



Project 2

Topic: Customer Segmentation using RFM Analysis

Group 19

Aditya Kumar
Nishant Upadhyay
Priyansh Nileshbhai Vagadia

kumar.aditya1@northeastern.edu

upadhyay.nish@northeastern.edu

vagadiya.p@northeastern.edu

Project Report: Customer Segmentation using RFM Analysis

Objective

The aim of this project is to perform RFM analysis on the dataset and segment the customers into distinct groups based on their RFM scores. These segments will provide valuable insights for marketing and customer retention strategies.

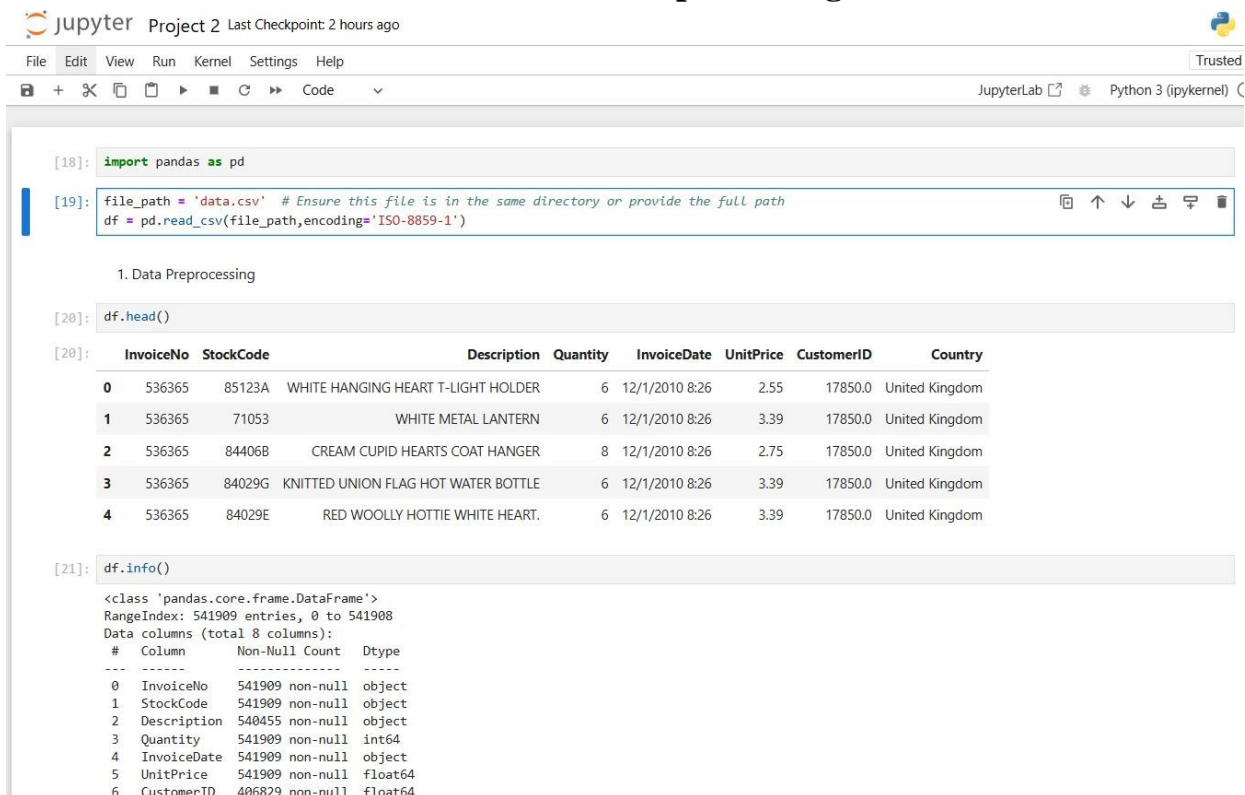
Importing Libraries:

This project used a number of primary Python libraries on the exercise of data analysis and visualization. The pandas library is used in data manipulation and preprocessing for efficient loading, cleaning, and exploration of the crime dataset. NumPy was used to perform a few data computations and transformation of information within the dataset. Visualization was done by utilizing Matplotlib and Seaborn; the former to plot the core plots, while the latter added style to these plots.

We used the publicly available data on ecommerce from Kaggle

<https://www.kaggle.com/datasets/carrie1/ecommerce-data>

Task 1: Data Preprocessing



```
[18]: import pandas as pd

[19]: file_path = 'data.csv' # Ensure this file is in the same directory or provide the full path
df = pd.read_csv(file_path, encoding='ISO-8859-1')
```

1. Data Preprocessing

```
[20]: df.head()
```

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

```
[21]: df.info()

<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      540455 non-null object
3   Quantity        541909 non-null int64
4   InvoiceDate      541909 non-null object
5   UnitPrice       541909 non-null float64
6   CustomerID      406829 non-null float64
```

```
Jupyter Project 2 Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help Trusted
+ - X Copy Paste Undo Redo Code

[22]: df.isnull().sum()

[22]: InvoiceID      0
      StockCode    0
      Description 1454
      Quantity     0
      InvoiceDate   0
      UnitPrice    0
      CustomerID 135080
      Country      0
      dtype: int64

[25]: df.describe()

[25]:
```

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

```


[26]: count = df['Description'].str.contains(r'\?\\?', na=False).sum()
      print(count)
      15

[27]: df = df[df['UnitPrice'] > 0]

[28]: df = df.dropna(subset=['CustomerID'])

[29]: df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
      df['CustomerID'] = df['CustomerID'].astype(int)

[30]: print("\nDataset Period (based on 'InvoiceDate'):")
      start_date = df['InvoiceDate'].min()
      end_date = df['InvoiceDate'].max()

      print("Start Date:", start_date)
      print("End Date:", end_date)

      Dataset Period (based on 'InvoiceDate'):
      Start Date: 2010-12-01 08:26:00
      End Date: 2011-12-09 12:50:00
```

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

Task 2: RFM Calculation

2. RFM Calculation

```
[31]: # Step 2: Convert InvoiceDate to datetime
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], errors='coerce')

# Step 3: Create the Monetary Value (Total Price for each transaction)
df['TotalPrice'] = df['Quantity'] * df['UnitPrice']

# Step 4: Calculate Recency, Frequency, and Monetary metrics

# Recency: The number of days since the Last purchase for each customer
# We will use the max date in the dataset as the reference point
current_date = df['InvoiceDate'].max() # Latest date in the dataset
recency_df = df.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (current_date - x.max()).days # Calculate days since Last purchase
}).reset_index()
recency_df.rename(columns={'InvoiceDate': 'Recency'}, inplace=True)

# Frequency: The total number of unique purchases (invoices) per customer
frequency_df = df.groupby('CustomerID').agg({
    'InvoiceNo': 'nunique' # Count unique invoices for each customer
}).reset_index()
frequency_df.rename(columns={'InvoiceNo': 'Frequency'}, inplace=True)

# Monetary: The total monetary value spent by each customer
monetary_df = df.groupby('CustomerID').agg({
    'TotalPrice': 'sum' # Sum of total price per customer
}).reset_index()
monetary_df.rename(columns={'TotalPrice': 'Monetary'}, inplace=True)

# Step 5: Merge the three dataframes to get the final RFM table
rfm_df = recency_df.merge(frequency_df, on='CustomerID', how='left')
rfm_df = rfm_df.merge(monetary_df, on='CustomerID', how='left')

# Step 6: Remove decimal points from CustomerID (convert to integers)
rfm_df['CustomerID'] = rfm_df['CustomerID'].astype(int)

# Step 7: Display the RFM dataframe
print(rfm_df.head())
```

