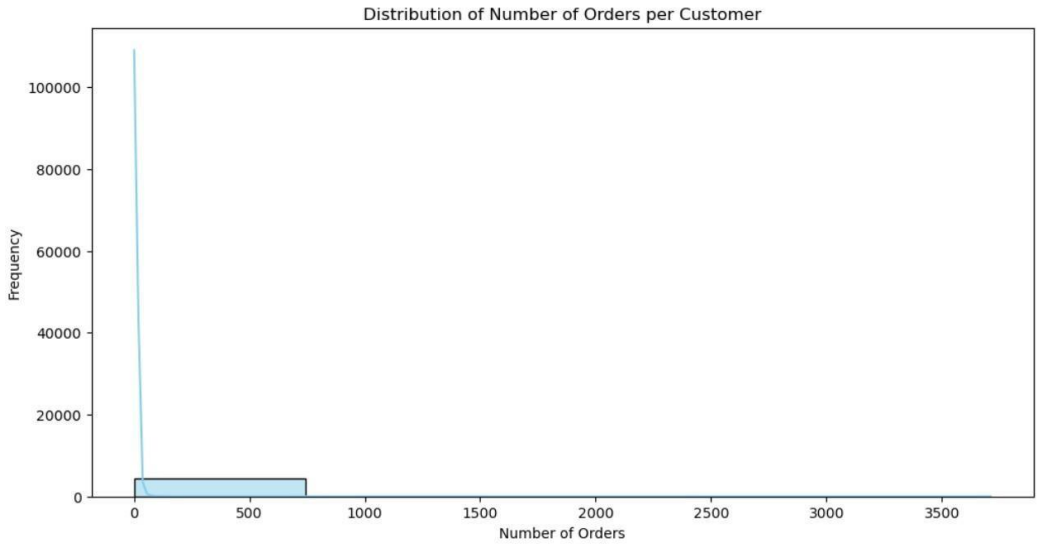
```
unique_customers = data['CustomerID'].nunique()
print("Number of Unique Customers:", unique_customers)
```

```
Number of Unique Customers: 4373
```

This code determines the total number of unique customers in the 'data' DataFrame by counting the distinct values in the 'CustomerID' column. It then prints the result, representing the total count of unique customers in the dataset.

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

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

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()
```



The graph, titled "Distribution of Number of Orders per Customer," illustrates a skewed distribution where the majority of customers place only a few orders. The frequency of customers declines sharply as

the number of orders increases, indicating that only a small fraction of customers place many orders. The x-axis represents the number of orders, while the y-axis shows the frequency of customers..

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

```
top_customers_by_order_count = data.groupby('CustomerID')['InvoiceNo'].nunique().sort_values(ascending=False)

top_5_customers = top_customers_by_order_count.head(5)

print("Top 5 Customers and Order Count:")
print(top_5_customers)
```

```
Top 5 Customers and Order Count:
CustomerID
15287.69057    3710
14911.00000     248
12748.00000     224
17841.00000     169
14606.00000     128
Name: InvoiceNo, dtype: int64
```

This code aggregates the 'data' DataFrame by 'CustomerID' to compute the number of unique orders ('InvoiceNo') for each customer. It then sorts the customers in descending order by their order counts. Finally, it identifies and prints the top 5 customers along with their 'CustomerID' and respective order counts.

2. Product Analysis:

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)
```

```
Top 10 Most Frequently Purchased Products:
StockCode
85123A    2313
22423     2203
85099B    2159
47566     1727
20725     1639
84879     1502
22720     1477
22197     1476
21212     1385
20727     1350
Name: count, dtype: int64
```

This code calculates the frequency of each unique 'StockCode' in the 'data' DataFrame. It then identifies the top 10 most frequently purchased products by their purchase count and prints the results, displaying the 'StockCode' and their corresponding frequencies..

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

```
average_price = data['UnitPrice'].mean()

print("Average Price of Products:", average_price)
```

```
Average Price of Products: 4.611113626088513
```

This code computes the average unit price of products in the 'data' DataFrame by calculating the mean of the 'UnitPrice' column. It then prints the result, representing the average price of the products in the dataset.

