GROUP PROJECT-2

Project Report: Customer Segment using RFM Analysis

Course: IE6400 Foundations Data Analytics Engineering

Fall Semester 2023

Group Number - 29

Sri Sai Prabhath Reddy Gudipalli (002207631)

Korukonda Raghavinisha (002832571)

Deepthi Umesha (002661626)

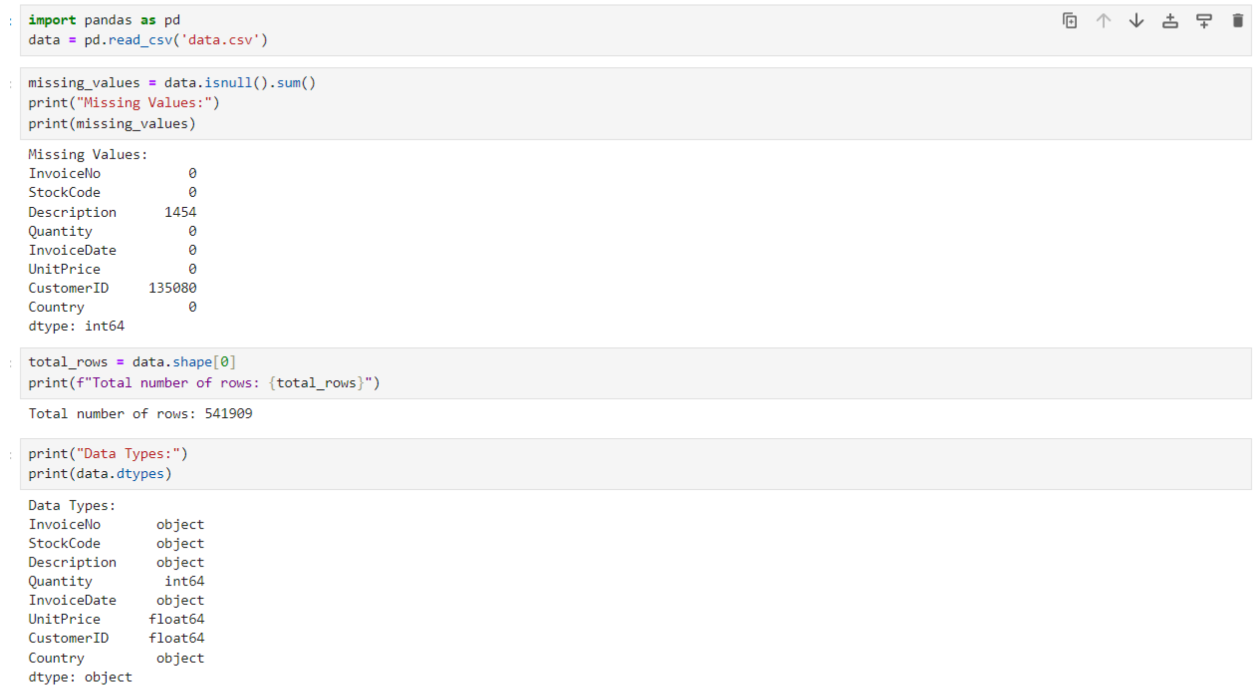
Ashwini Mahadevaswamy (002661627)

Yashwant Dontam (002844794)

Introduction

In this report, we delve into customer segmentation using RFM (Recency, Frequency, Monetary) analysis on an eCommerce dataset. The segmentation is achieved through KMeans clustering, resulting in three distinct customer groups based on their purchasing patterns. Cluster 0 is characterized by recent and frequent purchases with high monetary value, while Cluster 2 represents the least engaged customers. Notably, Cluster 1 consists of a single, unique customer profile. This strategic segmentation allows for targeted marketing initiatives and personalized customer engagement.

Task 1: Data Preprocessing



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

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

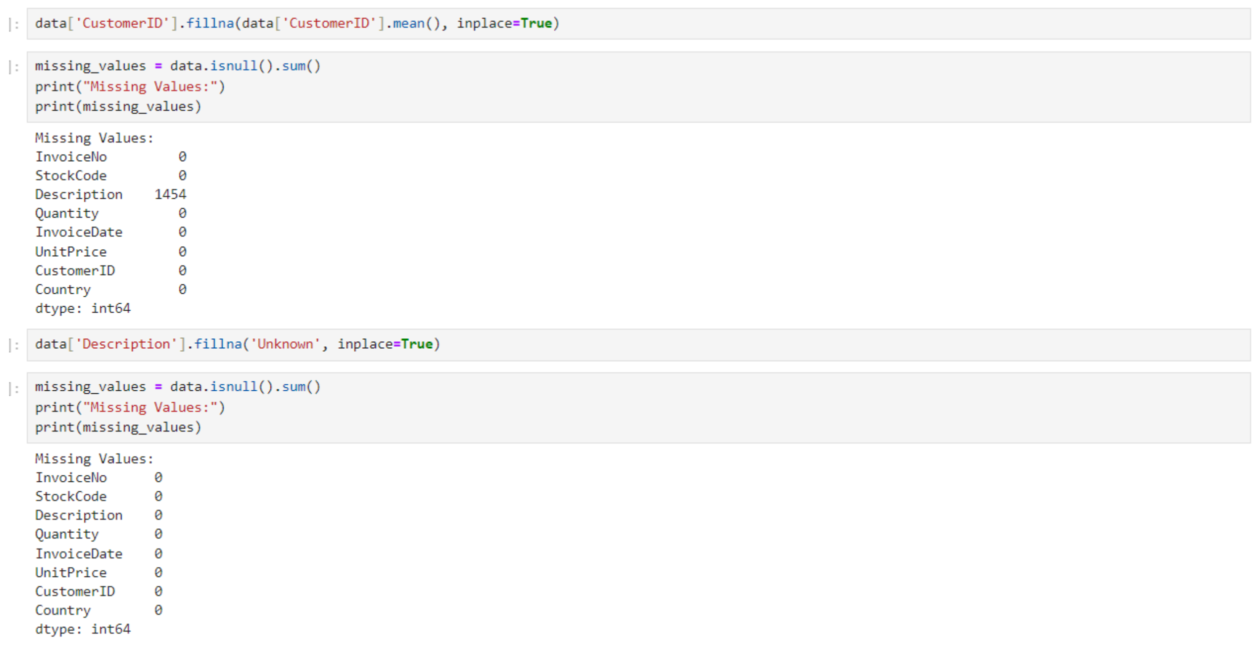
```
Missing Values:
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

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

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

```
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
```

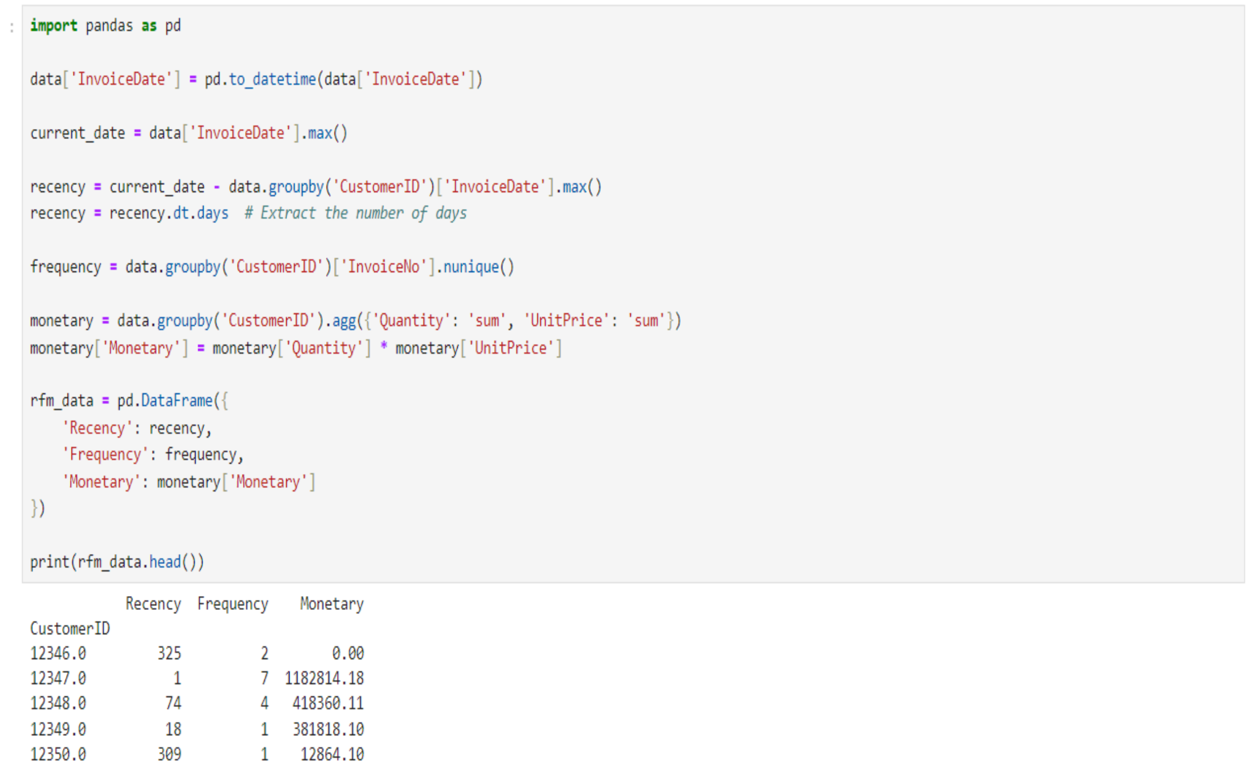
```
[ ]: data['Description'].fillna('Unknown', inplace=True)
```