	CustomerID	Recency	Frequency	Monetary
0	12346	325	2	0.00
1	12347	1	7	4310.00
2	12348	74	4	1797.24
3	12349	18	1	1757.55
4	12350	309	1	334.40

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

3. RFM Segmentation:

```
[33]: # Recency score: Lower recency values (more recent purchases) are better
recency_quartiles = rfm_df['Recency'].quantile([0.25, 0.5, 0.75])
rfm_df['RecencyScore'] = rfm_df['Recency'].apply(
    lambda x: 4 if x <= recency_quartiles[0.25] else
              3 if x <= recency_quartiles[0.50] else
              2 if x <= recency_quartiles[0.75] else 1
)

# Frequency score: Higher frequency values (more purchases) are better
frequency_quartiles = rfm_df['Frequency'].quantile([0.25, 0.5, 0.75])
rfm_df['FrequencyScore'] = rfm_df['Frequency'].apply(
    lambda x: 1 if x <= frequency_quartiles[0.25] else
              2 if x <= frequency_quartiles[0.50] else
              3 if x <= frequency_quartiles[0.75] else 4
)

# Monetary score: Higher monetary values (more spending) are better
monetary_quartiles = rfm_df['Monetary'].quantile([0.25, 0.5, 0.75])
rfm_df['MonetaryScore'] = rfm_df['Monetary'].apply(
    lambda x: 1 if x <= monetary_quartiles[0.25] else
              2 if x <= monetary_quartiles[0.50] else
              3 if x <= monetary_quartiles[0.75] else 4
)

# Combine RFM scores into a single RFM score
rfm_df['RFMScore'] = (
    rfm_df['RecencyScore'].astype(str) +
    rfm_df['FrequencyScore'].astype(str) +
    rfm_df['MonetaryScore'].astype(str)
)

# Display the updated RFM dataframe
print(rfm_df.head())
```

	CustomerID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryScore	RFMScore
0	12346	325	2	0.00	1	2	1	121
1	12347	1	7	4310.00	4	4	4	444
2	12348	74	4	1797.24	2	3	4	234
3	12349	18	1	1757.55	3	1	4	314
4	12350	309	1	334.40	1	1	2	112

```
[34]: # Combine RFM scores into a single RFM score
rfm_df['RFMScore'] = (
    rfm_df['RecencyScore'].astype(str) +
    rfm_df['FrequencyScore'].astype(str) +
    rfm_df['MonetaryScore'].astype(str)
)

# Display the updated RFM dataframe with the combined RFM score
print(rfm_df[['CustomerID', 'RecencyScore', 'FrequencyScore', 'MonetaryScore', 'RFMScore']].head())
```

	CustomerID	RecencyScore	FrequencyScore	MonetaryScore	RFMScore
0	12346	1	2	1	121
1	12347	4	4	4	444
2	12348	2	3	4	234
3	12349	3	1	4	314
4	12350	1	1	2	112

Task 4: Customer Segmentation

4. Customer Segmentation:

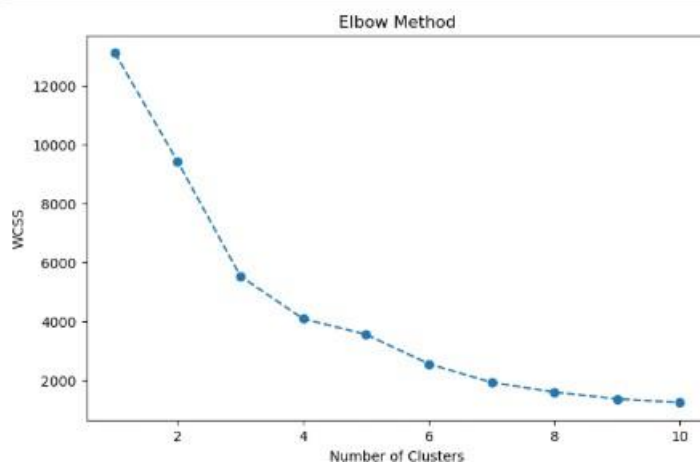
```
[35]: from sklearn.preprocessing import StandardScaler
      from sklearn.cluster import KMeans
      import matplotlib.pyplot as plt
      import seaborn as sns

      # Prepare the data for clustering
      rfm_clustering = rfm_df[['Recency', 'Frequency', 'Monetary']]

      # Standardize the data
      scaler = StandardScaler()
      rfm_scaled = scaler.fit_transform(rfm_clustering)

[36]: wcss = []
      for i in range(1, 11): # Test 1 to 10 clusters
          kmeans = KMeans(n_clusters=i, random_state=42)
          kmeans.fit(rfm_scaled)
          wcss.append(kmeans.inertia_)

      # Plot the Elbow graph
      plt.figure(figsize=(8, 5))
      plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
      plt.title('Elbow Method')
      plt.xlabel('Number of Clusters')
      plt.ylabel('WCSS')
      plt.show()
```



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.

```
[37]: # Apply K-Means clustering
      kmeans = KMeans(n_clusters=4, random_state=42)
      rfm_df['Cluster'] = kmeans.fit_predict(rfm_scaled)

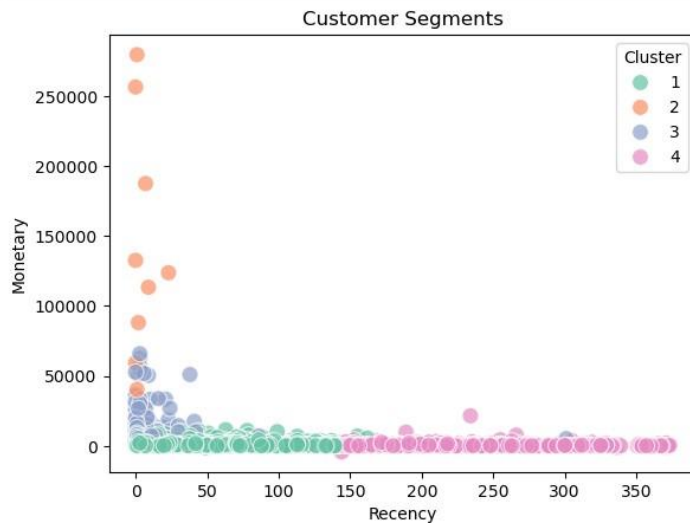
      # Map clusters to 1-based index (optional)
      rfm_df['Cluster'] += 1

[38]: cluster_summary = rfm_df.groupby('Cluster').agg({
      'Recency': 'mean',
      'Frequency': 'mean',
      'Monetary': 'mean',
      'CustomerID': 'count'
      }).rename(columns={'CustomerID': 'CustomerCount'})

print(cluster_summary)
```