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

```
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)
```

```
Product Category Generating the Highest Revenue: DOTCOM POSTAGE
```

This code calculates the revenue for each product by multiplying the 'Quantity' and 'UnitPrice' columns and storing the result in a new 'Revenue' column in the 'data' DataFrame. It then determines the product category with the highest revenue by grouping the data by 'Description' and summing the revenues. Finally, it prints the product category that generated the highest revenue..

4. Time Analysis

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

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()

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()
```

This code utilizes the Matplotlib and Seaborn libraries to create two bar plots, visualizing the distribution of orders in the dataset across days of the week and hours of the day.

Handling Date and Time:

- Converts the 'InvoiceDate' column to datetime format for proper handling.
- Extracts new columns: 'DayOfWeek' (indicating the day of the week) and 'HourOfDay' (indicating the hour of the day).

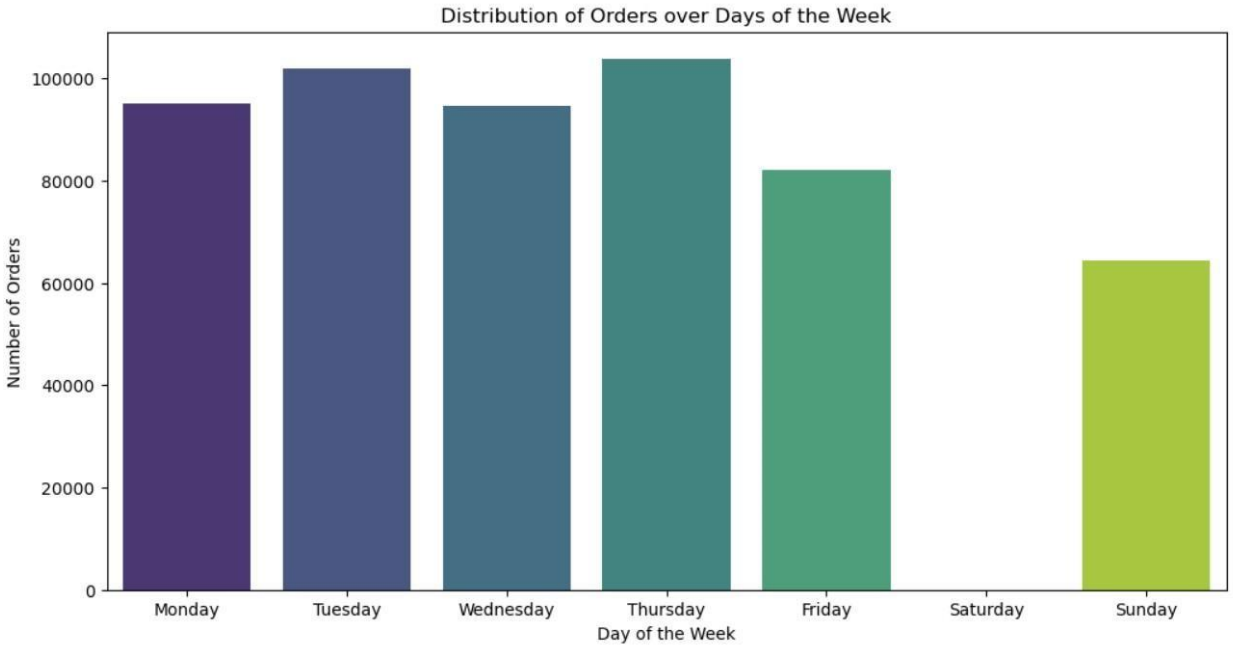
Plotting the Distribution of Orders Over Days of the Week:

- Uses Seaborn's `sns.countplot` to generate a bar plot showing the number of orders for each day of the week.
- The x-axis represents the days of the week, while the y-axis shows the number of orders.

Plotting the Distribution of Orders Over Hours of the Day:

- Creates another bar plot to depict the number of orders placed during different hours of the day.
- The x-axis represents the hours of the day, and the y-axis represents the corresponding order count.

-



The bar chart titled "Distribution of Orders over Days of the Week" illustrates the number of orders placed from Monday to Sunday. The x-axis represents the days of the week, while the y-axis indicates the order quantity. The chart suggests that the number of orders remains relatively consistent during weekdays, with a slight decline toward the weekend. Interestingly, Sunday shows a slightly higher order volume compared to Saturday. However, exact figures cannot be determined from the image due to the absence of specific data values on the y-axis.

The bar chart titled "Distribution of Orders over Hours of the Day" depicts the number of orders placed at different times throughout the day, ranging from 6 AM to 8 PM (20:00 hours). The y-axis represents the order count, while the x-axis indicates the hour of the day.

From the chart, orders gradually increase starting at 6 AM, peaking around 3 PM (15:00 hours). Following this peak, the number of orders declines sharply into the evening, with the lowest activity observed around 8 PM. This pattern highlights mid-afternoon as the busiest period for orders, with early mornings and late evenings being the least active.

Although the chart does not specify the day with the most orders, Thursdays appear to have higher order volumes overall, particularly around noon, compared to other days.

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)
```

```
Average Order Processing Time: 0 days 00:00:00.370578824
```