```
[ ]: 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
```

Imported the dataset and performed necessary data preprocessing steps, including data cleaning, handling missing values, and converted data types.

Task 2: RFM Calculation

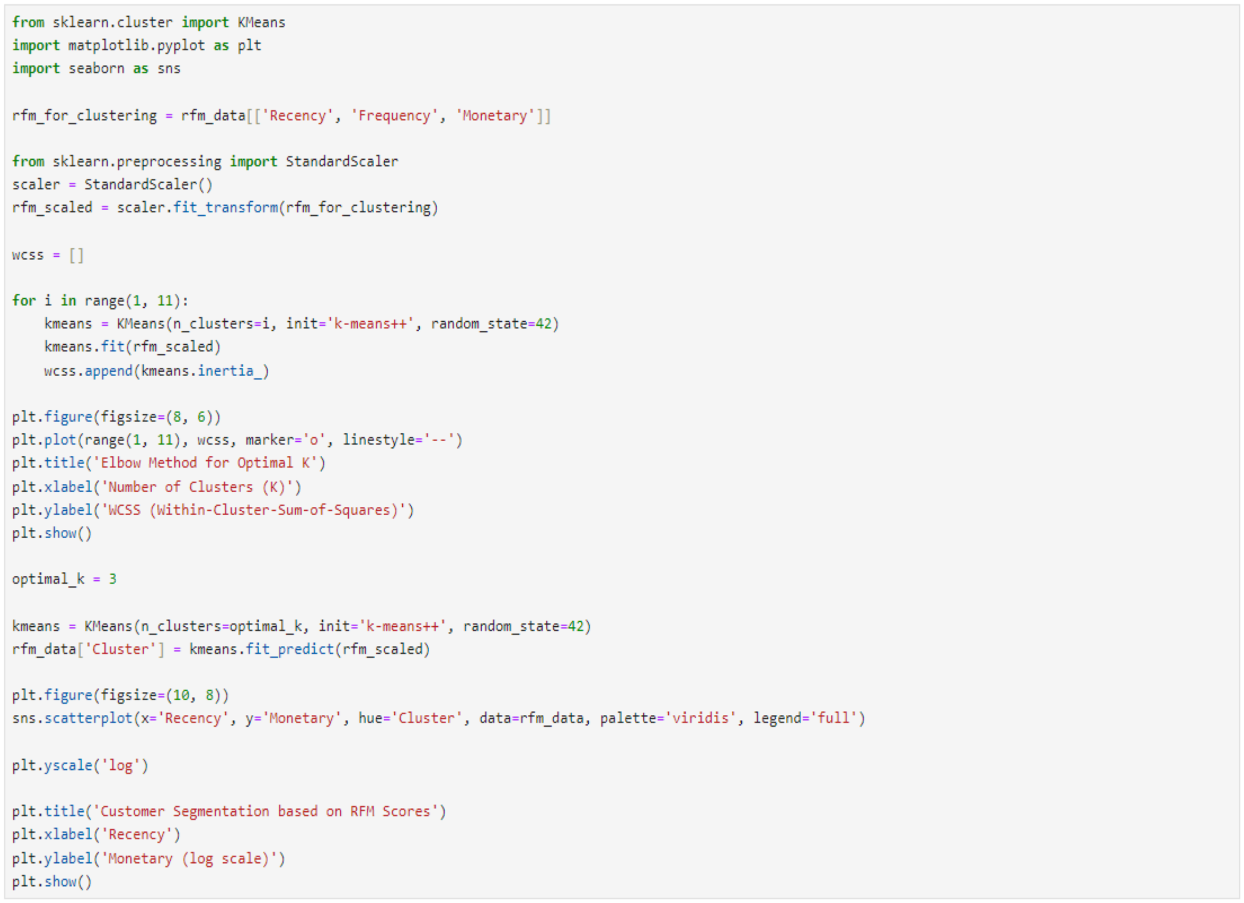


```python
import pandas as pd

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())
```

```
            Recency  Frequency    Monetary
CustomerID                                
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
```

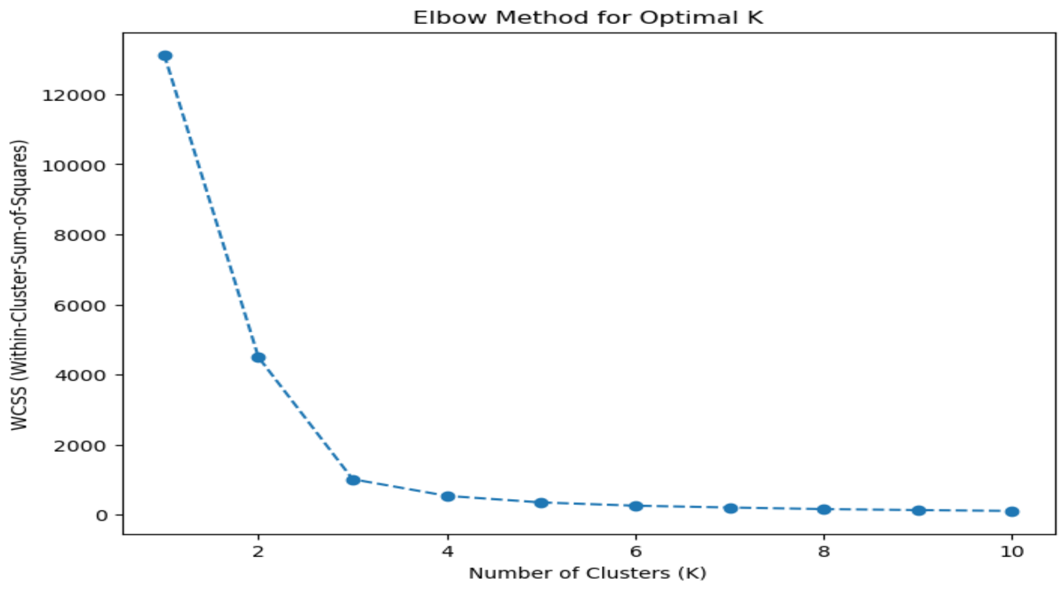
Performed RFM analysis, calculating Recency as days since the last purchase, Frequency as the count of unique invoices, and Monetary as the total spent per customer. The results are compiled into a new DataFrame, 'rfm_data', and the first few rows are displayed.

Task 3: RFM Segmentation, Customer Segmentation and Visualization

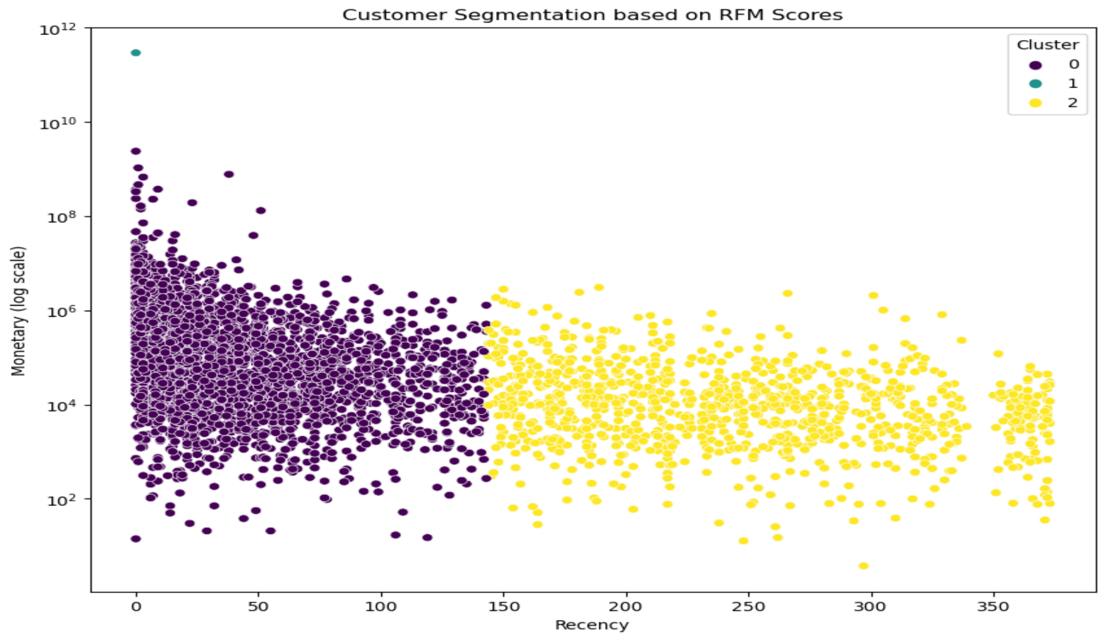


```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

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()
```

Executed KMeans clustering on RFM analysis data, scaling the features and determining the optimal number of clusters using the Elbow Method, which is identified as three. And then performed the clustering, assigned the cluster labels to the data, and visualized the results in a scatter plot with logarithmic scaling for the 'Monetary' value.



Elbow Method plot used to determine the optimal number of clusters for KMeans clustering. The graph shows the within-cluster sum of squares (WCSS) on the y-axis and the number of clusters (K) on the x-axis. The plot suggests a sharp decline in WCSS as K increases from 1 to 3, after which the decline slows, indicating that the optimal K is likely around 3, as the elbow of the curve is at this point.



The scatter plot visualizes three customer segments from RFM scoring with 'Recency' on the x-axis and 'Monetary' on a log scale y-axis. Each cluster is represented by a different color, showing the distribution of customer spending against the recency of purchase.

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
```

Calculated and displayed the average RFM values for three customer clusters. Cluster 0 has the most frequent and highest spenders, Cluster 1 has only one customer with moderate RFM values, and Cluster 2 has less frequent, lower-spending customers. The clusters contain 3290, 1, and 1082 customers, respectively.

Task 5: Marketing Recommendations

Here are actionable marketing recommendations for each segment:

Segment 0:

Characteristics: Average Recency: 39.54 days Average Frequency: 6.12 purchases Average Monetary: $3,127,292

Recommendations: Launch targeted email campaigns or promotions to encourage repeat purchases within a month. Offer loyalty programs or discounts for customers making frequent purchases. Provide personalized product recommendations based on their purchase history.

Segment 1:

Characteristics: Average Recency: 0 days (Very recent) Average Frequency: 3710 purchases (Highly frequent) Average Monetary: $294,087,800,000 (High spending)

Recommendations: Acknowledge and reward these high-value customers with exclusive perks or loyalty programs. Implement a personalized VIP service for their exceptional loyalty. Continuously engage with them through personalized communications and offers.

Segment 2:

Characteristics: Average Recency: 247.65 days Average Frequency: 1.89 purchases Average Monetary: $54,998

Recommendations: Re-engage dormant customers with targeted reactivation campaigns. Offer special promotions or discounts to encourage a second purchase. Implement a win-back strategy to bring back customers who haven't purchased recently.

General Recommendations:

Cross-Sell and Up-Sell: Implement cross-selling strategies based on complementary products for all segments. Encourage higher-value purchases through up-selling techniques.

Personalized Communication: Use personalized communication channels for each segment (e.g., email, SMS, app notifications). Tailor marketing messages based on individual customer preferences and behaviors.