	Recency	Frequency	Monetary	CustomerCount
Cluster				
1	41.790223	4.371318	1321.409146	3089
2	4.090909	109.727273	124312.306364	11
3	9.752577	28.510309	12168.264691	194
4	247.927577	1.805942	455.110716	1077


```
[39]: # Visualize clusters in a 2D scatter plot
sns.scatterplot(
    x=rfm_df['Recency'], y=rfm_df['Monetary'],
    hue=rfm_df['Cluster'], palette='Set2', s=100, alpha=0.7
)
plt.title('Customer Segments')
plt.xlabel('Recency')
plt.ylabel('Monetary')
plt.legend(title='Cluster')
plt.show()
```



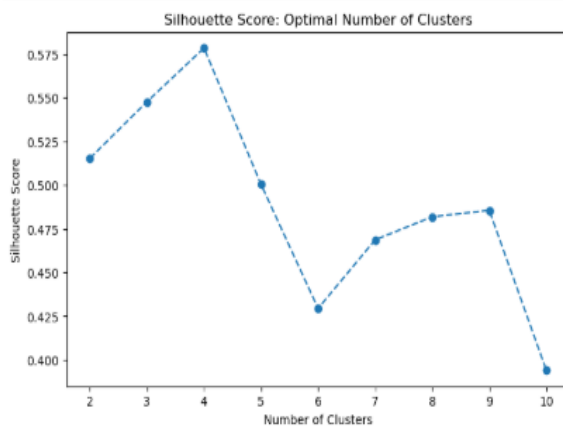
The Silhouette Score was used to evaluate the quality of clusters by measuring how well-separated and cohesive the clusters are, helping to determine the optimal number of clusters.

Refining the choice of clusters using the Silhouette Score.

```
[18]: from sklearn.metrics import silhouette_score

# Compute silhouette scores for a range of cluster numbers
silhouette_scores = []
for i in range(2, 11): # Silhouette score requires at least 2 clusters
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(rfm_scaled)
    score = silhouette_score(rfm_scaled, kmeans.labels_)
    silhouette_scores.append(score)

# Plot the Silhouette scores
plt.figure(figsize=(8, 5))
plt.plot(range(2, 11), silhouette_scores, marker='o', linestyle='--')
plt.title('Silhouette Score: Optimal Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.xticks(range(2, 11))
plt.show()
```



