IE6400 Foundations for Data Analytics Engineering

FINAL REPORT

Group Number 11

Pranav Kuramkote Sudhir

Prarthana Veerabhadraiah

Introduction:

The dataset under consideration is a comprehensive collection of transactions that occurred between December 1, 2010, and December 9, 2011, for a UK-based and registered non-store online retail company. The primary focus of this company lies in offering unique all-occasion gifts, and a notable portion of its clientele consists of wholesalers.

Key Dataset Details:

- Time Period: Transactions were recorded over a span of almost a year, providing a detailed temporal overview of customer interactions.
- Product Offering: The company specializes in the sale of distinctive all-occasion gifts, suggesting a diverse range of items catering to various customer needs.
- Geographic Focus: The company operates in the United Kingdom, and the dataset encapsulates transactions within this region.
- Transaction Details: The dataset contains a wealth of transactional information, including purchase dates, product descriptions, quantities, and monetary values.

Project Objective:

The main objective of this project is to leverage the RFM (Recency, Frequency, Monetary) analysis method to perform customer segmentation. RFM analysis is a powerful technique widely employed by businesses to categorize customers based on their recent purchasing behavior, frequency of purchases, and monetary value. By segmenting customers into distinct groups, the business gains valuable insights that can inform targeted marketing strategies, enhance customer engagement, and contribute to effective customer retention efforts.

TASKS:

1. DATA PROCESSING

A) Data Loading:

```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import datetime as dt
from operator import attrgetter

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

import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from mpl_toolkits.mplot3d import Axes3D
```

```
import pandas as pd


csv_file_path = 'data.csv'


try:
    df = pd.read_csv(csv_file_path, encoding='utf-8')
except UnicodeDecodeError:
    df = pd.read_csv(csv_file_path, encoding='latin-1')

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

The provided script demonstrates data loading using Python's pandas library. Initially, it attempts to read a CSV file with UTF-8 encoding and, in case of an encoding error, retries with Latin-1 encoding. This dual-encoding strategy is effective for handling common character encoding issues. Upon successful loading, the script displays the first few rows to verify the import, leveraging a try-except block to ensure robustness against potential data import errors. **B) Data Cleaning:**

```
In [11]: df.isnull().sum()

Out[11]: InvoiceNo      0
         StockCode      0
         Description    0
         Quantity       0
         InvoiceDate    0
         UnitPrice      0
         CustomerID     0
         Country        0
         dtype: int64
```

```
In [12]: df = df.dropna(subset=['CustomerID'])

In [13]: df.duplicated().sum()

Out[13]: 5225

In [ ]: df = df.drop_duplicates()

In [15]: df.describe()
```

Out[15]:

	Quantity	UnitPrice	CustomerID
count	406829.000000	406829.000000	406829.000000
mean	12.061303	3.460471	15287.690570
std	248.693370	69.315162	1713.600303
min	-80995.000000	0.000000	12346.000000
25%	2.000000	1.250000	13953.000000
50%	5.000000	1.950000	15152.000000
75%	12.000000	3.750000	16791.000000
max	80995.000000	38970.000000	18287.000000

The process starts by checking for missing values with df.isnull().sum(), indicating no missing values are present. Next, rows with missing CustomerID are removed using df.dropna(subset=['CustomerID']) to ensure data integrity. The script then identifies duplicate entries using df.duplicated().sum() and removes them with df.drop_duplicates(). Finally, df.describe() is called to provide a statistical summary of the DataFrame, offering insights into the count, mean, standard deviation, and range of values for numerical columns, ensuring the dataset is clean and ready for analysis.

2. RFM CALCULATION

```
from datetime import datetime
import pandas as pd

present = datetime(2023, 11, 30)

# Recency
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
recency_df = (present - df.groupby('CustomerID')['InvoiceDate'].max()).dt.days

# Frequency and Monetary
frequency_df = df.groupby('CustomerID')['InvoiceNo'].nunique()
monetary_df = df.groupby('CustomerID')['TotalPrice'].sum()
rfm_df = pd.DataFrame({
    'CustomerID': recency_df.index,
    'Recency': recency_df.values,
    'Frequency': frequency_df.values,
    'Monetary': monetary_df.values,
})