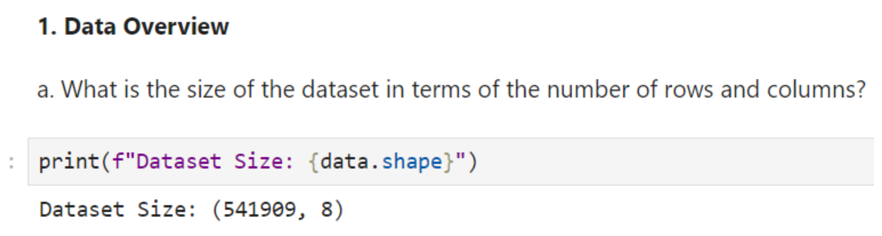
Customer Feedback and Surveys: Collect feedback to understand customer satisfaction and preferences. Use surveys to gather insights into potential improvements or new offerings.

Retention Campaigns: Develop customer retention campaigns for each segment to strengthen loyalty. Monitor customer satisfaction and address concerns promptly.

Social Media Engagement: Leverage social media platforms to engage with customers from different segments. Run targeted social media campaigns to increase brand awareness and loyalty.

Questions

1. Data Overview:



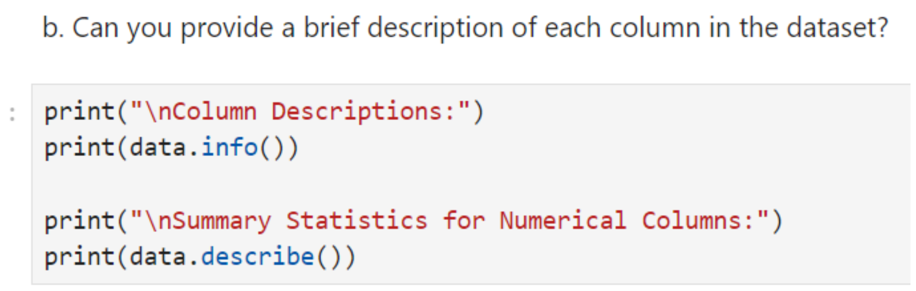
1. Data Overview

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

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

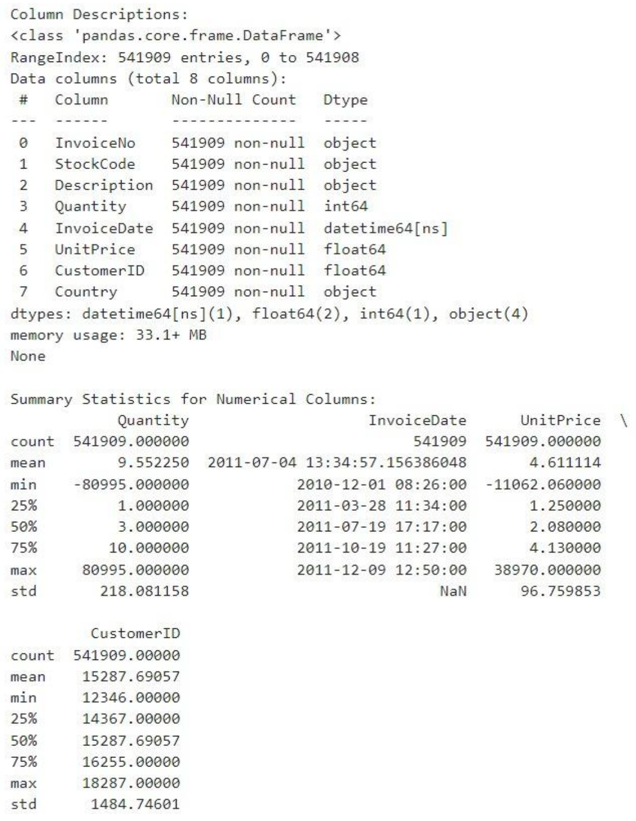
```
Dataset Size: (541909, 8)
```

This code prints the size of the dataset, indicating the number of rows and columns, using the f-string formatting.



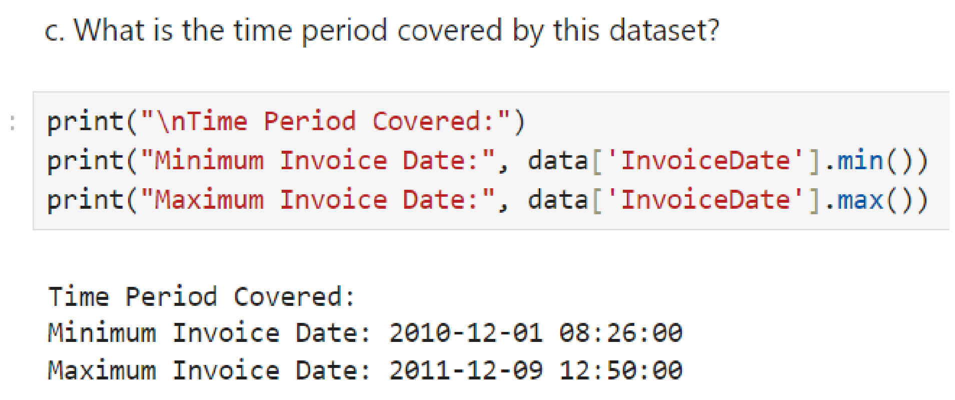
b. Can you provide a brief description of each column in the dataset?

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



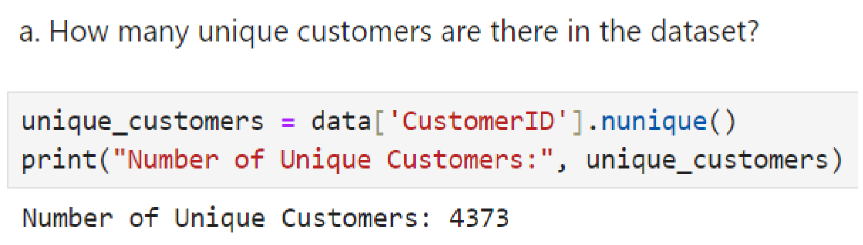
c. What is the time period covered by this dataset?

```
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 provides insights into the time period covered by the dataset. It prints the minimum and maximum invoice dates, indicating the range of transactions.

2. Customer Analysis:

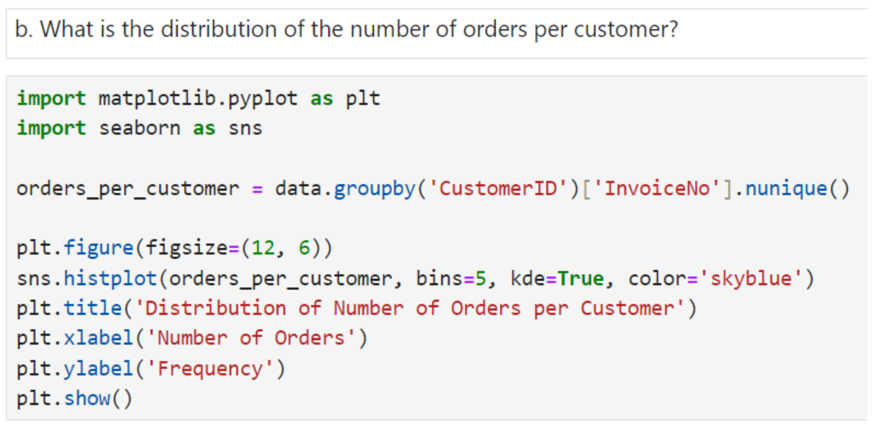


a. How many unique customers are there in the dataset?

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

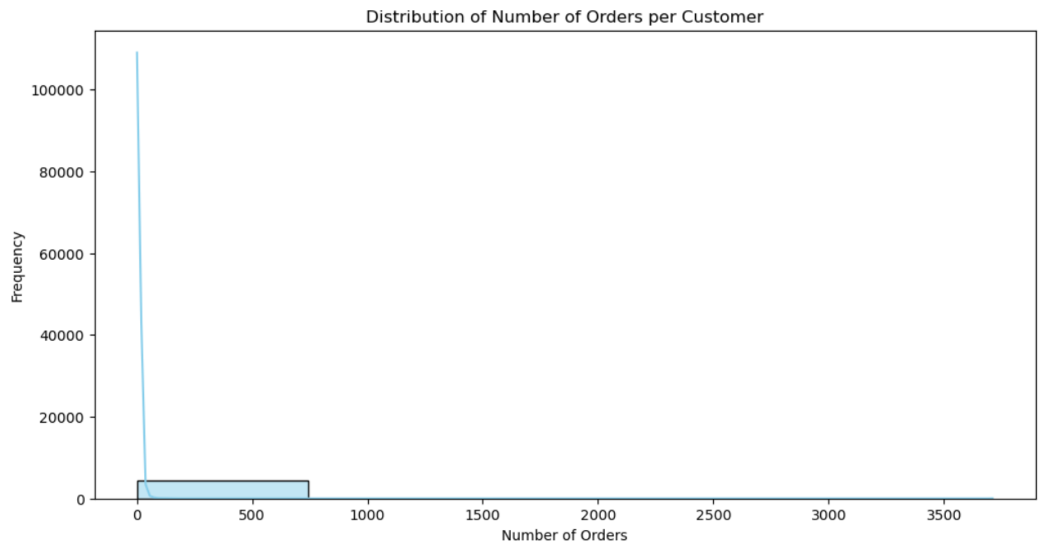
```
Number of Unique Customers: 4373
```

This code calculates the number of unique customers in the 'data' DataFrame by counting the distinct 'CustomerID' values. It then prints the result, indicating 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 image shows a graph titled "Distribution of Number of Orders per Customer". It displays a skewed distribution with most customers placing a small number of orders. The frequency drastically decreases as the number of orders per customer increases, suggesting that few customers place many orders. The x-axis represents the number of orders, and the y-axis indicates the frequency of customers.

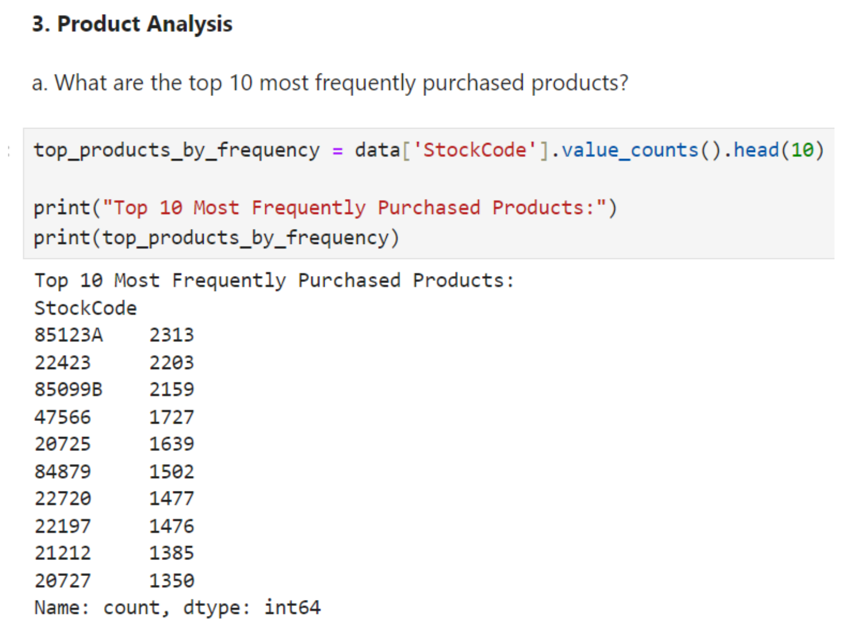
c. Can you identify the top 5 customers who have made the most purchases by order count?