Task 5: Segment Profiling

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.

```
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.

Cluster 1: Active High-Spenders Recency: Mean: 13.1 days, Median: 10 days These customers made recent purchases, indicating they are active. Frequency: Mean: 28.5 transactions, Median: 24 transactions They shop frequently compared to other clusters. Monetary: Mean: 12,110.50, Median: 7,904.28, Total: \$2,361,546.57 They are significant spenders, contributing substantially to total revenue. Customer Count: 195 Profile: Loyal, high-value customers who are both active and frequent buyers. Strategy: Retain this group with exclusive loyalty programs, early product launches, or VIP offers.

Cluster 2: Average Buyers Recency: Mean: 50.6 days, Median: 39 days These customers haven't purchased as recently but aren't inactive yet. Frequency: Mean: 4.4 transactions, Median: 3 transactions Moderate frequency of purchases. Monetary: Mean: 1,322.40, Median: 806.90, Total: \$4,127,220.75 Moderate spenders who contribute a large share of the revenue due to their size. Customer Count: 3,121 Profile: A large segment of average customers who purchase occasionally. Strategy: Engage with personalized offers or seasonal promotions to increase purchase frequency.

Cluster 3: Elite High-Spenders Recency: Mean: 5.3 days, Median: 3 days They have made very recent purchases, showing strong engagement. Frequency: Mean: 109.9 transactions, Median: 77 transactions Exceptionally frequent buyers. Monetary: Mean: 124,312.31, Median: 113,384.14, Total: \$1,367,435.37 They are extremely high spenders and represent a premium group. Customer Count: 11 Profile: Elite, highly engaged customers who generate substantial revenue despite being a small group. Strategy: Offer exclusive perks, one-on-one customer support, and lifetime value incentives to maintain this relationship.

Cluster 4: Inactive Low-Spenders Recency: Mean: 268.7 days, Median: 244 days They haven't made purchases for a long time, indicating they are inactive or churned. Frequency: Mean: 1.6 transactions,

Median: 1 transaction Low purchase frequency, indicating minimal engagement. Monetary: Mean: 424.75, Median: 296.75, Total: \$443,863.12 Low spenders with minimal revenue contribution. Customer Count: 1,045 Profile: A disengaged group of inactive, low-value customers. Strategy: Reactivate with win-back campaigns, discounts, or reminders. If unresponsive, focus resources on other segments.

Task 6: Marketing Recommendations

Here are actionable marketing recommendations for each segment:

Segment 0:

Average Recency: 39.54 days; Average Frequency: 6.12 purchases. Average monetary amount: \$3,127,292.

Recommendations: Within one month, launch targeted email marketing or promotions to drive repeat sales. Offer loyalty programs or incentives to customers who make frequent purchases. Make individualized 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:

The average recency is 0 days (very recent). Average frequency of purchases: 3710 (very frequent) Average monetary amount: \$294,087,800,000 (high spending)

Recommendations: Recognize and reward these high-value consumers with special benefits or loyalty programs. Implement a bespoke VIP service to reward their remarkable commitment. Maintain contact with them with personalized messaging and offers.

General Recommendations

Cross-Sell and Up-Sell: Use cross-selling methods based on complimentary items for all segments. Encourage higher-value purchases by using upselling strategies.

Personalized Communication: Create unique communication channels for each section (e.g., email, SMS, app notifications). Tailor marketing messages to particular client interests and habits.

Customer Feedback and Surveys: Collect feedback to better understand consumer satisfaction and preferences. Use surveys to obtain feedback on prospective upgrades or new services.

Retention Campaigns: Create client retention initiatives for each segment to increase loyalty. Monitor client satisfaction and respond quickly to any problems.

Social Media Engagement: Use social media platforms to interact with customers across several segments. Launch focused social media initiatives to raise brand recognition and loyalty.

Task 7: Visualization

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

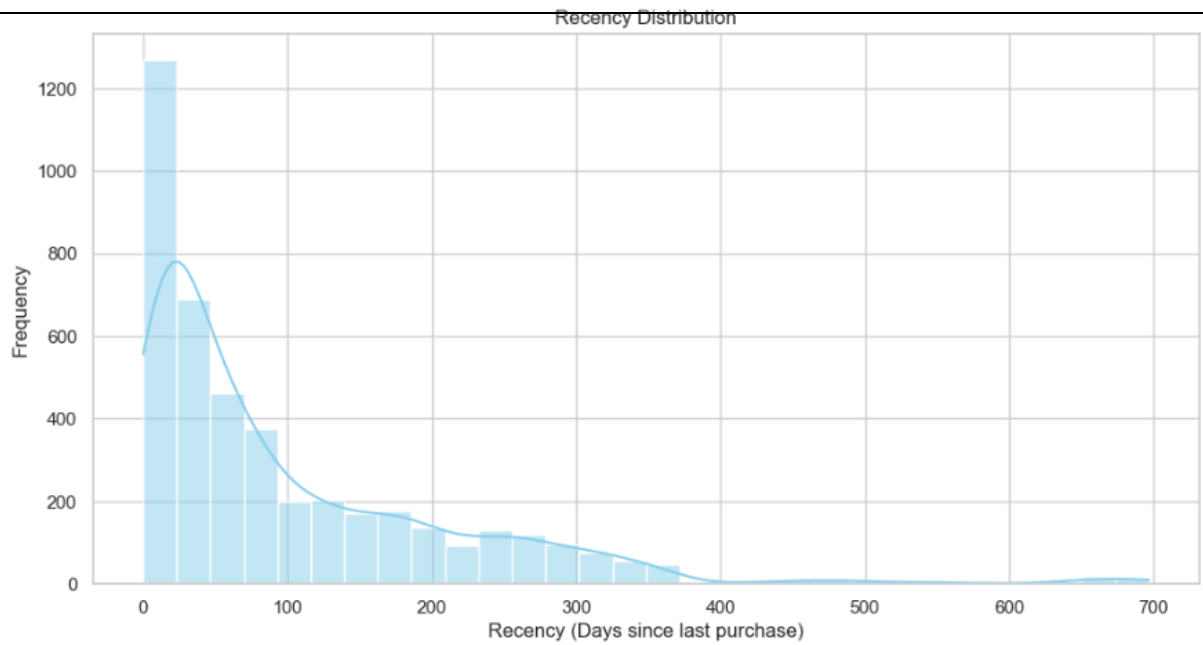
# Set the style for the plots
sns.set(style="whitegrid")

# Plot Recency distribution
plt.figure(figsize=(12, 6))
sns.histplot(rfm_df['Recency'], kde=True, bins=30, color='skyblue')
plt.title('Recency Distribution')
plt.xlabel('Recency (Days since last purchase)')
plt.ylabel('Frequency')
plt.show()

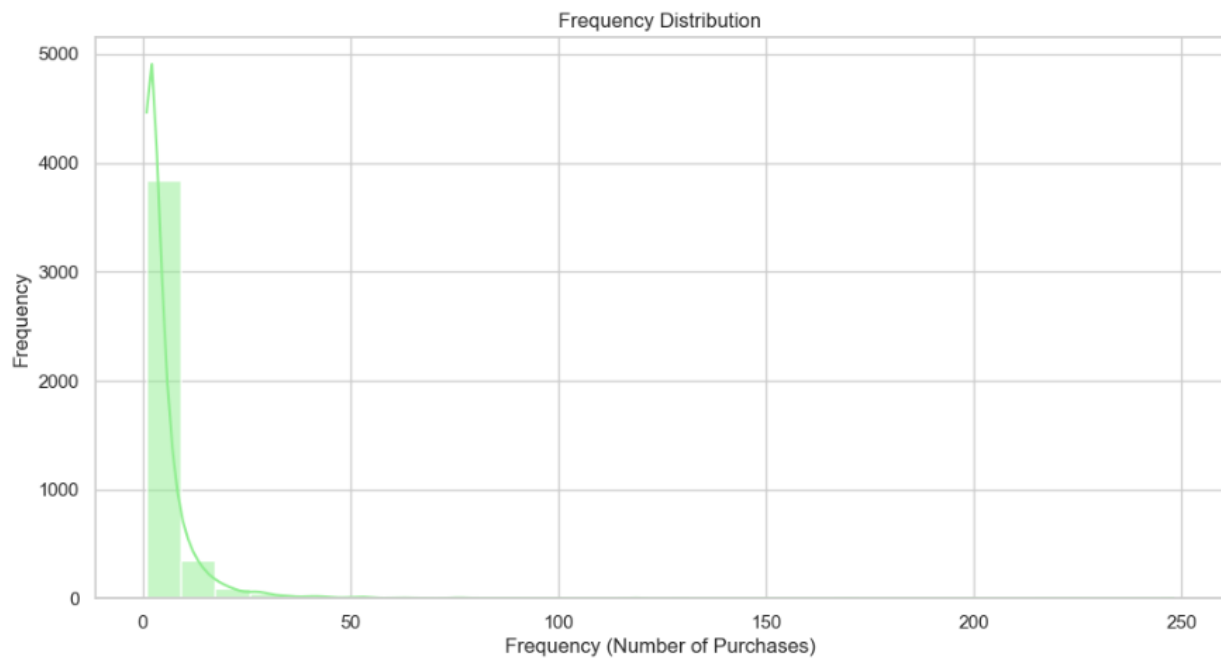
# Plot Frequency distribution
plt.figure(figsize=(12, 6))
sns.histplot(rfm_df['Frequency'], kde=True, bins=30, color='lightgreen')
plt.title('Frequency Distribution')
plt.xlabel('Frequency (Number of Purchases)')
plt.ylabel('Frequency')
plt.show()

# Plot Monetary distribution
plt.figure(figsize=(12, 6))
sns.histplot(rfm_df['Monetary'], kde=True, bins=30, color='salmon')
plt.title('Monetary Distribution')
plt.xlabel('Monetary (Total Spending)')
plt.ylabel('Frequency')
plt.show()
```

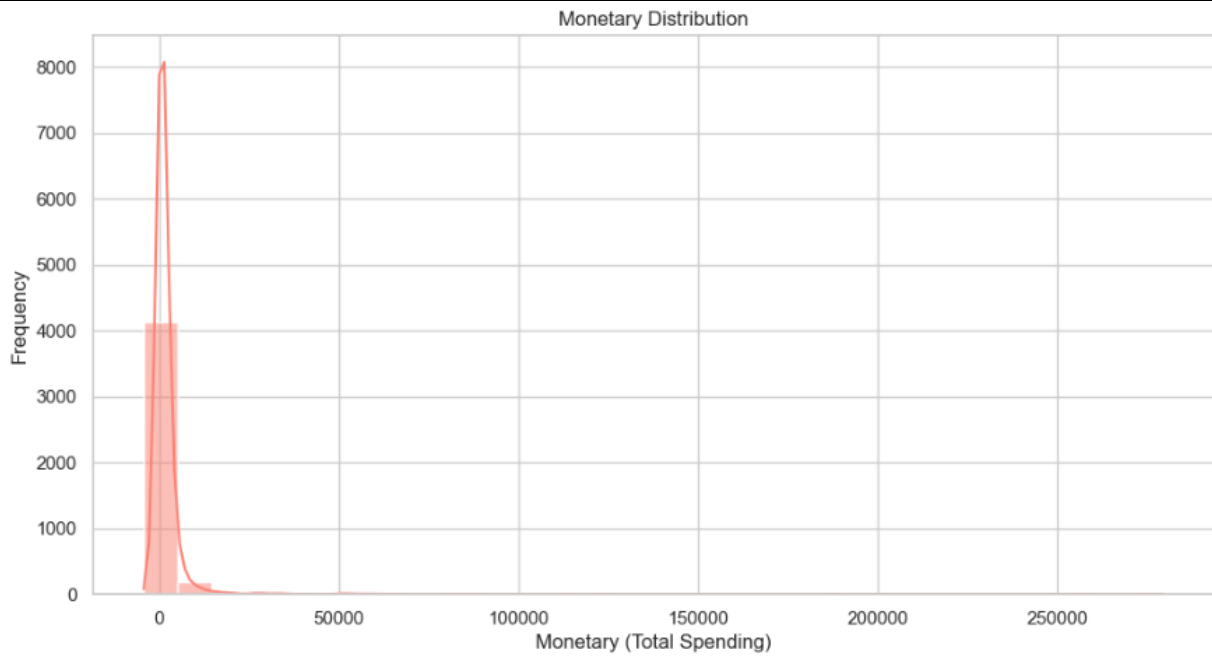
The Recency distribution was plotted to visualize the spread of customer recency (days since last purchase) and identify patterns in recent activity.