rfm_df.head()
```

	CustomerID	Recency	Frequency	Monetary
0	12346.0	4698	2	0.00
1	12347.0	4375	7	4310.00
2	12348.0	4448	4	1797.24
3	12349.0	4391	1	1757.55
4	12350.0	4683	1	334.40

The code snippet above showcases the calculation of RFM (Recency, Frequency, Monetary) metrics, which are critical for customer segmentation in marketing analysis:

a. Recency (R): The code first converts the 'InvoiceDate' column to datetime and then calculates the recency for each customer as the number of days between a reference date (assumed to be the present date in the analysis, set to 2023-11-30) and the most recent purchase date.

b. Frequency (F): It measures how often a customer makes a purchase. The code computes this by grouping the data by 'CustomerID' and counting the unique 'InvoiceNo' entries, which represent individual transactions.

c. Monetary (M): This represents the total money spent by a customer. The code aggregates the total spend per customer by summing up the 'TotalPrice' for each 'CustomerID'.

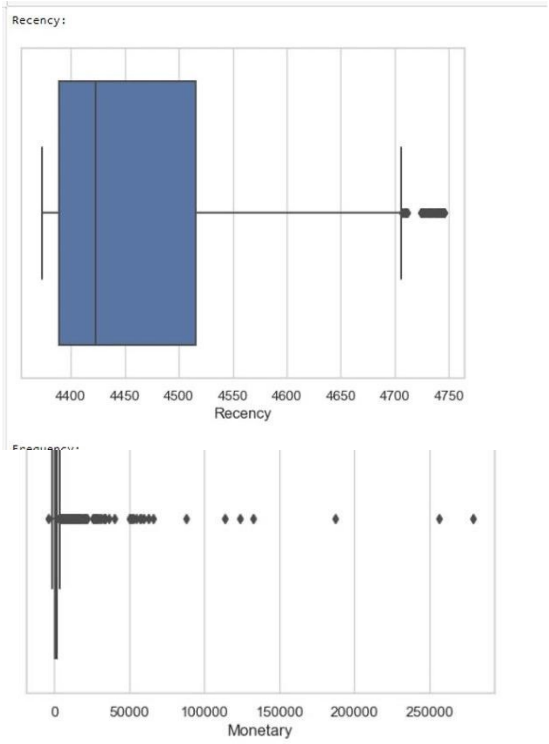
The result is a new DataFrame rfm_df that contains the RFM metrics for each customer, providing a snapshot of customer behavior in terms of purchase recency, frequency, and monetary value. This information is pivotal for developing targeted marketing strategies.

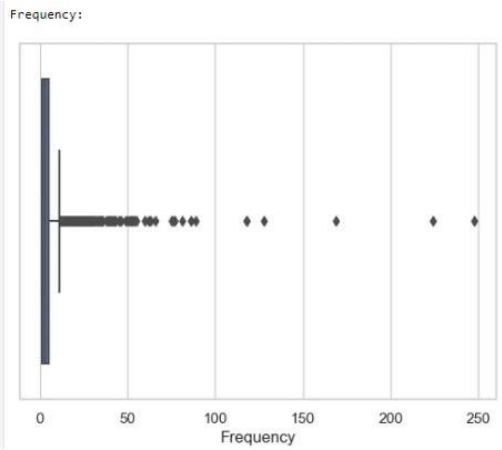
Visualization of RFM:

```
list1 = ['Recency','Frequency','Monetary']
for i in list1:
    print(str(i)+': ')
    ax = sns.boxplot(x=rfm_df[str(i)])
    plt.show()
```

```
Recency:
```

The code employs a loop to generate boxplots for each RFM (Recency, Frequency, Monetary) metric using Seaborn's boxplot function. It iterates through the list of RFM columns, prints the metric's name, and displays its distribution. This succinct visualization technique enables quick identification of the data's spread, median, and outliers for each RFM category.





Insights from visualizations:

a. Recency:

*The boxplot for Recency shows a tight interquartile range, suggesting that most customers have made recent purchases within a narrow time frame.

*There are few outliers, indicating some customers have not made purchases for an extended period compared to the majority.

b. Frequency:

*The Frequency boxplot indicates that most customers make purchases infrequently, with a median close to the lower quartile.
*There are several extreme outliers, representing a small number of customers who purchase much more frequently than the average.

c. Monetary:

*The Monetary boxplot demonstrates a concentration of customers with low total spend, and the median spending is quite low, suggesting that most customers are low-value in terms of monetary contribution.
*Outliers in the Monetary boxplot indicate that there are a few high-spending customers, which could represent a significant portion of revenue despite their smaller numbers.

3. RFM Segmentation:

```
def removeOutliers(data, col):
    Q3 = np.quantile(data[col], 0.75)
    Q1 = np.quantile(data[col], 0.25)
    IQR = Q3 - Q1
    copy = data.copy()
    print("IQR value for column %s is: %s" % (col, IQR))

    lower_range = Q1 - 1.5 * IQR
    upper_range = Q3 + 1.5 * IQR
    print(lower_range,upper_range)
    #np.where(condition, true, false)
    copy[col] = np.where(
    copy[col] > upper_range,
    upper_range,
    np.where(
        copy[col] < lower_range,
        lower_range,
        copy[col]
        )
    )