```
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 groups the 'data' DataFrame by 'CustomerID' and calculates the number of unique orders ('InvoiceNo') for each customer. It then sorts the customers in descending order based on their order count. Finally, it selects the top 5 customers with the highest order counts and prints their 'CustomerID' and corresponding order counts.

3. Product Analysis:



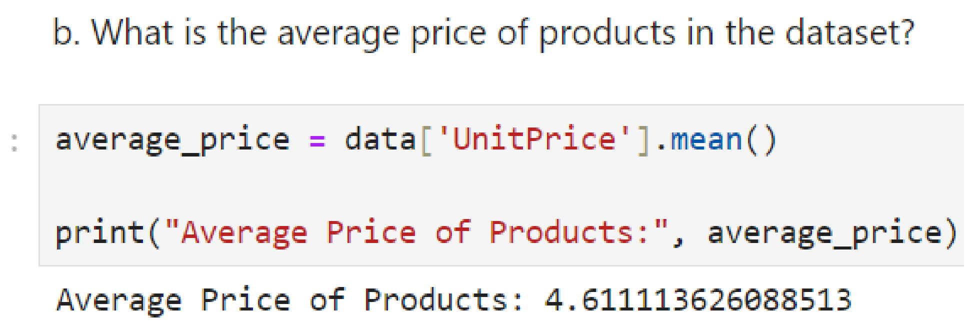
3. Product Analysis

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

```python
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 identifies and counts the frequency of each unique 'StockCode' in the 'data' DataFrame. It then selects the top 10 most frequently purchased products based on this count and prints the result, showing the StockCodes and their corresponding purchase frequencies.

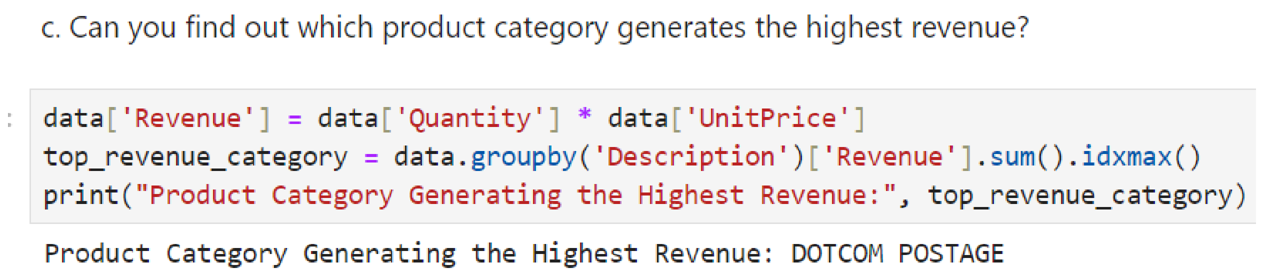


b. What is the average price of products in the dataset?

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

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

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

This code calculates the average unit price of products in the 'data' DataFrame. It computes the mean of the 'UnitPrice' column and prints the result, indicating the average price of the products in the dataset.



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

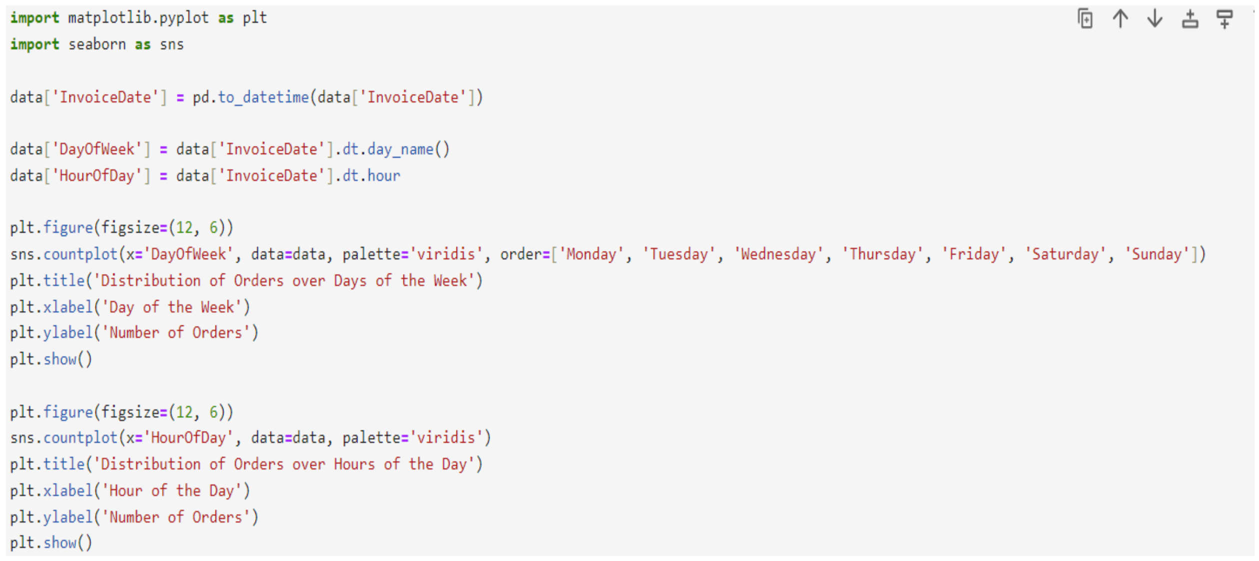
```python
data['Revenue'] = data['Quantity'] * data['UnitPrice']
top_revenue_category = data.groupby('Description')['Revenue'].sum().idxmax()
print("Product Category Generating the Highest Revenue:", top_revenue_category)
```

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

This code computes the revenue for each product by multiplying the quantity and unit price, creating a new 'Revenue' column in the 'data' DataFrame. It then identifies the product category that generated the highest revenue by grouping the data by 'Description' and summing the revenues. Finally, it prints the product category with 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()
```

code uses the matplotlib and seaborn libraries to create two bar plots for visualizing the distribution of orders in a dataset over days of the week and hours of the day.

Handling Date and Time:

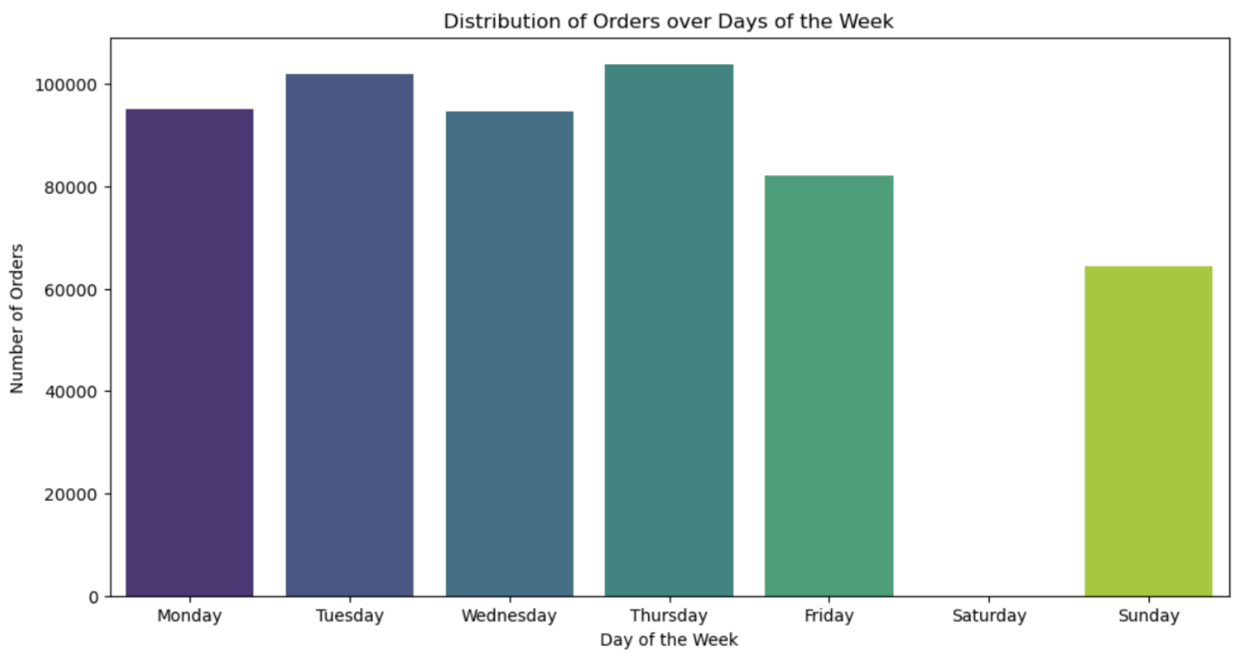
- Converts the 'InvoiceDate' column to datetime format.
- Creates new columns for the day of the week ('DayOfWeek') and hour of the day ('HourOfDay').