The Frequency distribution was plotted to explore how often customers make purchases and identify frequent vs. infrequent buyers.



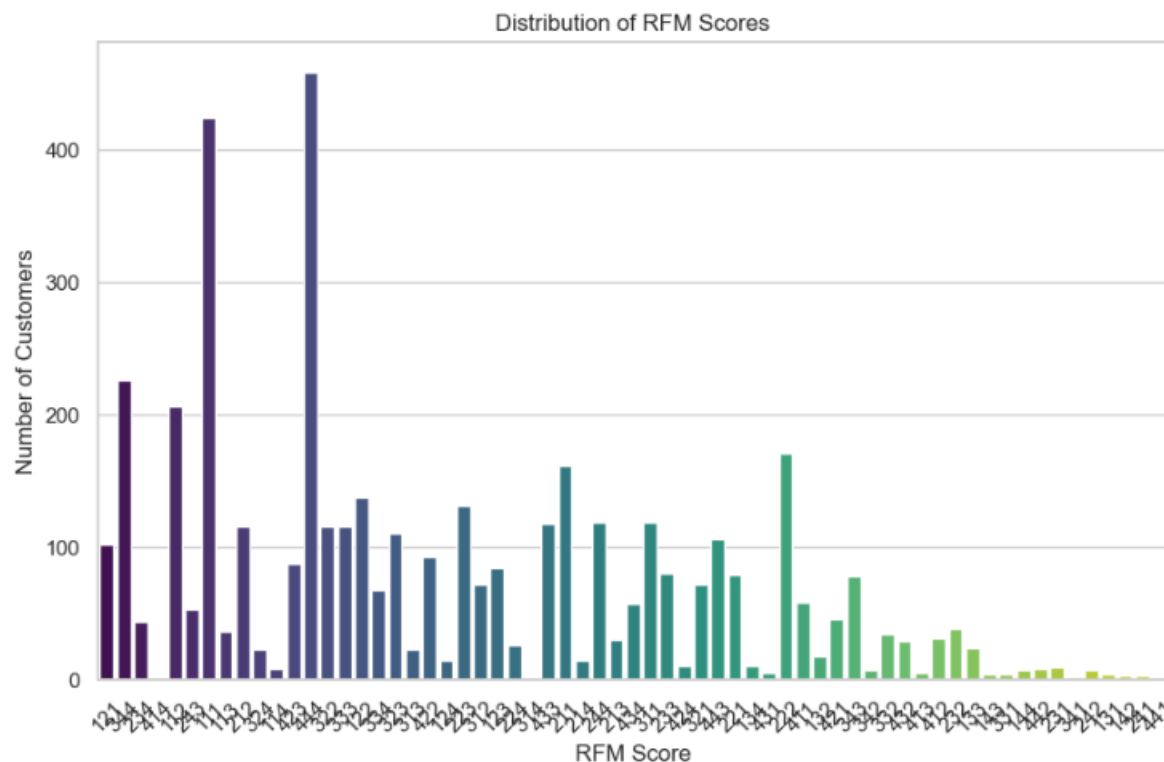
The Monetary distribution was plotted to examine the total spending of customers and understand the spending behavior across the customer base.



```
# Combine RFM Score and plot a bar chart for the distribution of RFM scores
rfm_df['RFMScore'] = rfm_df['RecencyScore'].astype(str) + \
    rfm_df['FrequencyScore'].astype(str) + \
    rfm_df['MonetaryScore'].astype(str)

plt.figure(figsize=(10, 6))
sns.countplot(data=rfm_df, x='RFMScore', palette='viridis')
plt.title('Distribution of RFM Scores')
plt.xlabel('RFM Score')
plt.ylabel('Number of Customers')
plt.xticks(rotation=45)
plt.show()
```

The RFM score was created to combine the Recency, Frequency, and Monetary scores into a single metric, and the bar chart was used to visualize the distribution of these combined RFM scores across customers.

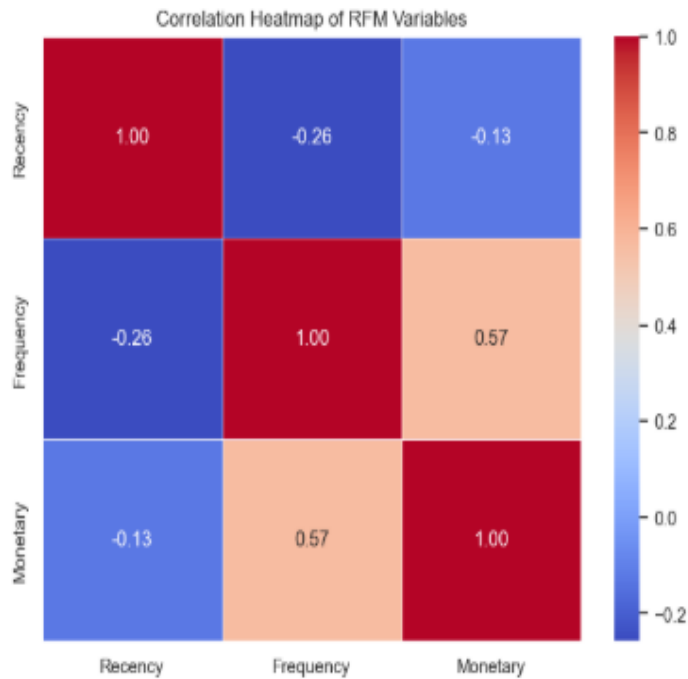


The correlation matrix and heatmap were used to visualize and assess the relationships between the RFM

(Recency, Frequency, Monetary) variables.

```
: # Compute correlation matrix
rfm_corr = rfm_df[['Recency', 'Frequency', 'Monetary']].corr()

# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(rfm_corr, annot=True, cmap='coolwarm', linewidths=0.5, fmt='.2f', cbar=True)
plt.title('Correlation Heatmap of RFM Variables')
plt.show()
```



Questions

1. Data Overview:

Size of dataset in terms of numbers of rows and columns:

```
7]: df.shape
```

a. 7]: (541909, 8)

This code gives the size of the dataset, indicating the number of rows and columns. We could see that there are 541909 records(rows) and 8 features(columns) in the given dataset.