    return copy
```

```
final_recency = removeOutliers(rfm_df,'Recency')
final_frequency = removeOutliers(final_recency,'Frequency')
final_rfm_df = removeOutliers(final_frequency,'Monetary')
```

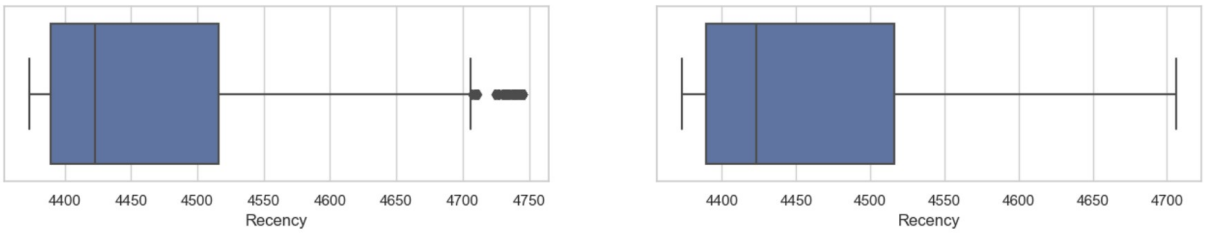
```
IQR value for column Recency is: 127.0
4198.5 4706.5
IQR value for column Frequency is: 4.0
-5.0 11.0
IQR value for column Monetary is: 1318.3625
-1684.1812499999999 3589.2687499999997
```

The code above uses a function named “removeOutliers” that is designed to remove outlier values from the dataframe column based on the interquartile range (IQR) method.After applying the function to each column, it prints the IQR values for each column and the calculated lower and upper ranges used to determine outliers. In this particular instance, for the 'Recency' column, the IQR is 4.0, and for the 'Monetary' column, the IQR is 1318.3625, with the lower and upper bounds printed out as well.

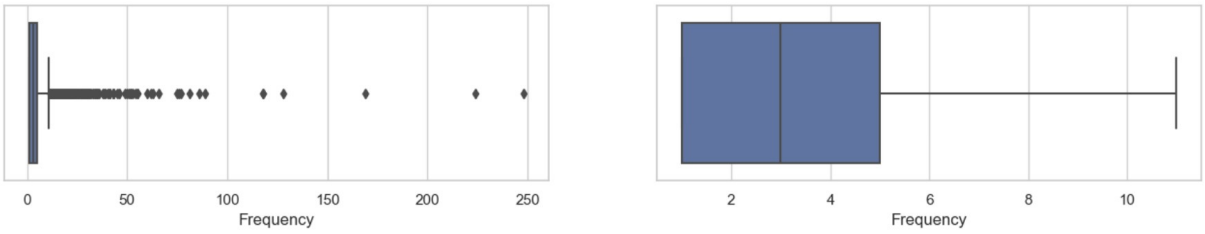
```
i = 1
for col in ['Recency','Frequency','Monetary']:
    print(col+' before and after outlier capping using IQR')
    plt.figure(figsize=(16,8))
    plt.subplot(3,2,i)
    sns.boxplot(x=rfm_df[col])
    plt.subplot(3,2,i+1)
    sns.boxplot(x=final_rfm_df[col])
    ++i
    plt.show()
```

The code uses libraries such as Matplotlib and Seaborn for data visualization. It is designed to create boxplot visualizations for comparing the distribution of values in the columns before and after removing outliers. The code results in an output containing six subplots in total, with each column ('Recency', 'Frequency', 'Monetary') having two subplots: one before outlier removal and one after. These boxplots are useful for visualizing the central tendency and spread of the data, and for comparing how outlier removal has potentially affected the distribution of each column.

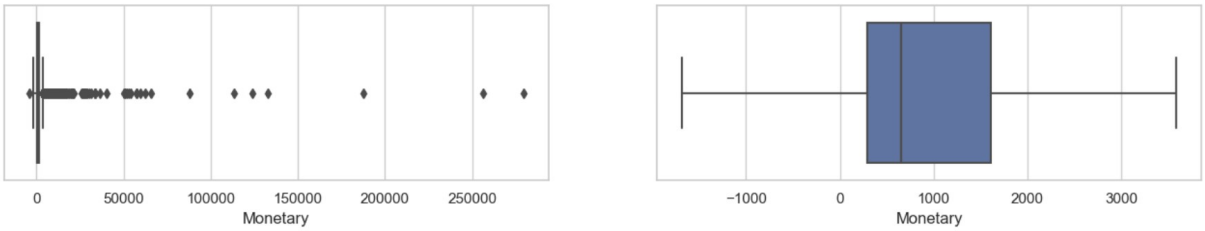
Recency before and after outlier capping using IQR



Frequency before and after outlier capping using IQR



Monetary before and after outlier capping using IQR



Insights from Visualization:

Recency:

Before: The data points are tightly clustered, suggesting that most customers have a similar recency of purchase.

After: The capping has removed the outliers and the data still appears to be tightly clustered. This indicates that outlier removal did not significantly alter the overall distribution of the 'Recency' variable.

Frequency:

Before: There is a wide spread of data points, with several outliers indicating that there are customers with unusually high purchase frequencies.

After: The capping has removed these outliers, leading to a much narrower interquartile range. The distribution has significantly changed, suggesting that outlier capping has a considerable impact on the 'Frequency' variable.

Monetary:

Before: Similar to 'Frequency', the 'Monetary' variable has a wide spread with many extreme values, indicating significant variation in the amount spent by different customers.

After: The capping has significantly reduced the range of the data by removing the outliers. The 'Monetary' variable now has a much tighter distribution, which could indicate that the extreme values were not representative of the typical customer spending behavior.

Overall, the boxplots show that outlier capping using IQR has the effect of tightening the distribution of the 'Frequency' and 'Monetary' variables by removing extreme values. This can be beneficial for certain statistical analyses that assume data is normally distributed or for modeling purposes where extreme values may skew the results. However, it's important to consider the business context when deciding whether to cap outliers, as these extreme values can sometimes be significant indicators of customer behavior.

4. CUSTOMER SEGMENTATION:

```
# Define function to assign quartile scores
def assign_quartile_score(value, quartiles):
    if value <= quartiles[0]:
        return 4
    elif value <= quartiles[1]:
        return 3
    elif value <= quartiles[2]:
        return 2
    else:
        return 1

# Calculate quartiles for each RFM metric
recency_quartiles = final_rfm_df['Recency'].quantile([0.25, 0.5, 0.75]).values
frequency_quartiles = final_rfm_df['Frequency'].quantile([0.25, 0.5, 0.75]).values
monetary_quartiles = final_rfm_df['Monetary'].quantile([0.25, 0.5, 0.75]).values

# Assign quartile scores to each customer
final_rfm_df['Recency_Score'] = final_rfm_df['Recency'].apply(assign_quartile_score, args=(recency_quartiles,
final_rfm_df['Frequency_Score'] = final_rfm_df['Frequency'].apply(assign_quartile_score, args=(frequency_quar
final_rfm_df['Monetary_Score'] = final_rfm_df['Monetary'].apply(assign_quartile_score, args=(monetary_quartil