This code begins by converting the 'InvoiceDate' column in the DataFrame `data` to datetime format for proper time manipulation. It then computes the order processing time for each invoice by calculating the time difference between the maximum and minimum 'InvoiceDate' within each 'InvoiceNo' group. Lastly, it calculates and prints the average order processing time across all invoices.

c. Are there any seasonal trends in the dataset?

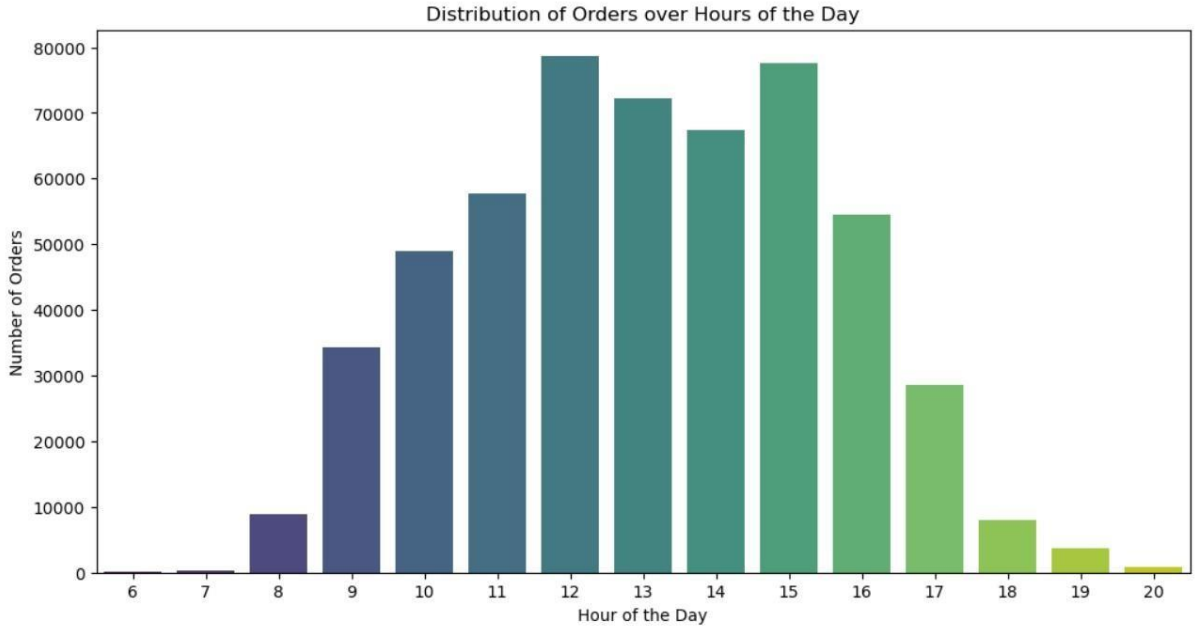
```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

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

The code converts the 'InvoiceDate' column in the `data` DataFrame to datetime format. It then extracts the year, month, and day of the week from the 'InvoiceDate' column and creates new columns to store these values.