Plotting the Distribution of Orders over Days of the Week:

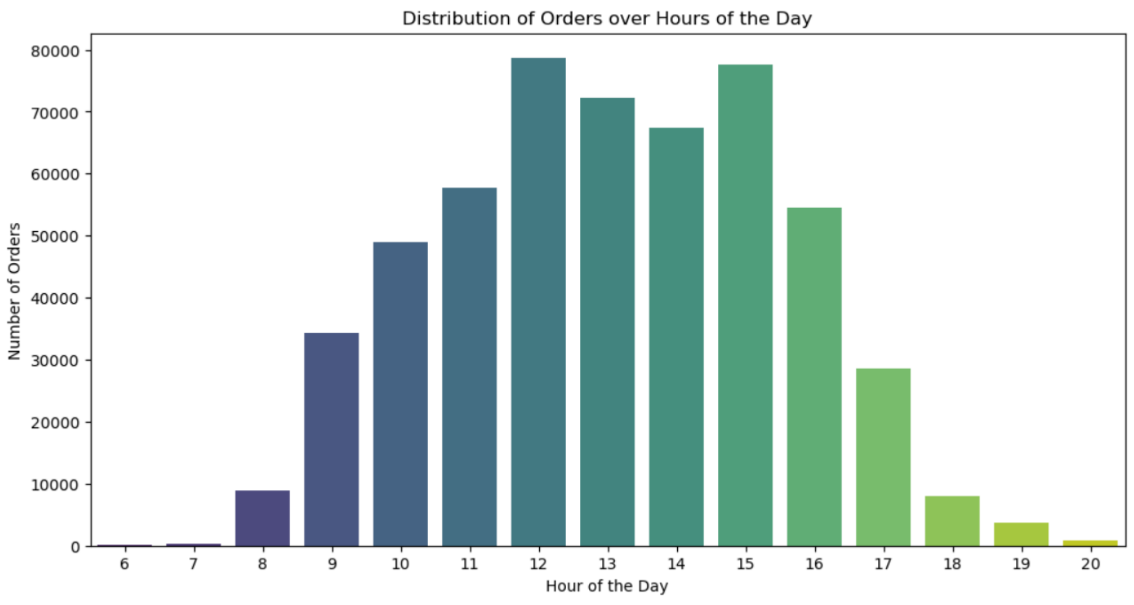
- Creates a bar plot with Seaborn (sns.countplot) to show the number of orders for each day of the week.
- The x-axis represents days of the week, and the y-axis represents the number of orders.

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.



The uploaded image is a bar chart titled "Distribution of Orders over Days of the Week". It shows the number of orders placed on each day of the week from Monday to Sunday. The bars represent the number of orders, with the y-axis indicating the quantity and the x-axis representing the days of the week. Visually, it seems that the number of orders is relatively consistent across the weekdays with a slight decrease towards the weekend, though Sunday appears to have a slightly higher number of orders compared to Saturday. The precise numbers cannot be determined from the image alone without the data values on the y-axis.

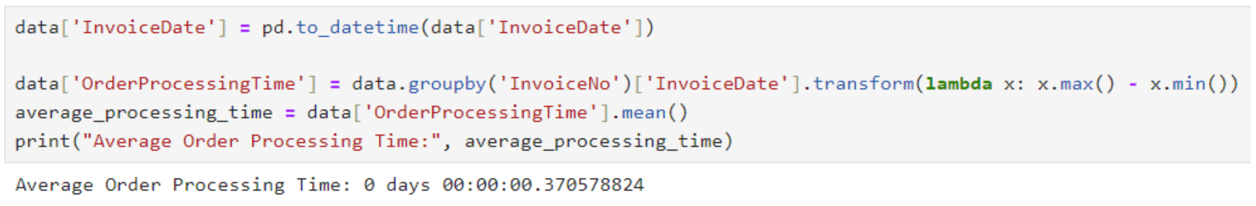


The image shows a bar chart titled "Distribution of Orders over Hours of the Day." It illustrates the number of orders placed at different hours throughout the day, starting from 6 AM to 8 PM (20:00 hours). The y-axis represents the number of orders, while the x-axis represents the hour of the day.

From the visual, there's a notable increase in the number of orders from 6 AM, with a peak around 3 PM (15:00 hours). After this peak, there's a sharp decline in the number of orders as the day progresses towards the evening, with the least orders placed around 8 PM. This suggests that the busiest order time is in the mid-afternoon, while early morning and late evening are the quietest.

There is no specific day on which the most of the orders are placed but, the highest number of orders seem to have been placed on thursdays and around noon time during most days.

b. What is the average order processing time?

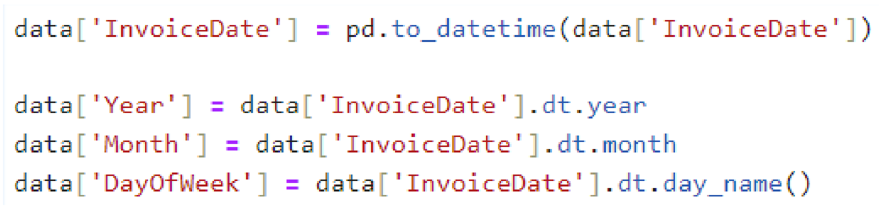


```python
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

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
```

code first converts the 'InvoiceDate' column in the DataFrame data to datetime format. Then, it calculates the order processing time for each invoice by finding the time difference between the maximum and minimum 'InvoiceDate' within each group of 'InvoiceNo'. Finally, it calculates and prints the average order processing time across all invoices.

c. Are there any seasonal trends in the dataset?

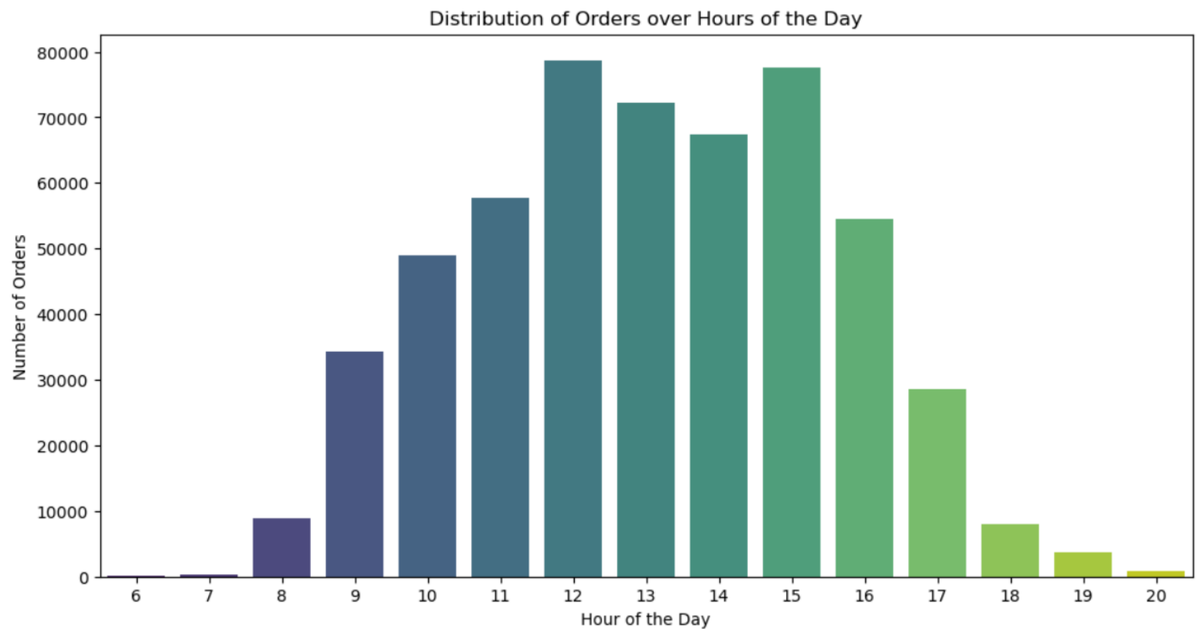


```python
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

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

code converts the 'InvoiceDate' column in the DataFrame data to datetime format and then extracts and creates new columns for the year, month, and day of the week from the 'InvoiceDate'.

```
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()
```

generates a Seaborn countplot to visualize the distribution of orders over different hours of the day. The resulting bar plot shows the number of orders for each hour, with the x-axis representing hours, and it includes a title and axis labels for clarity. The plot is displayed in a figure with a specified size of 12 by 6 inches.



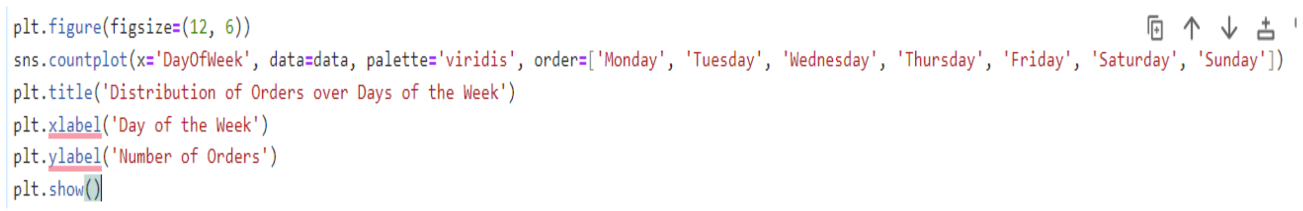
The image shows a bar chart titled "Distribution of Orders over Hours of the Day." It depicts the number of orders placed during various hours from early morning (6 AM) to evening (8 PM). The y-axis indicates the number of orders, and the x-axis represents the hours of the day. Observations from the chart are:

- The number of orders starts to rise from 6 AM, with a more significant increase at 10 AM.

- There is a peak in order volume at 3 PM (15:00), which is the highest point on the chart.

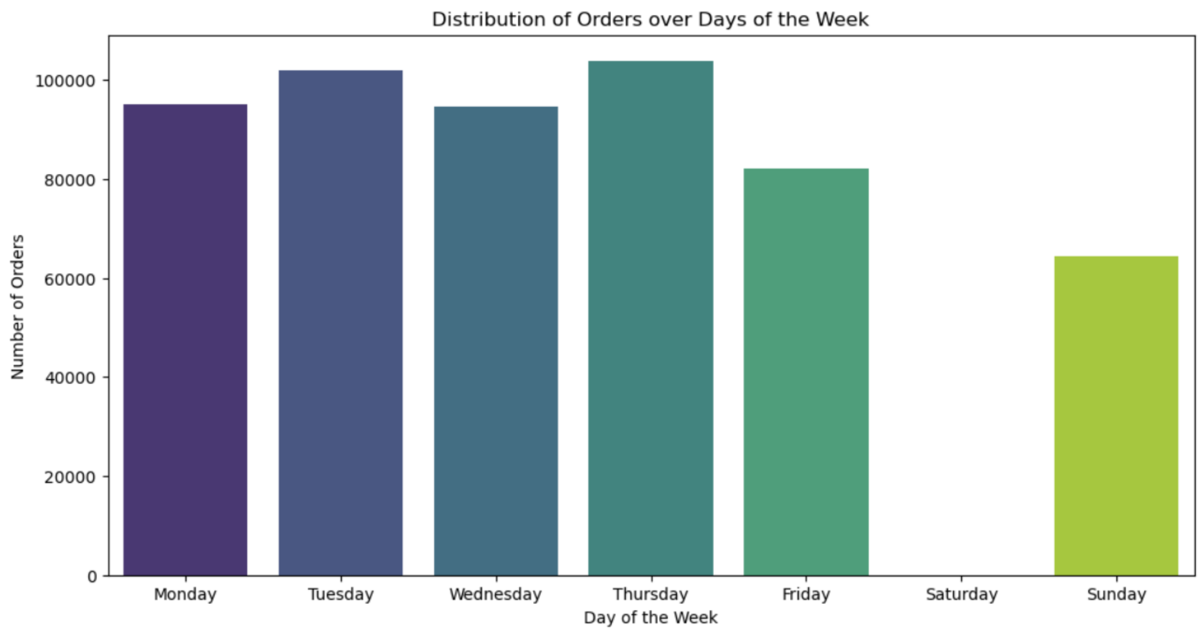
- After 3 PM, there is a sharp decline in the number of orders, with the fewest orders occurring after 6 PM (18:00).

This distribution suggests that the busiest time for placing orders is in the mid-afternoon, and activity significantly drops off in the late afternoon to evening.

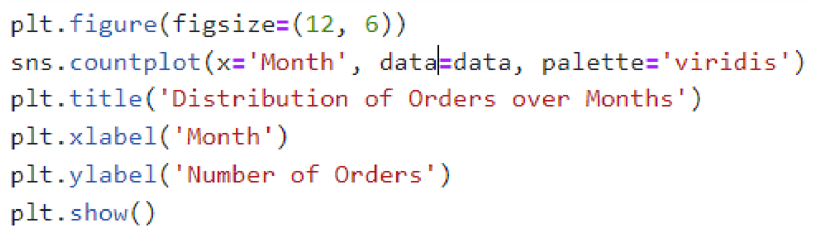