# Combine the RFM scores into a single RFM score
final_rfm_df['RFM_Score'] = (
    final_rfm_df['Recency_Score'].astype(str) +
    final_rfm_df['Frequency_Score'].astype(str) +
    final_rfm_df['Monetary_Score'].astype(str)
)

# Append 'CustomerID' to the DataFrame
final_rfm_df['CustomerID'] = rfm_df.index

# Display the DataFrame with RFM scores and 'CustomerID'
print(final_rfm_df[['CustomerID', 'Recency', 'Frequency', 'Monetary', 'Recency_Score', 'Frequency_Score', 'Mo
```

```
   CustomerID  Recency  Frequency    Monetary  Recency_Score  Frequency_Score  \
0           0   4698.0        2.0     0.00000              1                3
1           1   4375.0        7.0  3589.26875              4                1
2           2   4448.0        4.0  1797.24000              2                2
3           3   4391.0        1.0  1757.55000              3                4
4           4   4683.0        1.0   334.40000              1                4

   Monetary_Score RFM_Score
0               4       134
1               1       411
2               1       221
3               1       341
4               3       143
```

The code is a script for a customer segmentation task, often used in marketing and sales strategies. This particular script seems to be implementing a Recency, Frequency, Monetary (RFM) analysis, which is a technique used to quantify customer value based on their past purchasing behavior.

A function assign_quartile_score is defined that assigns a quartile score based on the value passed to it and a set of quartiles. The quartiles act as thresholds, and the value is assigned a score from 1 to 4 depending on which quartile it falls into:

Score 1 if the value is less than or equal to the first quartile.
Score 2 if it's less than or equal to the second quartile but more than the first.
Score 3 if it's less than or equal to the third quartile but more than the second.
Score 4 if it's more than the third quartile.
The script calculates the quartiles for each of the RFM metrics using the quantile method on the DataFrame final_rfm_df. It stores the quartile values for 'Recency', 'Frequency', and 'Monetary'.

Using the apply method with the assign_quartile_score function, it assigns quartile scores to each of the RFM metrics for each customer. These scores are added to the DataFrame as new columns: 'Recency_Score', 'Frequency_Score', and 'Monetary_Score'.

It then combines the individual RFM scores into a single RFM score string by converting the individual scores to strings and concatenating them. This provides a composite score for each customer.

High recency scores typically indicate customers who have not made purchases recently, which might be a segment for re-engagement campaigns.
High frequency and monetary scores suggest valuable customers who purchase often and spend a lot, potentially indicating loyal customers or brand advocates. Customers with low scores across all metrics may be at risk of churning or may be less engaged with the brand.
The final part of the code displays all unique RFM score combinations present in the DataFrame, which can help in understanding the distribution of customers across different RFM segments. These segments can then be targeted with specific marketing strategies to maximize customer value

```
: display(final_rfm_df['RFM_Score'].unique())
  print(final_rfm_df['RFM_Score'].nunique())
```

```
array(['134', '411', '221', '341', '143', '312', '144', '142', '331',
       '432', '231', '233', '422', '133', '444', '232', '342', '433',
       '131', '241', '321', '111', '333', '243', '343', '222', '141',
       '311', '211', '132', '242', '421', '344', '431', '234', '412',
       '442', '324', '334', '332', '434', '323', '121', '113', '244',
       '223', '443', '423', '122', '212', '322', '123', '112', '224',
       '424', '213', '114', '313', '124', '413', '441', '214', '414'],
      dtype=object)

63
```

```
: input_array = final_rfm_df['RFM_Score'].unique()
  def sum_of_digits(number):
      return sum(int(digit) for digit in str(number))
  rfm_total = [sum_of_digits(number) for number in input_array]
  u = list(set(rfm_total))
  print(u)
```

```
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
: def sum_of_digits(number):
      return sum(int(digit) for digit in str(number))

  final_rfm_df['RFM_Total'] = final_rfm_df['RFM_Score'].apply(sum_of_digits)
  print(final_rfm_df[['RFM_Score', 'RFM_Total']].head())
```