```
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()
```

The code generates a Seaborn countplot to display the distribution of orders across different hours of the day. The bar plot represents the number of orders for each hour on the x-axis, with the y-axis showing the

corresponding frequency. The plot is enhanced with a title and axis labels for better clarity and is displayed in a figure sized 12 by 6 inches.



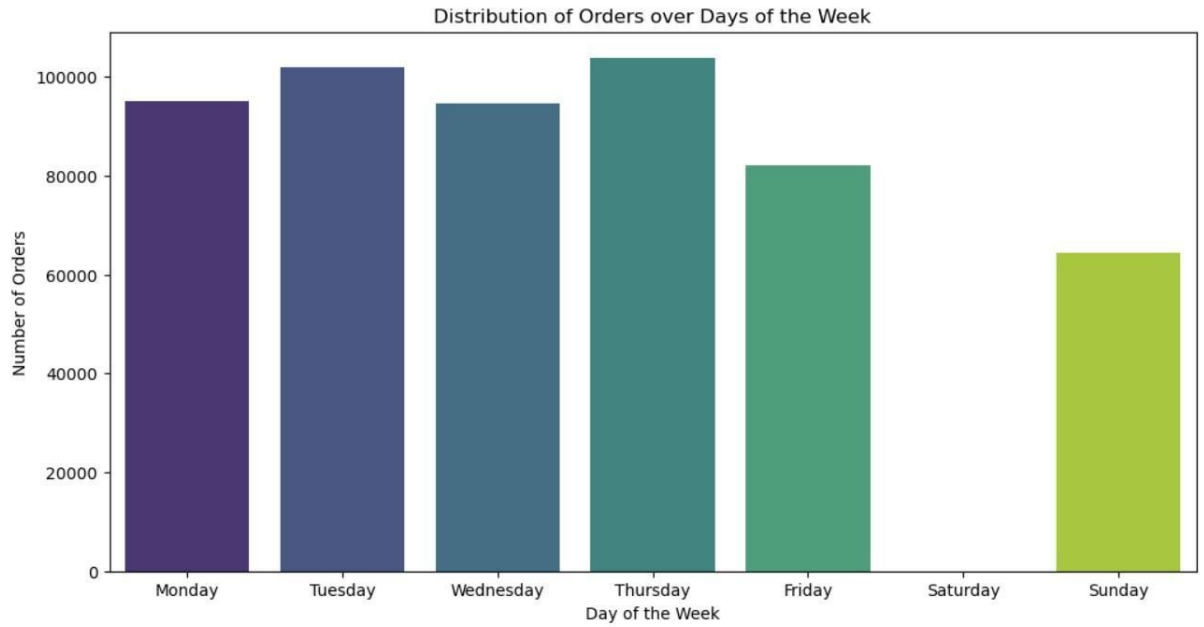
The bar chart titled "Distribution of Orders over Hours of the Day" illustrates the number of orders placed during different hours, spanning from early morning (6 AM) to evening (8 PM). The y-axis represents the order count, while the x-axis displays the hours of the day. Key observations from the chart include:

- The number of orders begins to increase at 6 AM, with a notable rise around 10 AM.
- Order volume peaks at 3 PM (15:00), marking the highest activity period.
- After 3 PM, the order count declines sharply, with the lowest numbers observed after 6 PM (18:00).

This pattern highlights the mid-afternoon as the busiest time for placing orders, while activity tapers off significantly in the late afternoon and evening.