Brief Description of each column in dataset:

```
[8]: df.info()

<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     540455 non-null object  
3   Quantity        541909 non-null int64   
4   InvoiceDate      541909 non-null object  
5   UnitPrice       541909 non-null float64  
6   CustomerID      406829 non-null float64  
7   Country         541909 non-null object  
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

```
[9]: df.describe()
```

```
[9]:
```

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

b.

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. Time period covered by the dataset:

```
[10]: print("\nDataset Period (based on 'InvoiceDate'):")
      start_date = df['InvoiceDate'].min()
      end_date = df['InvoiceDate'].max()

      print("Start Date:", start_date)
      print("End Date:", end_date)
```

```
Dataset Period (based on 'InvoiceDate'):
Start Date: 1/10/2011 10:04
End Date: 9/9/2011 9:52
```

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?

```
2]: print('Number of unique custoemrs: ',df['CustomerID'].nunique())
```

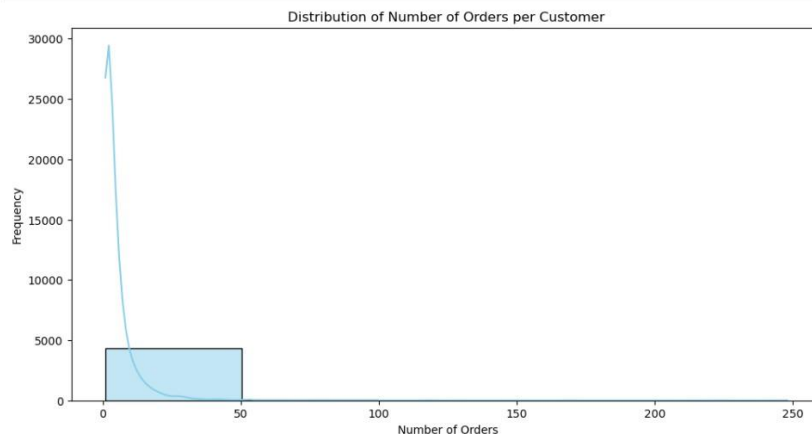
Number of unique custoemrs: 4372

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.

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

```
19]: import matplotlib.pyplot as plt
import seaborn as sns

orders_per_customer = df.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.

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

```
1]: top_customers_by_order_count = df.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

14911.0 248

12748.0 224

17841.0 169

14606.0 128

13089.0 118

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

```
] : top_products_by_frequency = df['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
850998	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.

3.b What is the average size of products in the dataset?

```
avg = df['UnitPrice'].mean()
print('Average Price of Products =', avg)
```

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.

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

```
df['Revenue'] = df['Quantity'] * df['UnitPrice']
top_revenue_category = df.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 by the unit price, resulting in a new 'Revenue' column in the 'data' DataFrame. It then selects the product category with the highest revenue by grouping the data by 'Description' and adding the revenues. Finally, it shows 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

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

df['DayOfWeek'] = df['InvoiceDate'].dt.day_name()
df['HourOfDay'] = df['InvoiceDate'].dt.hour

plt.figure(figsize=(12, 6))
sns.countplot(x='DayOfWeek', data=df, 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 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:

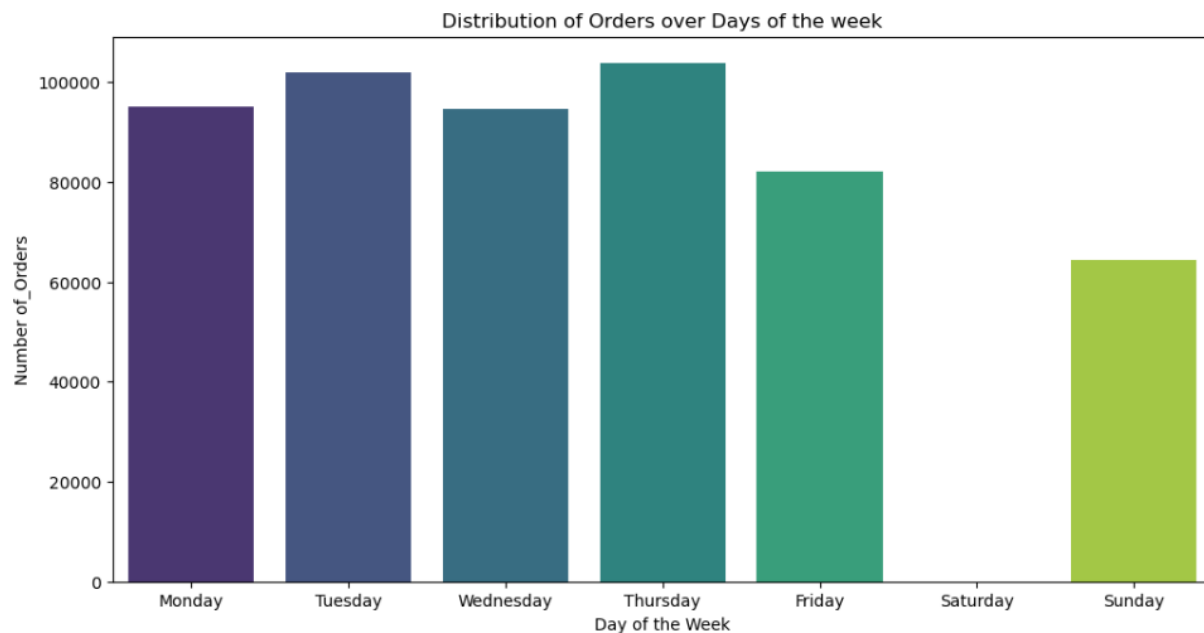
- Converts the 'InvoiceDate' column to datetime format.
- Creates new columns for the day of the week ('DayOfWeek') and hour of the day ('HourOfDay').

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.

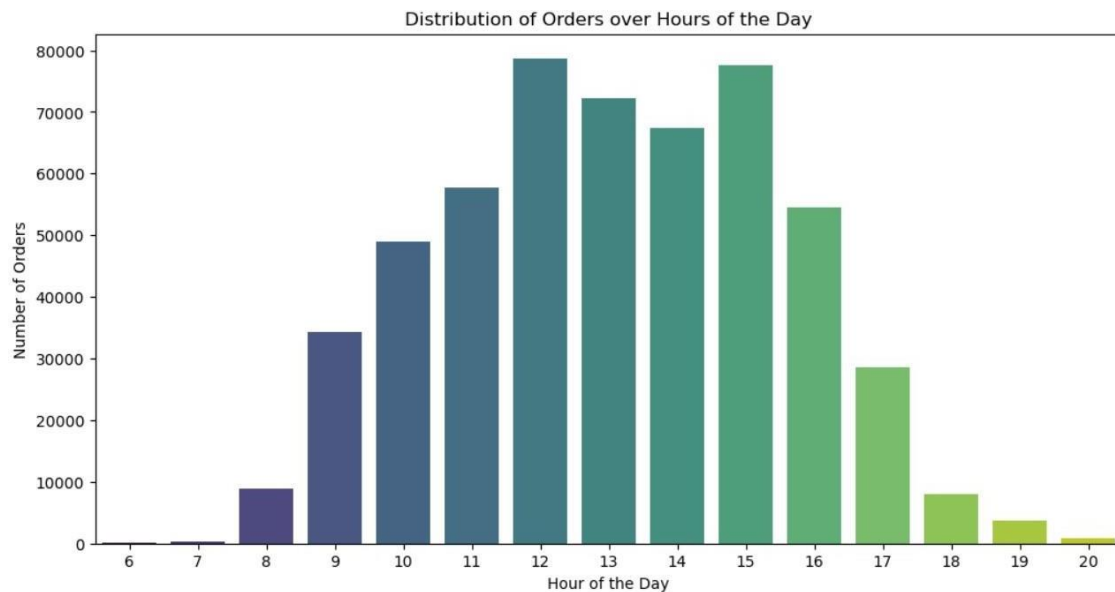
Plotting the Distribution of Orders over Hours of the Day:

- 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 chart is a bar chart named "Order Distribution by Day of the Week." It shows the number of orders placed on each day of the week, Monday through Sunday. The bars reflect the number of orders,

with the y-axis signifying quantity and the x-axis representing weekdays. Visually, it appears that the number of orders is quite stable during the week, with a slight reduction towards the weekend, while Sunday appears to have a slightly higher number of orders than Saturday. The precise numbers cannot be ascertained just from the image 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?

4b. What is average order processing time?

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

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

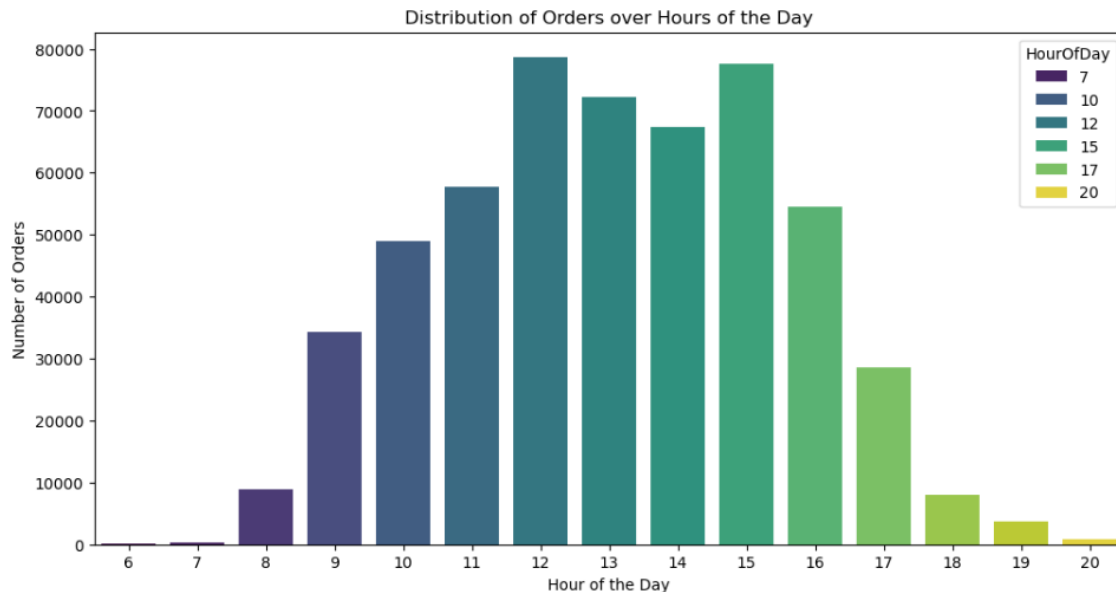
code first converts the 'InvoiceDate' column in the DataFrame `df` to datetime format. Then, it calculates the order processing time for each invoice by finding the time difference between the maximum and minimum 'InvoiceDate' within each group of 'InvoiceNo'. Finally, it calculates and prints the average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

4c. Are there any seasonal trends in the dataset?

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

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