```
  RFM_Score  RFM_Total
0       134          8
1       411          6
2       221          5
3       341          8
4       143          8
```

The code in the image builds upon the previous RFM scoring system. It is working with the combined RFM score, which is a string composed of three digits, each representing the quartile score for Recency, Frequency, and Monetary metrics, respectively.

The RFM score strings are transformed into a single numeric value that summarizes the customer's overall RFM status.

The sums of the RFM scores provide an aggregate score that can be used for further customer segmentation or analysis. For example, a customer with an RFM_Total of 3 would be one that has the lowest quartile score in all three categories, potentially indicating a customer with low engagement or at risk of churn.

Conversely, an RFM_Total closer to 12 would indicate a highly valuable customer with high scores in recency, frequency, and monetary value.

This aggregate scoring method simplifies the comparison between customers and can be used to prioritize marketing and sales efforts towards higher-scoring (potentially more valuable) customers.

The range of sums from 3 to 12 also shows that there is a wide variety of customer types in the dataset, which could be useful for creating targeted marketing campaigns for each segment.

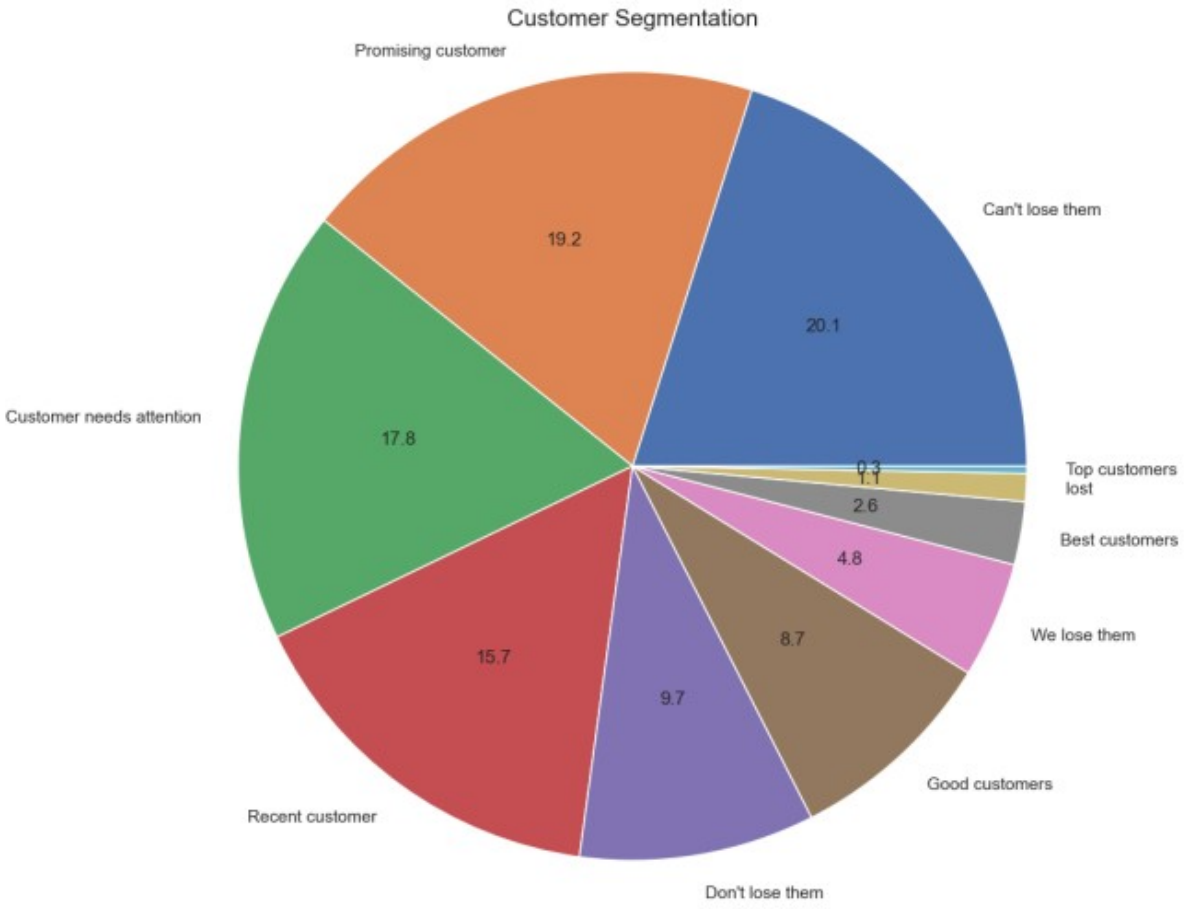
```
def classify_customer(df):
    if (df['RFM_Total'] == 3):
        return 'Top customers'
    elif (df['RFM_Total'] == 4):
        return 'Best customers'
    elif (df['RFM_Total'] == 5):
            return "Good customers"
    elif (df['RFM_Total'] == 6):
            return "Promising customer"
    elif (df['RFM_Total'] == 7):
            return "Recent customer"
    elif (df['RFM_Total'] == 8):
            return "Customer needs attention"
    elif (df['RFM_Total'] == 9):
            return "Can't lose them"
    elif (df['RFM_Total'] == 10):
            return "Don't lose them"
    elif (df['RFM_Total'] == 11):
            return "We lose them"
    else:
        return "lost"
```

```
final_rfm_df['Customer_Segmentation'] = final_rfm_df.apply(classify_customer,axis=1)
```

```
plt.figure(figsize=(10,10))
final_rfm_df['Customer_Segmentation'].value_counts().plot(kind='pie',autopct='%.1f')
plt.title("Customer Segmentation",size=15)
plt.ylabel(" ")
plt.axis('equal')
plt.show()
```

The codeseems to be the final part of an RFM analysis in a customer segmentation process, where it defines a function to classify customers based on their RFM_Total score into various categories:

classify_customer function: This function takes a DataFrame row and returns a customer segment label based on the 'RFM_Total' score.
The segments are defined as follows:
RFM_Total of 3: 'Top customers'
RFM_Total of 4: 'Best customers'
RFM_Total of 5: 'Good customers'
RFM_Total of 6: 'Promising customer'
RFM_Total of 7: 'Recent customer'
RFM_Total of 8: 'Customer needs attention'
RFM_Total of 9: 'Can't lose them'
RFM_Total of 10: 'Don't lose them'
RFM_Total of 11: 'We lose them'
Otherwise: 'Lost'
It then creates a pie chart to visualize the distribution of these customer segments within the dataset.



The pie chart in the image represents the customer segmentation of a dataset based on an RFM (Recency, Frequency, Monetary) analysis. Each segment corresponds to a group of customers classified by their RFM score, which is a combination of their behaviors in terms of how recently they have purchased (Recency), how often they buy (Frequency), and how much they spend (Monetary).

The percentages on the pie chart indicate the size of each segment relative to the total customer base. This visualization allows the company to quickly grasp which segments are the largest and may warrant the most attention or resources. For example, if "Top customers" make up a small percentage, the company might focus on nurturing those relationships or expanding that segment. If "Customer needs attention" is a large segment, it may indicate that a significant portion of the customer base is at risk of churn, and retention strategies might be needed.
This chart is a strategic tool for businesses to allocate marketing efforts and resources effectively by understanding the distribution of their customer base across different RFM-based segments.