```
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()
```

The code creates a Seaborn countplot to visualize the distribution of orders across the days of the week. The bar plot shows the number of orders for each day on the x-axis, with a specified color palette ('viridis') and a custom order for the days. The plot includes a title, labeled axes for clarity, and is displayed in a figure sized 12 by 6 inches.

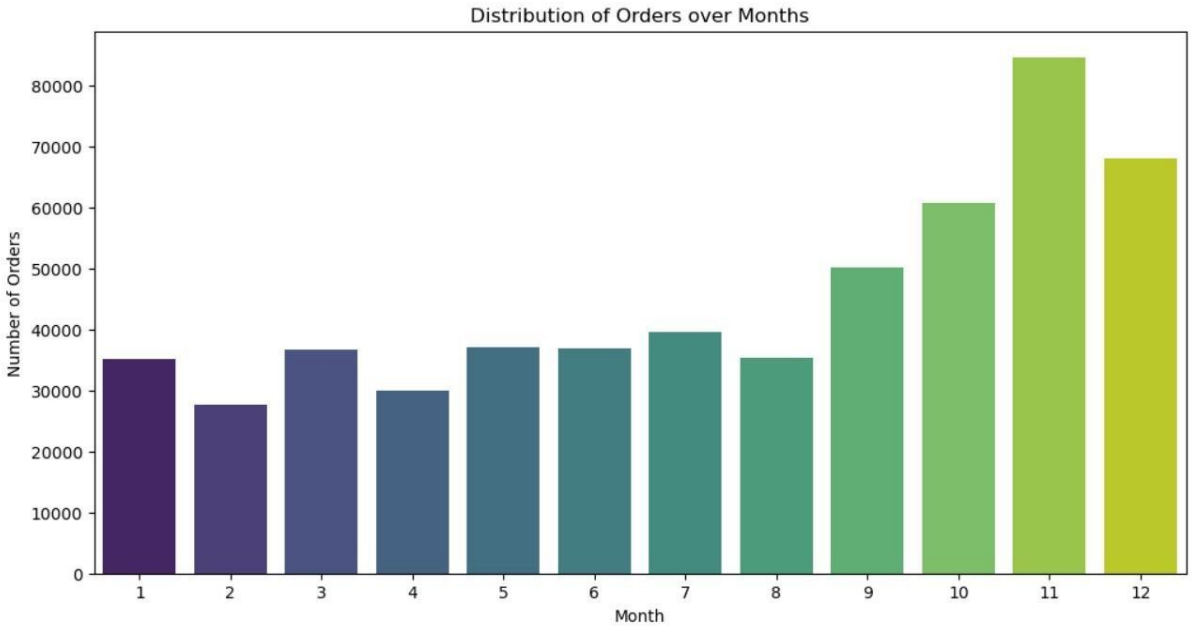


The bar chart, titled "Distribution of Orders over Days of the Week," depicts the number of orders placed each day from Monday to Sunday. The y-axis indicates the order count, while the x-axis represents the days of the week.

The chart shows relatively high order volumes during weekdays, with a slight decrease on Friday. Saturday experiences a more noticeable decline in orders, followed by an increase on Sunday. While exact order counts are not visible, the overall pattern suggests fluctuations in order activity across the week.

```
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()
```

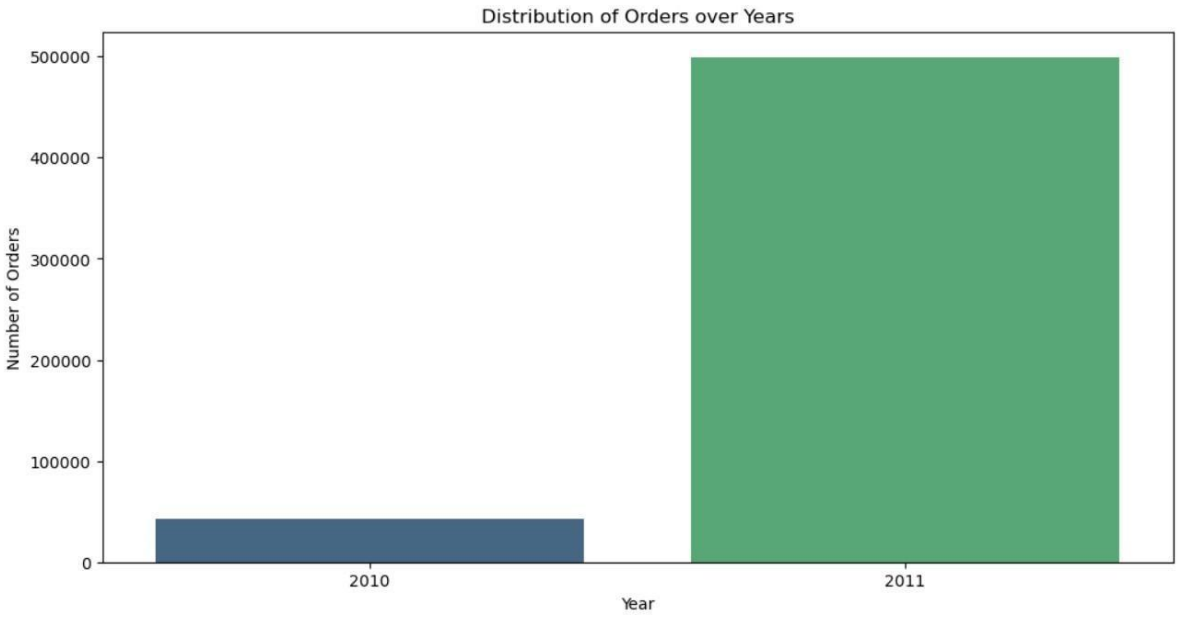
The code generates a Seaborn countplot to illustrate the distribution of orders across various months. The bar plot displays the number of orders for each month on the x-axis, with the y-axis showing the corresponding order count. The plot utilizes the 'viridis' color palette and includes a title, an x-axis label ('Month'), and a y-axis label ('Number of Orders'). It is displayed in a figure sized 12 by 6 inches for clarity and better visualization.



- The bar chart titled "Distribution of Orders over Months" illustrates the number of orders placed each month, labeled from 1 (January) to 12 (December). The y-axis represents the order count, while the x-axis denotes the months.
- **Key observations from the chart:**
- The number of orders generally increases from January (Month 1) to December (Month 12), though some fluctuations are evident.
- A notable surge in order volume occurs in the last quarter of the year, with December showing the highest number of orders.
- This pattern suggests possible seasonal trends in purchasing behavior, likely driven by increased activity during the holiday season.