```python
plt.figure(figsize=(12, 6))
sns.countplot(x='DayOfWeek', data=data, palette='viridis', order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.title('Distribution of Orders over Days of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Orders')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

code generates a Seaborn countplot to illustrate the distribution of orders across different days of the week. The resulting bar plot displays the number of orders for each day, with the x-axis representing days of the week. The plot includes a specified color palette ('viridis'), a custom order of days, and labels for the title, x-axis, and y-axis. The plot is displayed in a figure with a size of 12 by 6 inches.

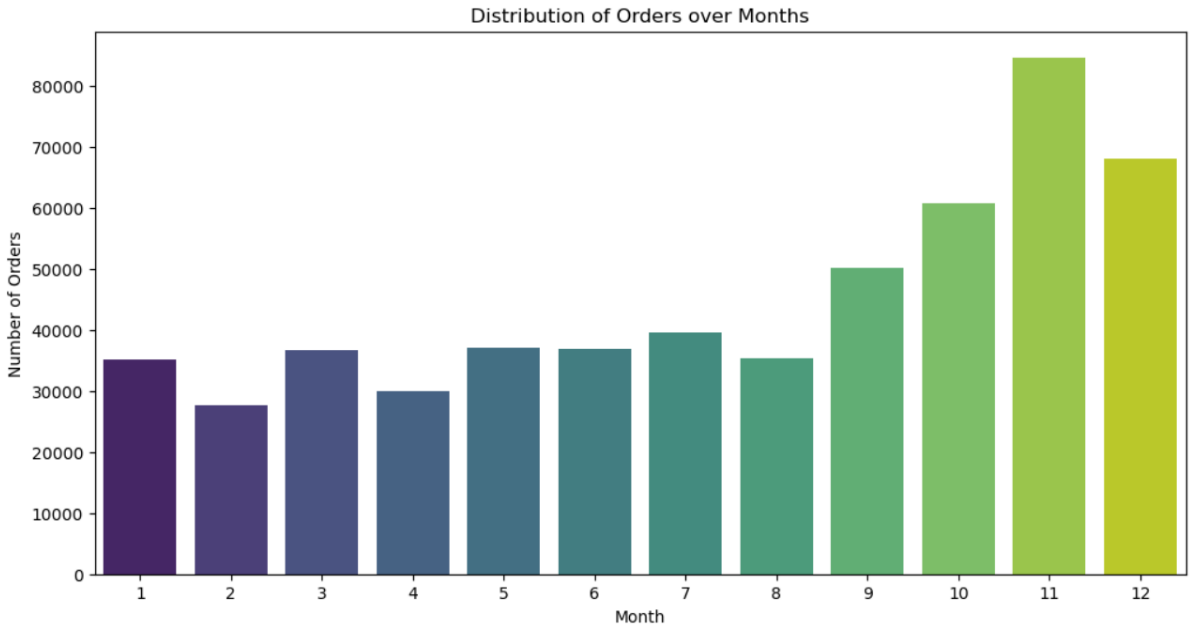


The image displays a bar chart titled "Distribution of Orders over Days of the Week." It represents the number of orders placed on each day, from Monday to Sunday. The y-axis quantifies the number of orders, while the x-axis lists the days of the week. Visually, it appears that the number of orders is relatively high on weekdays, with a slight dip on Friday. Saturday shows a more significant drop in the number of orders, but there is an increase again on Sunday. The exact number of orders for each day is not visible, but the pattern suggests a fluctuation in order volume throughout the week.



```python
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()
```

code creates a Seaborn countplot to visualize the distribution of orders over different months. The resulting bar plot displays the number of orders for each month, with the x-axis representing the months. The plot uses the 'viridis' color palette, and it includes a title, x-axis label ('Month'), y-axis label ('Number of Orders'), and is presented in a figure with a size of 12 by 6 inches.



The image presents a bar chart titled "Distribution of Orders over Months." It shows the number of orders for each month, labeled from 1 to 12, corresponding to January through December. The y-axis measures the number of orders, and the x-axis represents the month.

Key observations from the chart:

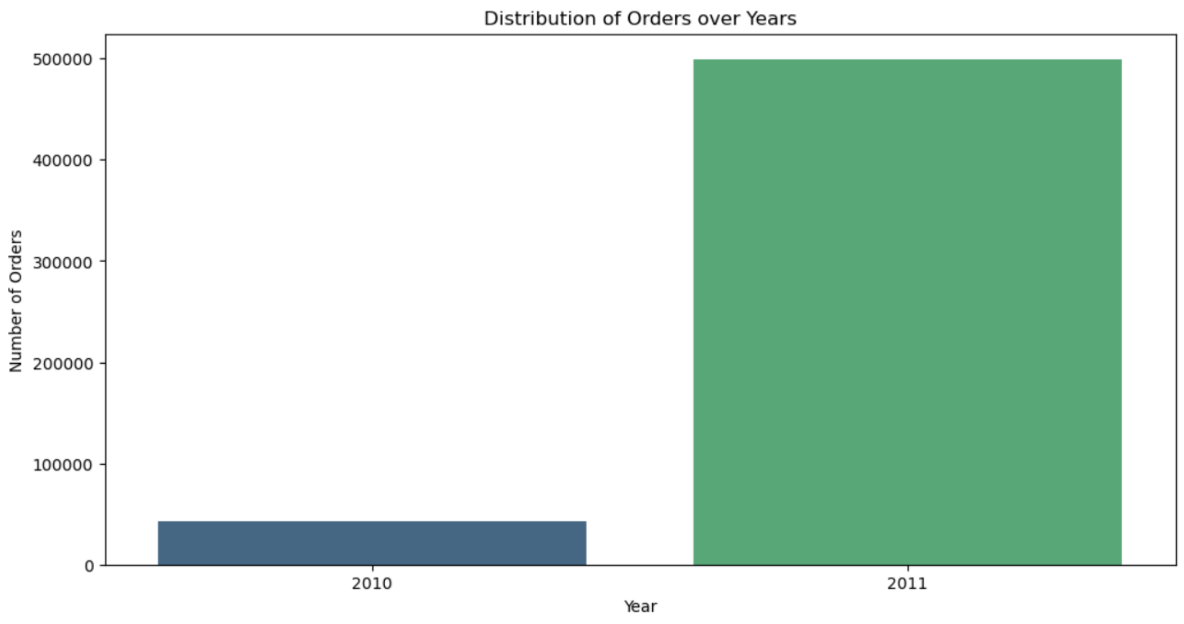
- The number of orders appears to increase progressively from January (Month 1) to December (Month 12), with some fluctuations.

- There is a significant increase in the number of orders in the last quarter of the year, with December having the most orders.

- This pattern might suggest seasonal trends in purchasing behavior, with a possible peak 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()
```

code utilizes Seaborn to create a countplot, illustrating the distribution of orders over different years. The resulting bar plot displays the number of orders for each year, with the x-axis representing the years. The plot uses the 'viridis' color palette, and it includes a title, x-axis label ('Year'), y-axis label ('Number of Orders'), and is presented in a figure with a size of 12 by 6 inches.

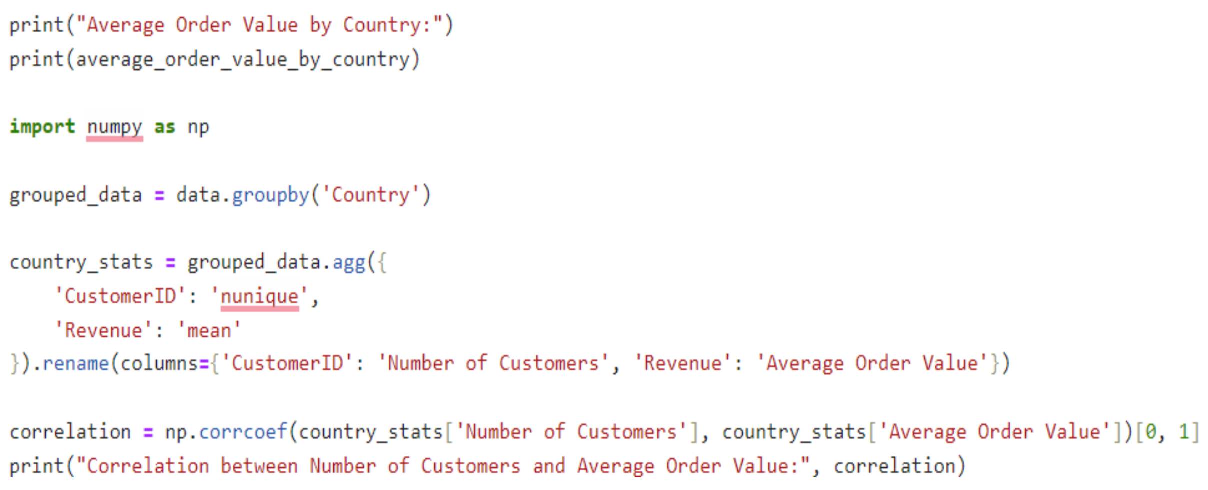


The image displays a bar chart titled "Distribution of Orders over Years." It compares the number of orders between two years, 2010 and 2011. The y-axis indicates the number of orders, while the x-axis specifies the years. From the chart, it's evident that there's a substantial increase in the number of orders from 2010 to 2011. The bar for 2010 shows significantly fewer orders compared to 2011, where the number of orders has increased dramatically, indicating growth or a significant change in order volume between the two years.

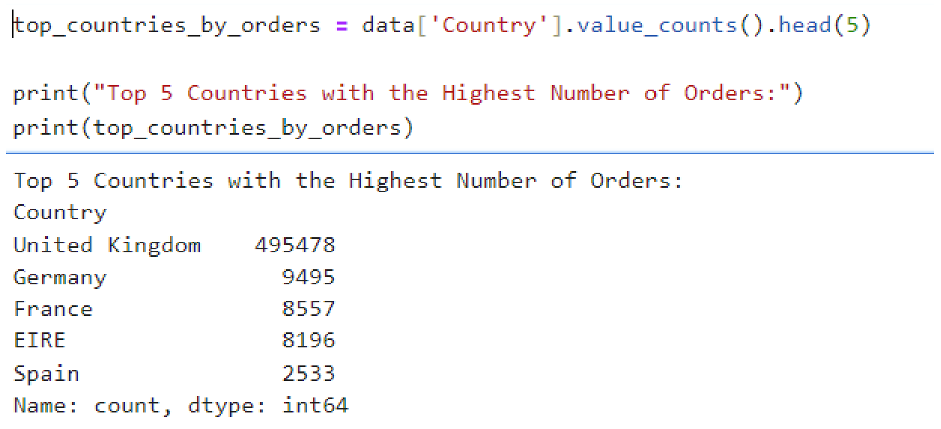
We can see that there is a huge rise in the number of orders placed from the year 2010 to 2011. We can also observe from the plots that the number of orders placed has risen as it nears the end of every year. We can clearly see that the number of orders on saturdays are very less compared to other days. Very less number of orders placed during night times on all days.

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)
```