```
[48]: plt.figure(figsize=(12, 6))
sns.countplot(x='HourOfDay', data=df, palette='viridis', hue='HourOfDay')
plt.title('Distribution of Orders over Hours of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Orders')
plt.show()
```

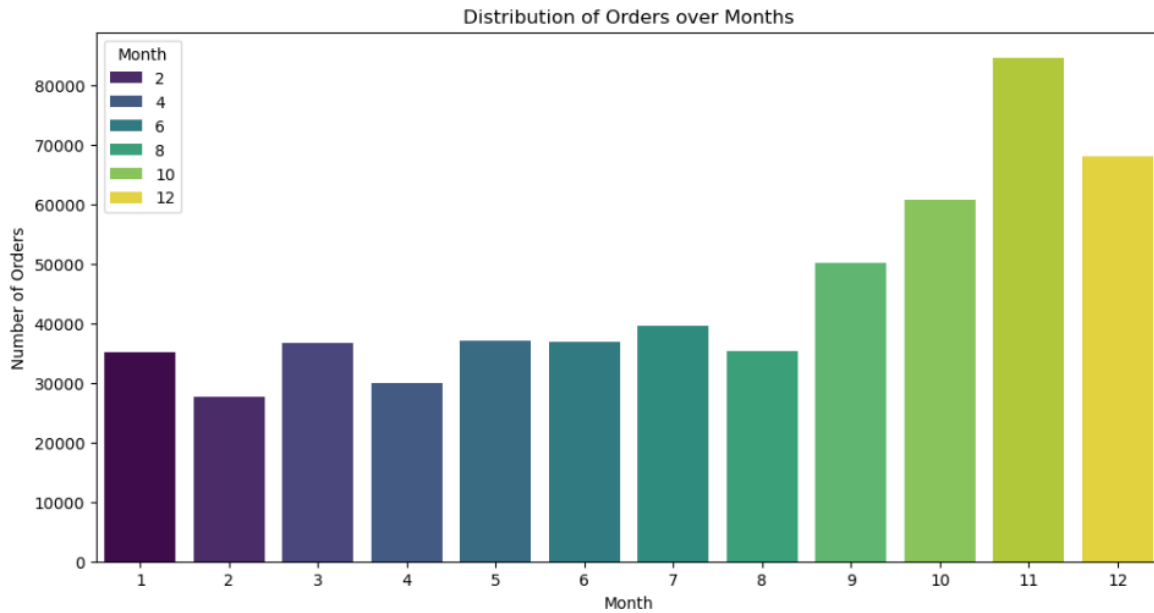


The bar chart in the picture with illustrates "Distribution of Orders over Hours of the Day." It shows the quantity of orders placed at different times, ranging from 6 AM to 8 PM. The x-axis shows the hours of the day, while the y-axis shows the number of orders. The data shows that the number of orders begins to increase at 6 AM and increases more noticeably at 10 AM.

The highest point on the chart, 3 PM (15:00), is when the order volume peaks.

Order volume drastically drops after 3 PM, with the fewest orders coming in around 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.