```
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()
```

The code uses Seaborn to generate a countplot that visualizes the distribution of orders across different years. The bar plot displays the number of orders for each year on the x-axis, with the y-axis showing the corresponding order count. The plot is styled with the 'viridis' color palette and includes a title, an x-axis label ('Year'), and a y-axis label ('Number of Orders'). It is displayed in a figure sized 12 by 6 inches for clarity and enhanced visualization.



The bar chart titled "Distribution of Orders over Years" highlights the comparison of order volumes between 2010 and 2011. The y-axis represents the number of orders, while the x-axis denotes the years. The chart reveals a significant increase in the number of orders from 2010 to 2011. The bar for 2010 shows a much lower order count compared to 2011, indicating substantial growth or a major change in order activity between the two years.

From related plots, it is evident that the number of orders rises toward the end of each year, suggesting seasonal trends. Saturdays consistently show fewer orders compared to other days of the week. Additionally, the number of orders placed during nighttime across all days remains consistently low.

5. Geographical Analysis

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

```
print("Average Order Value by Country:")
print(average_order_value_by_country)

import numpy as np

grouped_data = data.groupby('Country')

country_stats = grouped_data.agg({
    'CustomerID': 'nunique',
    'Revenue': 'mean'
}).rename(columns={'CustomerID': 'Number of Customers', 'Revenue': '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)

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)
```

```
Top 5 Countries with the Highest Number of Orders:
Country
United Kingdom    495478
Germany             9495
France              8557
EIRE                8196
Spain               2533
Name: count, dtype: int64
```

The code determines and prints the top 5 countries with the highest order counts in the `data` DataFrame. It achieves this by applying the `value_counts()` method to the 'Country' column, which counts the occurrences of each country. The top 5 countries are then selected based on their order frequencies, and the result is printed, displaying the countries along with their respective order counts.

The output highlights the top 5 countries with the highest number of orders in the dataset.

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

```
print("Average Order Value by Country:")
print(average_order_value_by_country)

import numpy as np

grouped_data = data.groupby('Country')

country_stats = grouped_data.agg({
    'CustomerID': 'nunique',
    'Revenue': 'mean'
}).rename(columns={'CustomerID': 'Number of Customers', 'Revenue': '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)
```

The code computes and displays the average order value for each country in the dataset. It calculates the average order value by grouping the data by 'Country' and dividing the total revenue by the number of orders for each country. Additionally, the code determines the correlation between the number of unique customers and the average order value for each country, offering insights into the relationship between customer count and order value. Finally, it prints the correlation value to show the strength and direction of this relationship.