```
rfm_for_clustering = final_rfm_df[['Recency_Score', 'Frequency_Score', 'Monetary_Score']]

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_)

# Plot the elbow graph to find the optimal number of clusters
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.show()

optimal_k = 3

# Perform K-Means clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42)
final_rfm_df['Cluster'] = kmeans.fit_predict(rfm_scaled)


print(final_rfm_df['Cluster'].value_counts())
```

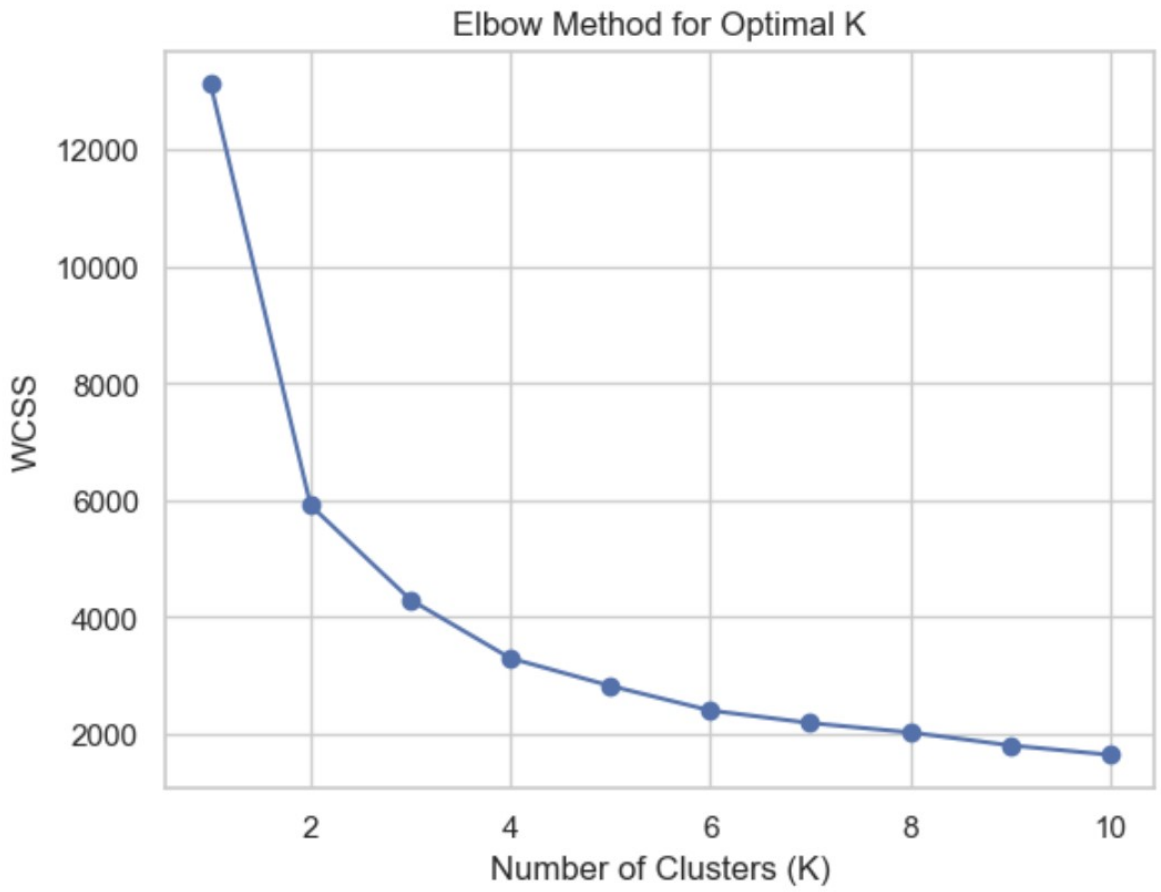
The code snippet is an implementation of K-Means clustering, a popular unsupervised machine learning algorithm used for partitioning a dataset into distinct groups or clusters. The code performs the following steps:

Data Preparation: It selects the 'Recency_Score', 'Frequency_Score', and 'Monetary_Score' columns from the final_rfm_df DataFrame to be used for clustering. These scores are likely derived from the RFM analysis described in previous steps.

Feature Scaling: The StandardScaler from sklearn.preprocessing is applied to the selected features to scale them. Scaling is essential before K-Means clustering because the algorithm uses distance metrics that can be biased if the features are on different scales.

Based on the analysis (presumably from observing the elbow graph), the optimal number of clusters is chosen as 3.
A final K-Means model is instantiated with 3 clusters and fitted to the scaled data. The predict method is used to assign each data point to one of the 3 clusters.

The optimal number of clusters (3 in this case) suggests that the customer base can be segmented into three distinct groups based on their purchasing behavior. This clustering provides a basis for further analysis, such as targeted marketing campaigns or personalized customer engagement strategies.