```
plt.figure(figsize=(12, 6))
sns.countplot(x='Month', data=df, palette='viridis', hue='Month')
plt.title('Distribution of Orders over Months')
plt.xlabel('Month')
plt.ylabel('Number of Orders')
plt.show()
```



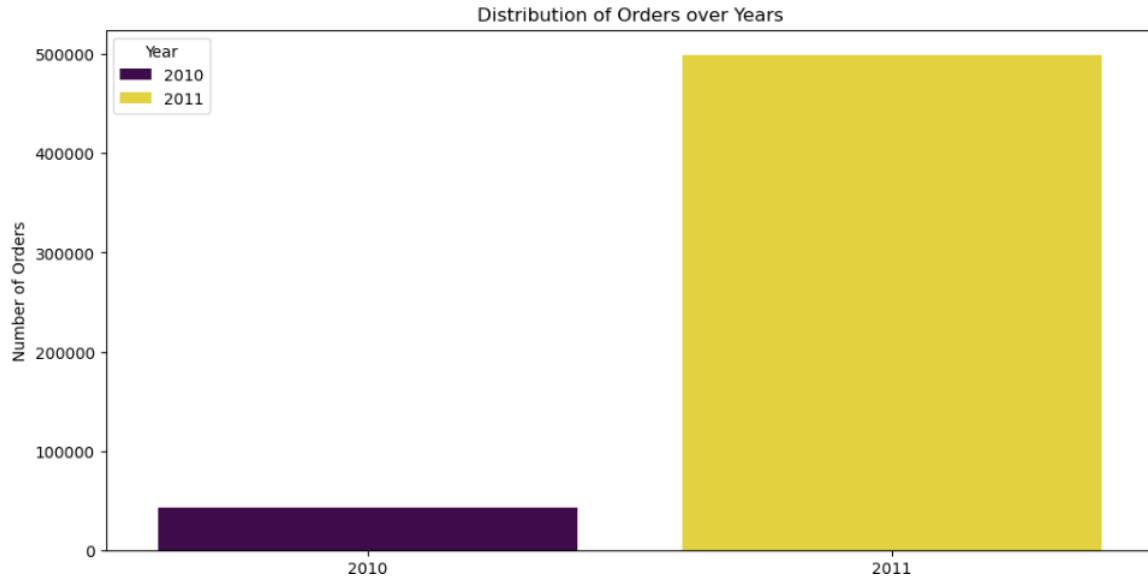
code creates a Seaborn countplot to visualize the distribution of orders over different months. The resulting bar plot displays the number of orders for each month, with the x-axis representing the months. The plot uses the 'viridis' color palette, and it includes a title, x-axis label ('Month'), y-axis 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". With labels ranging from 1 to 12, which represent January through December, it displays the quantity of orders for each month. The x-axis shows the month, and the y-axis counts the number of orders.

The chart's main findings are as follows:

- With some variations, the quantity of orders seems to rise steadily from January (Month 1) to December (Month 12).
- Orders significantly increase in the last quarter of the year, with December seeing the highest volume.
- With a potential peak during the holidays, this pattern may indicate seasonal changes in consumer behavior.


```
[52]: plt.figure(figsize=(12, 6))
sns.countplot(x='Year', data=df, palette='viridis', hue='Year')
plt.title('Distribution of Orders over Years')
plt.xlabel('Year')
plt.ylabel('Number of Orders')
plt.show()
```



A countplot showing the distribution of orders across various years is produced by the code using Seaborn. With the years on the x-axis, the resulting bar plot shows the quantity of orders for each year. A title, x-axis label ('Year'), y-axis label ('Number of Orders'), and a figure measuring 12 by 6 inches are all included in the plot, which employs the 'viridis' color scheme.

The bar chart in the picture is called "Distribution of Orders over Years." It contrasts the quantity of orders placed in 2010 and 2011. The x-axis lists the years, and the y-axis shows the number of orders. It is clear from the figure that the number of orders increased significantly between 2010 and 2011.

In contrast to 2011, when the number of orders skyrocketed, the 2010 bar displays noticeably fewer orders, suggesting growth or a notable shift in order volume between the two years.

It is evident that the number of orders placed increased significantly between 2010 and 2011. Additionally, we can see from the plots that as each year draws to a close, more orders are being placed. It is evident that there are far fewer orders placed on Saturdays than on other days. Very few orders are placed at night on any given day.

5. Geographical Analysis

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

```

In [ ]: import numpy as np
grouped_data = df.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 = df['Country'].value_counts().head(5)
print("Top 5 Countries with the Highest Number of Orders: ")
print(top_countries_by_orders)

```

Correlation between Number of Customers and Average Order Value: -0.11599992366073718

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 top 5 nations with the most orders in the DataFrame data are determined and printed by the code. After counting the instances of each nation using the 'value_counts()' method on the 'Country' column, it chooses the top 5 nations according to order frequency. The top nations and the accompanying order counts are displayed in the printed result.

b. From the dataset, the result shows the top 5 nations with the most orders. Is there a correlation between the country of the customer and the average order value?

```

grouped_data = df.groupby('Country')
country_stats = grouped_data.agg({ 'CustomerID': 'nunique',
    'Revenue': 'mean'
}).rename(columns={'CustomerID': 'No of Customers', 'Revenue': 'Average Order Value'})

correlation = np.corrcoef(country_stats['No of Customers'], country_stats['Average Order Value'])[0,1]
print ("Correlation between Number of Customers and Average Order Value:", correlation)

```

Correlation between Number of Customers and Average Order Value: -0.11599992366073718

For every nation in the dataset, the average order value is computed and printed in the code. It also determines the relationship between the average order value for each nation and the number of unique clients. The code prints the correlation between these two factors and offers insights into the relationship between the average order value and the number of clients in each nation.

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.

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

7a. How long, on average, do customers remain active (between their first and last purchase)?

```
df['InvoiceDate'] = pd.to_datetime (df['InvoiceDate'])
customer_active_duration = df.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 17:25:29.204025618
```

This code figures out how long customers stay active by looking at the time between their first and last purchase. The average amount of time that elapses between a client's first and last transactions is determined by calculating the average length of customer involvement. The outcome provides an idea of how long consumers typically stay involved with the company.

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

```

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
recency = df.groupby('CustomerID')['InvoiceDate'].max()
frequency = df.groupby('CustomerID')['InvoiceNo'].nunique()
monetary = df.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	2011-09-02 03:02:35.844350208	3.291843	968.043887	
1	2011-12-03 17:47:45.000000000	92.800000	55470.275500	
2	2011-11-22 18:58:41.117647104	20.420588	7606.949059	
3	2011-12-06 12:30:20.000000000	64.666667	241136.560000	

Number of Customers

Cluster	
0	4009
1	20
2	340
3	3

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

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

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

Total Profit: 9747747.933999998

This code determines the overall amount of money earned from each transaction. For each transaction, it multiplies the quantity of goods sold by their unit pricing, adds up these figures, and prints the total profit.

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

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

The top five products in the dataset with the highest profit margins are identified by this code. It determines the percentage of sales kept as profit, or the profit margin, for every product. 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

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