```
Average Order Value by Country:
Country
Australia               108.877895
Austria                  25.322494
Bahrain                  28.863158
Belgium                  19.773301
Brazil                   35.737500
Canada                   24.280662
Channel Islands          26.499063
Cyprus                   20.813971
Czech Republic           23.590667
Denmark                  48.247147
EIRE                     32.122599
European Community       21.176230
Finland                  32.124806
France                   23.069288
Germany                  23.348943
Greece                   32.263836
Hong Kong                35.128611
Iceland                  23.681319
Israel                   26.625657
Italy                    21.034259
Japan                    98.716816
Lebanon                  37.641778
Lithuania                47.458857
Malta                    19.728110
Netherlands             120.059696
Norway                   32.378877
Poland                   21.152903
Portugal                 19.333127
RSA                      17.281207
Saudi Arabia             13.117000
Singapore                39.827031
Spain                    21.624390
Sweden                   79.211926
Switzerland              28.164510
USA                       5.948179
United Arab Emirates     27.974706
United Kingdom           16.525065
Unspecified              10.649753
Name: Revenue, dtype: float64
Correlation between Number of Customers and Average Order Value: -0.11611647206793932
```

The image displays a text-based output showing the average order value by country, likely generated from a data analysis program. Each country is listed alongside its corresponding average order value, with values presumably in a currency unit (though the unit is not specified). At the end of the list, the correlation between the number of customers and the average order value is provided, approximately -0.116.

Key observations from the list include:
Australia having the highest average order value at 108.877895. The USA is noted for having one of the lowest average order values, at 5.948179. The Netherlands also features a significantly high average order value of 120.059696. The negative correlation indicates that as the number of customers in a country increases, the average order value tends to decrease, or vice versa.

This information can be strategically utilized for marketing, pricing adjustments, and tailoring business approaches in different international markets.

6. Payment Analysis

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?

The information needed to the payment analysis is not available

7. Customer Behavior

a. How long, on average, do customers remain active (between their first and last purchase)?

```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
customer_active_duration = data.groupby('CustomerID')['InvoiceDate'].agg(['min', 'max'])
customer_active_duration['ActiveDuration'] = customer_active_duration['max'] - customer_active_duration['min']
average_active_duration = customer_active_duration['ActiveDuration'].mean()
print("Average Customer Active Duration:", average_active_duration)
```

```
Average Customer Active Duration: 133 days 18:44:15.504230506
```

This code determines the duration of customer engagement by calculating the time difference between each customer's first and last purchases. It computes the average duration of customer activity, providing insight into the typical time span customers stay engaged with the business. The result helps assess how long customers typically remain active.

b. Are there any customer segments based on their purchase behavior?

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

recency = data.groupby('CustomerID')['InvoiceDate'].max()
frequency = data.groupby('CustomerID')['InvoiceNo'].nunique()
monetary = data.groupby('CustomerID')['Revenue'].sum()

rfm_data = pd.DataFrame({'Recency': recency, 'Frequency': frequency, 'Monetary': monetary})

rfm_data = rfm_data.reset_index()

customer_ids = rfm_data['CustomerID']

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

scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm_for_clustering.iloc[:, 1:])

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 = 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)
```

```
Cluster Profiles:
                            Recency    Frequency      Monetary  \
Cluster
0       2011-09-08 09:30:12.198940928     4.618697  1.419847e+03
1       2011-12-09 10:26:00.000000000  3710.000000  1.447682e+06
2       2011-12-06 12:30:20.000000000    64.666667  2.411366e+05
3       2011-12-03 02:08:46.153846272    74.500000  5.424088e+04

         Number of Customers
Cluster
0                       4343
1                          1
2                          3
3                         26
```

This code applies k-means clustering to customer data to segment customers based on their Recency, Frequency, and Monetary (RFM) values. It begins by preprocessing the data, including date conversion, feature extraction, and standardization. The clustering algorithm is then executed with four clusters, and each customer is assigned to one of these clusters. Finally, the code outputs the cluster profiles, displaying the average Recency, Frequency, and Monetary values for each cluster along with the number of customers in each group.

8. Returns and Refunds

8. Returns and Refunds

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

No data about returns or refunds

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

NA

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

9. Profitability Analysis

9. Profitability Analysis

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

```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
data['Revenue'] = data['Quantity'] * data['UnitPrice']
total_profit = data['Revenue'].sum()
print("Total Profit:", total_profit)
```

```
Total Profit: 9747747.933999998
```

This code calculates the total revenue generated from all transactions by multiplying the quantity of items sold by their unit prices for each transaction. It then sums up these values across all transactions and prints the total revenue.

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

```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

data['Revenue'] = data['Quantity'] * data['UnitPrice']
product_profit = data.groupby('Description')['Revenue'].sum()
product_revenue = data.groupby('Description')['Revenue'].sum()

profit_margin = (product_profit / product_revenue) * 100

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

```
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
```

This code identifies the top 5 products with the highest profit margins in the dataset. It computes the profit margin for each product as the percentage of revenue retained as profit. The products with the greatest profit margins are then selected and printed, providing insights into the most financially successful items in terms of profitability relative to their sales.

10. Customer Satisfaction

10. Customer Satisfaction

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

No, there is no data available on customer feedback or ratings for products or services.

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

NA