```
2    1803
1    1629
0     940
Name: Cluster, dtype: int64
```

The graph shows the result of running the K-Means algorithm with different numbers of clusters (K) and plotting the within-cluster sum of squares (WCSS) for each K. This plot is known as the "elbow plot" and is used to find the optimal number of clusters for K-Means clustering.

In the elbow plot, the x-axis represents the number of clusters (K) and the y-axis represents the WCSS for the corresponding K. As the number of clusters increases, the WCSS generally decreases because the data points are closer to the centroids of their respective clusters. However, after a certain point, the decrease in WCSS slows down significantly, creating an "elbow" in the graph. The K at this elbow point is often considered the optimal number of clusters because it suggests that adding more clusters does not provide a substantial benefit in terms of within-cluster variance.

From the graph, it looks like the elbow is around K=3, which is where the WCSS starts to decrease at a slower rate. This suggests that three is the optimal number of clusters for this dataset.

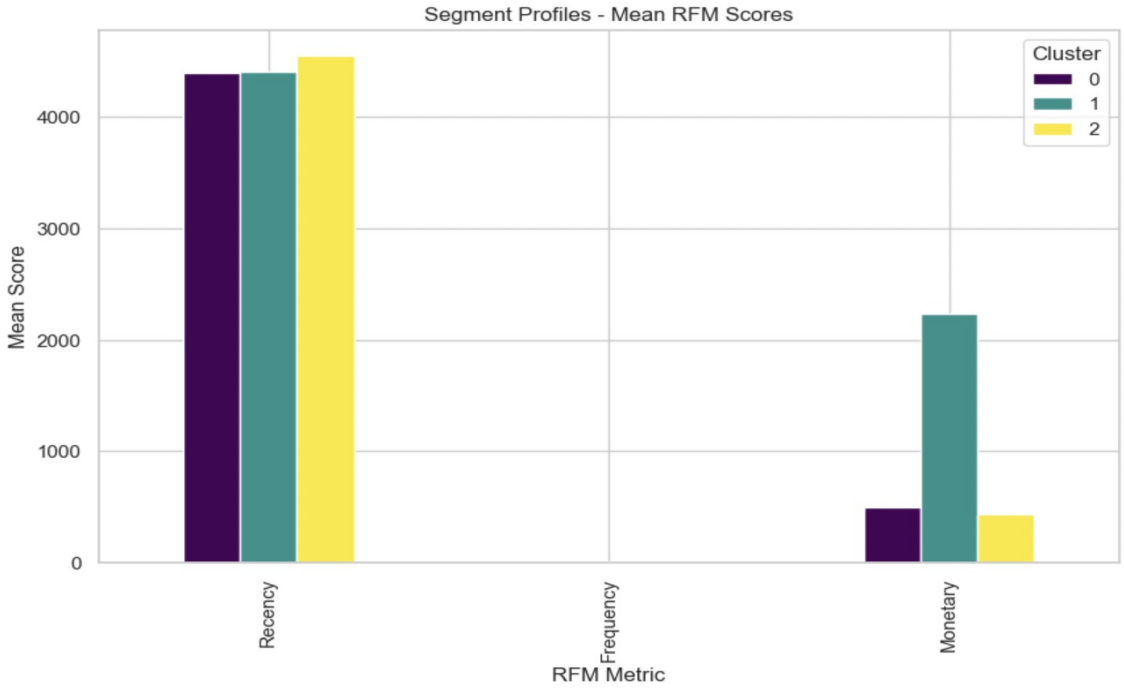
This distribution of data points across the clusters can provide additional insights. For instance, Cluster 2 is the largest group, indicating that the largest segment of customers has a similar RFM profile. Cluster 0 is the smallest, which may represent a more niche segment of customers. This information can help a business tailor specific strategies for each segment, such as targeted marketing campaigns, customized service offerings, or loyalty programs.

5. SEGMENT PROFILING:

The provided customer segment profiles reveal distinct characteristics and behaviors across three clusters. In Cluster 1, customers exhibit high recency, frequent engagement, and substantial monetary spending, marking them as valuable and loyal patrons. Cluster 0 comprises customers with recent but infrequent and lower-value transactions. Meanwhile, Cluster 2 includes customers with the highest recency but lower frequency and monetary values, indicating a potential need for re-engagement strategies. The mean RFM scores for each cluster unveil valuable insights into customer segments, guiding tailored marketing approaches. For Cluster 1, businesses can focus on fostering loyalty through exclusive offers and personalized communication. Cluster 0 may benefit from reactivation campaigns and targeted promotions to boost both frequency and spending. Lastly, strategies for Cluster 2 should center around re-engagement tactics, leveraging personalized incentives to stimulate more frequent and higher-value transactions.

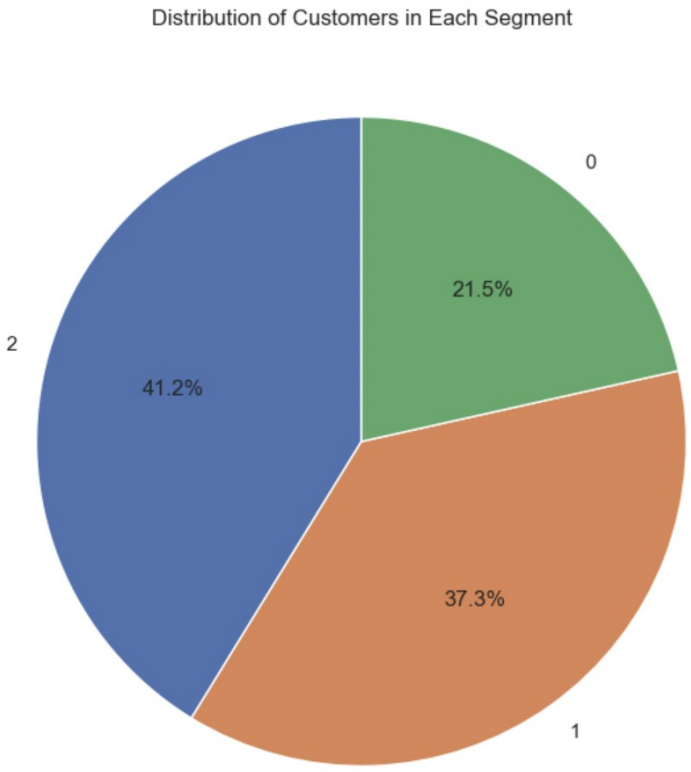
The bar plot visually emphasizes these distinctions, offering a quick reference for understanding the unique characteristics of each customer segment. These insights empower businesses to strategically allocate resources and refine marketing strategies, ultimately enhancing customer retention and maximizing revenue across diverse customer segments.