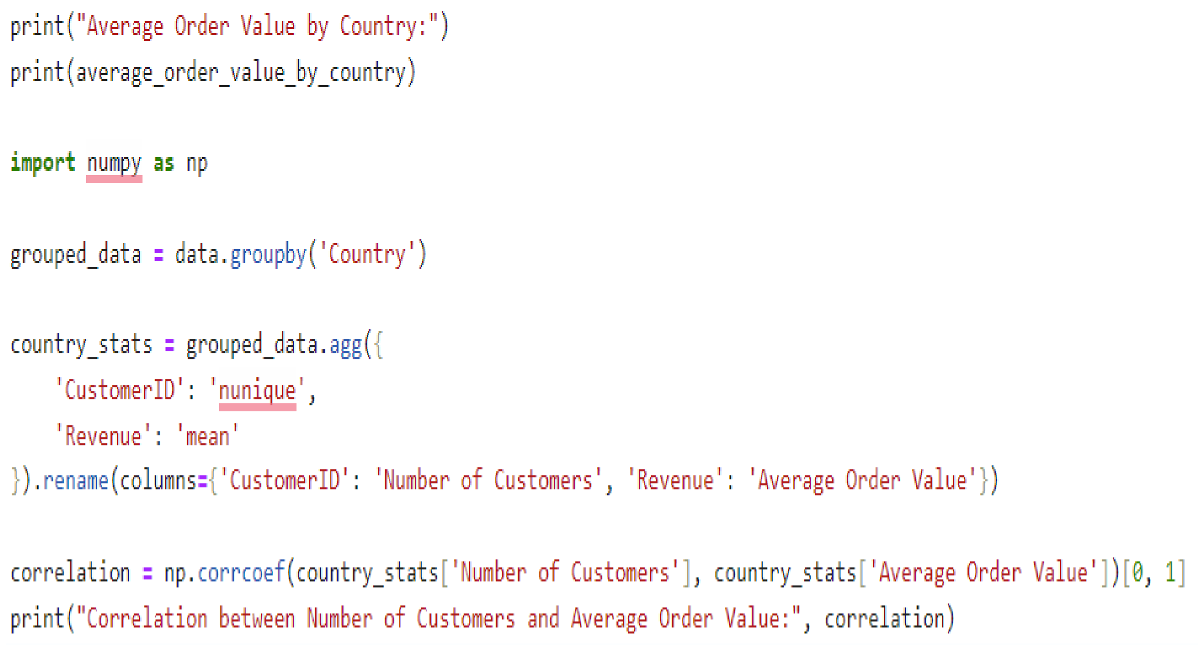
```
top_countries_by_orders = data['Country'].value_counts().head(5)

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
```

code calculates and prints the top 5 countries with the highest number of orders in the DataFrame data. It uses the 'value_counts()' method on the 'Country' column to count the occurrences of each country and then selects the top 5 countries based on order frequency. The result is printed, showing the top countries and their corresponding order counts.

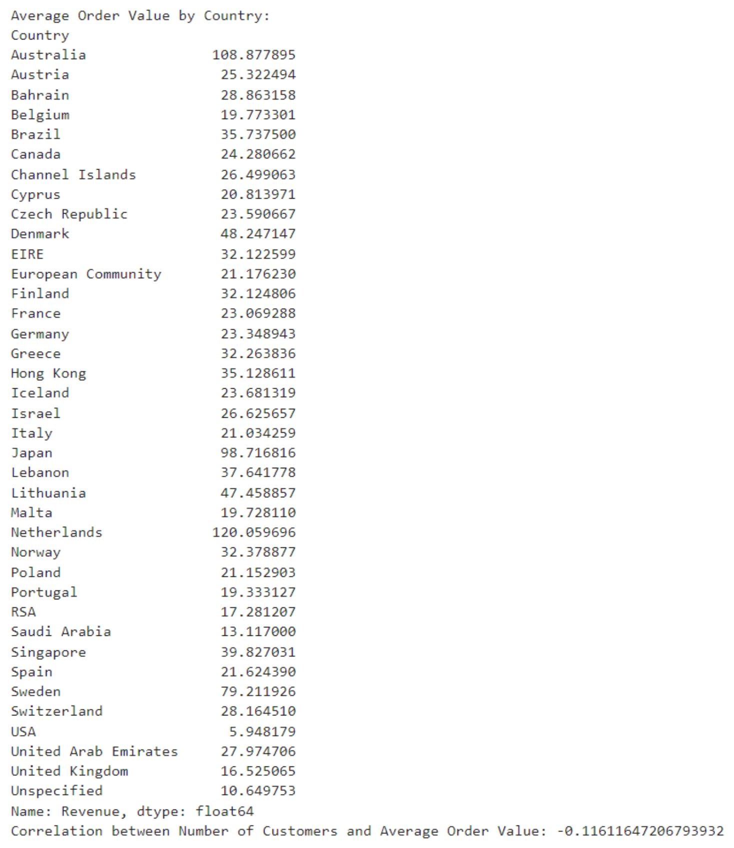
The output represents the top 5 countries with the highest number of orders from the dataset.

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



```python
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)
```

code calculates and prints the average order value for each country in the dataset. Additionally, it calculates the correlation between the number of unique customers and the average order value for each country. The code provides insights into how the average order value relates to the number of customers in each country and prints the correlation between these two aspects.



```
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 contains a text-based list that provides information on the average order value by country. It seems to be output from a data analysis program, with countries listed alongside their corresponding average order value. The values are likely in a currency unit, but the unit is not specified. Additionally, at the bottom of the list, there's a statement about the correlation between the number of customers and the average order value, which is approximately -0.116.

Here are some highlights from the list:

- Australia has the highest average order value listed, at 108.877895.

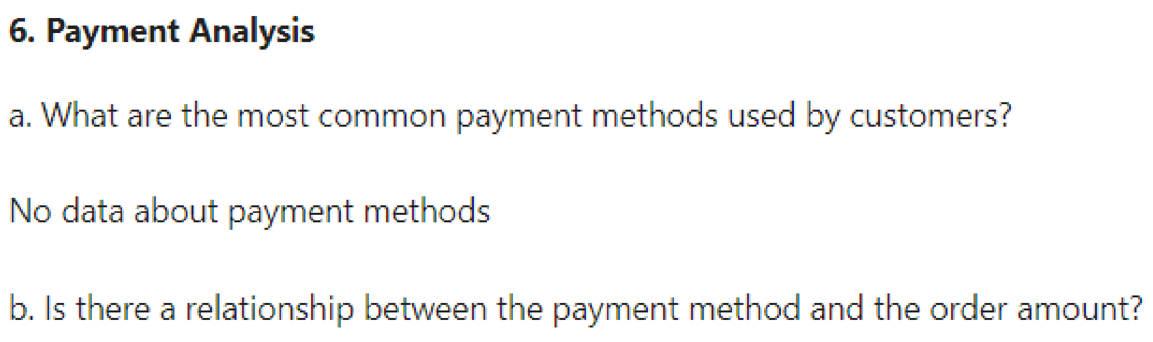
- The USA has one of the lowest average order values, at 5.948179.

- The Netherlands has a notably high average order value as well, at 120.059696.

- There is a negative correlation between the number of customers and the average order value, suggesting that countries with more customers might have a lower average order value, or vice versa.

This data could be used to inform business strategies such as marketing and pricing in different international markets.

6. Payment Analysis



6. Payment Analysis

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

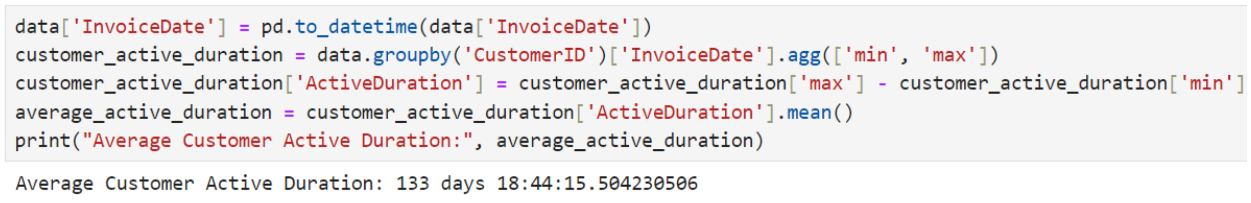
No data about payment methods

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)?

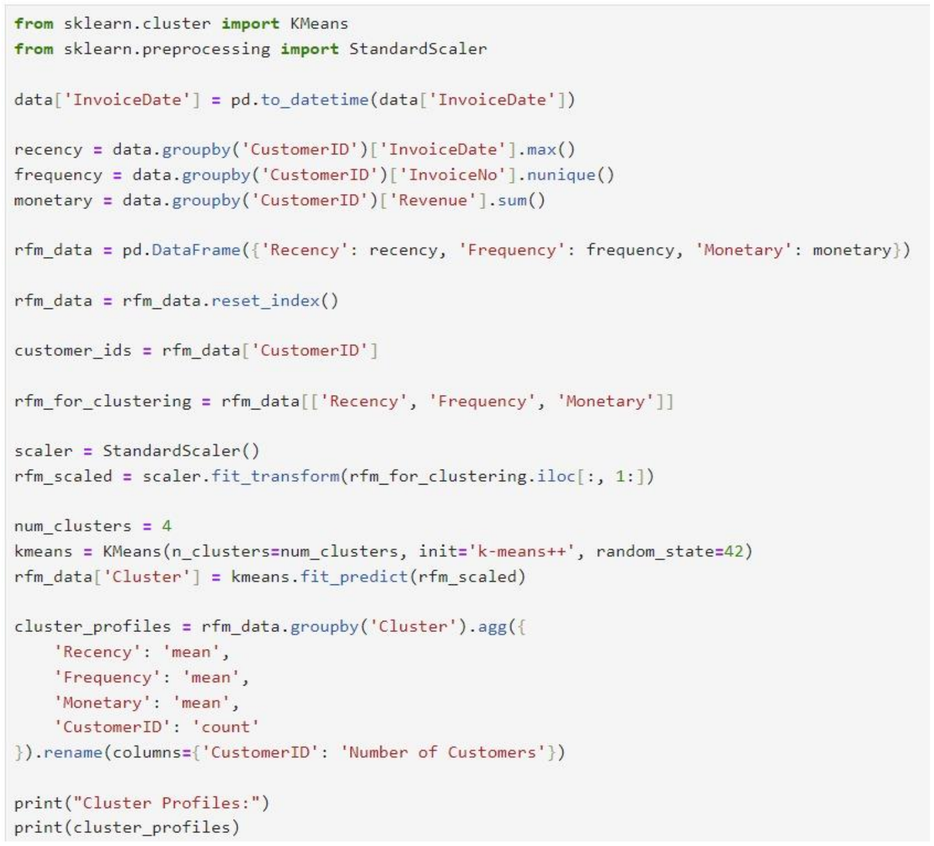


```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
customer_active_duration = data.groupby('CustomerID')['InvoiceDate'].agg(['min', 'max'])
customer_active_duration['ActiveDuration'] = customer_active_duration['max'] - customer_active_duration['min']
average_active_duration = customer_active_duration['ActiveDuration'].mean()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

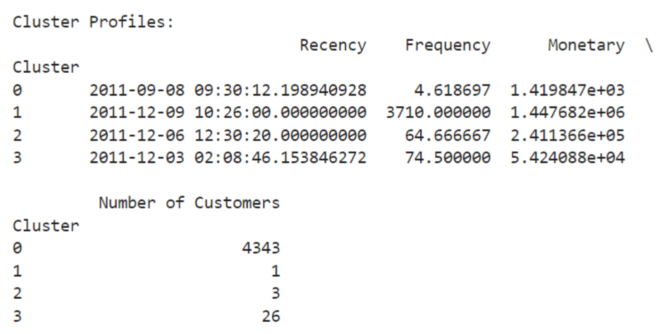
```
Average Customer Active Duration: 133 days 18:44:15.504230506
```

This code figures out how long customers stay active by looking at the time between their first and last purchases. It calculates the average duration of customer engagement, telling us, on average, how much time passes from a customer's first transaction to their last. The result gives a sense of the typical duration customers remain active with the business.

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



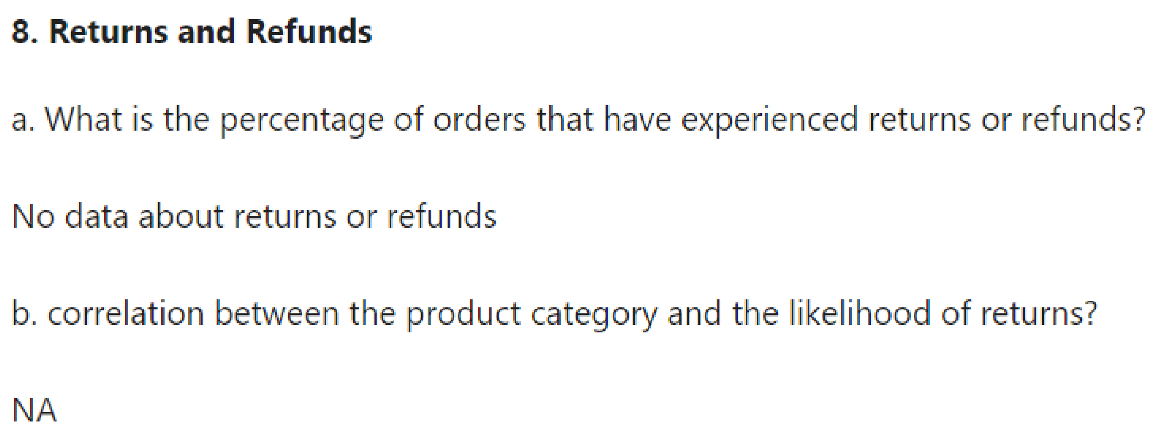
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

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 performs k-means clustering on customer data to create distinct clusters based on Recency, Frequency, and Monetary values. It first preprocesses the data by converting dates, extracting relevant features, and standardizing them. The code then applies k-means clustering with four clusters and assigns each customer to a cluster. Finally, it prints cluster profiles, showing the average Recency, Frequency, and Monetary values along with the number of customers in each cluster.

8. Returns and Refunds



8. Returns and Refunds

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

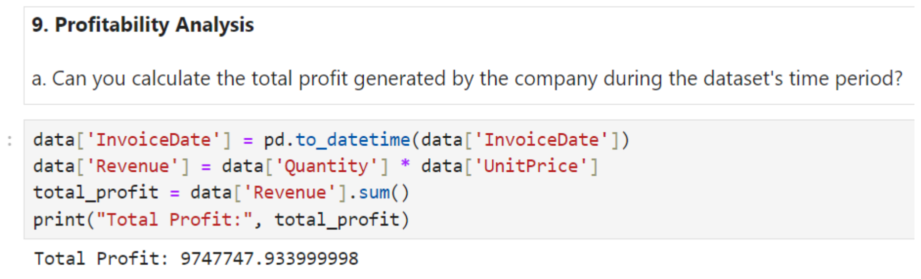
No data about returns or refunds

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



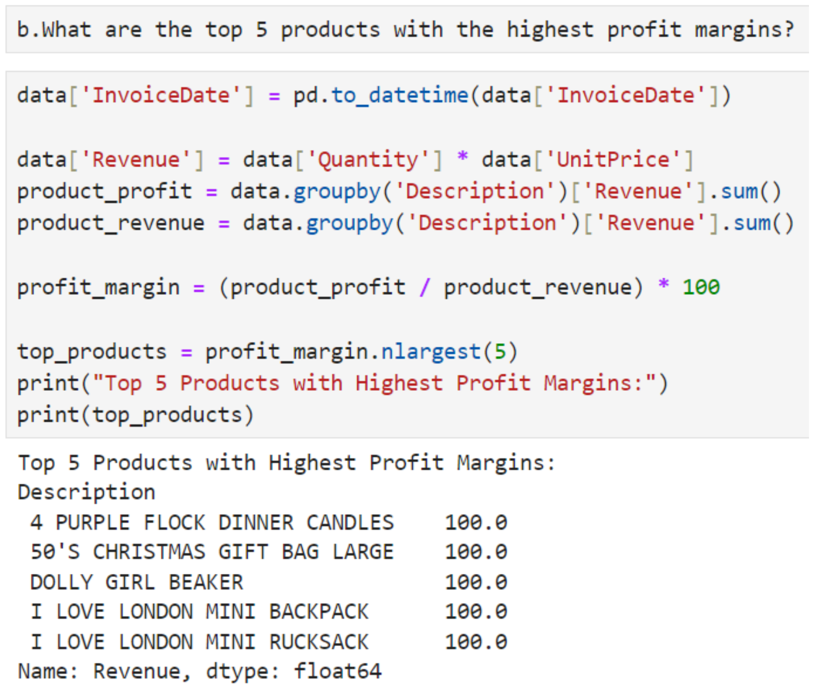
9. Profitability Analysis

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 figures out how much money was made in total from all the transactions. It does this by multiplying the quantity of items sold by their unit prices for each transaction, adding up these values, and then printing the total profit.



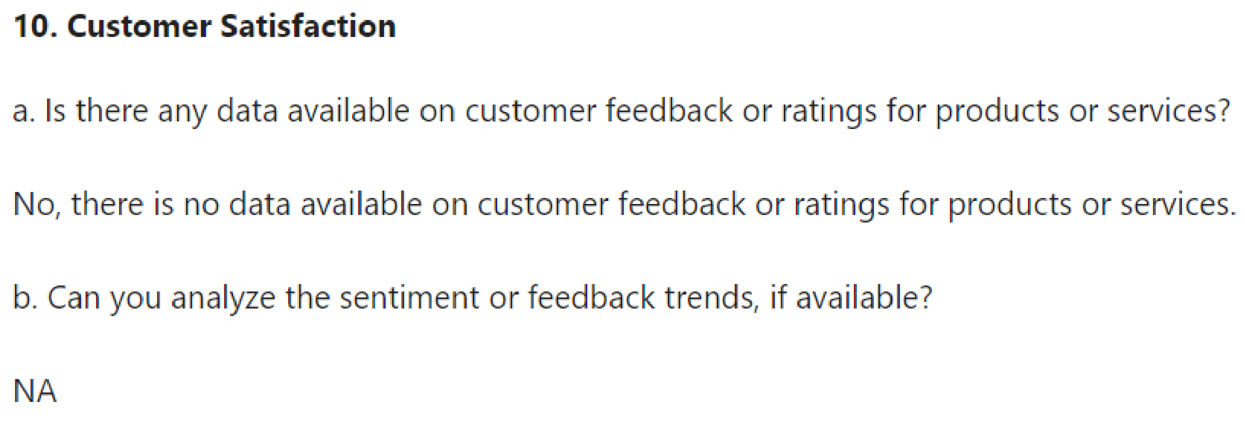
b.What are the top 5 products with the highest profit margins?

```
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 finds the top 5 products with the best profit margins in the dataset. It calculates the profit margin for each product, which is the percentage of revenue retained as profit. The products with the highest profit margins are then identified and printed, giving insight into the most financially successful items 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