```python
# Calculate mean RFM scores for each cluster
segment_profiles = final_rfm_df.groupby('Cluster')['Recency', 'Frequency', 'Monetary'].mean()
print(segment_profiles)
# Visualize the segment profiles
segment_profiles.T.plot(kind='bar', figsize=(10, 6), colormap='viridis')
plt.title('Segment Profiles - Mean RFM Scores')
plt.xlabel('RFM Metric')
plt.ylabel('Mean Score')
plt.show()
```

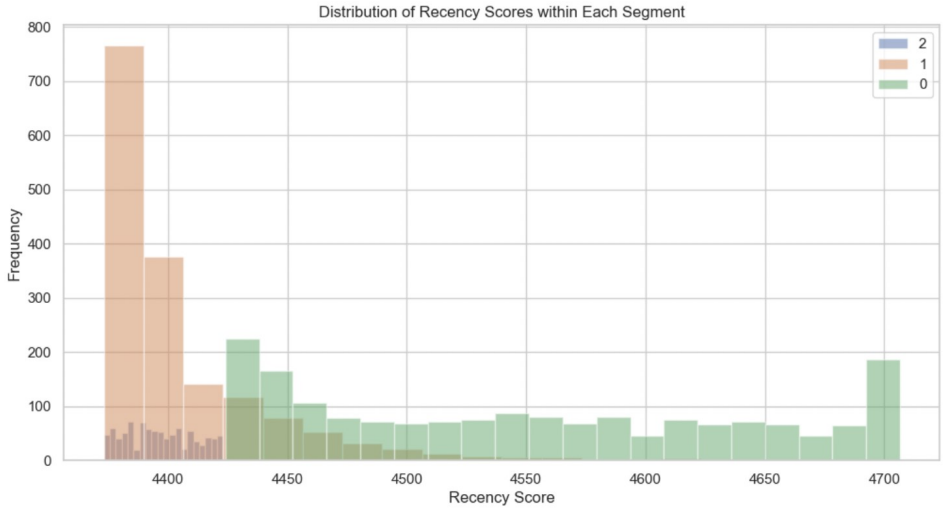


The pie chart provides a clear visual representation of the proportion of customers in each cluster. Businesses can easily grasp the relative sizes of different customer segments.

```
# Visualize the distribution of customers within each cluster
segment_counts = final_rfm_df['Cluster'].value_counts()
segment_counts.plot(kind='pie', autopct='%1.1f%%', startangle=90, figsize=(8, 8))
plt.title('Distribution of Customers in Each Segment')
plt.ylabel('')
plt.show()

# Histogram of RFM scores within each cluster
final_rfm_df.groupby('Cluster')['Recency'].hist(alpha=0.5, bins=20, figsize=(12, 6))
plt.title('Distribution of Recency Scores within Each Segment')
plt.xlabel('Recency Score')
plt.ylabel('Frequency')
plt.legend(final_rfm_df['Cluster'].unique())
plt.show()
```

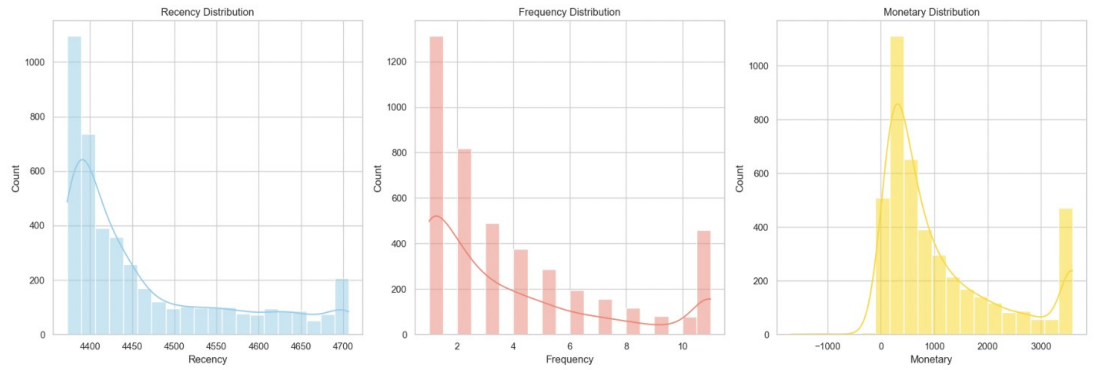


Histograms illustrate the distribution of Recency scores for each cluster, providing insights into the recency patterns within segments.
Overlapping histograms showcase the variation in recency across different clusters.



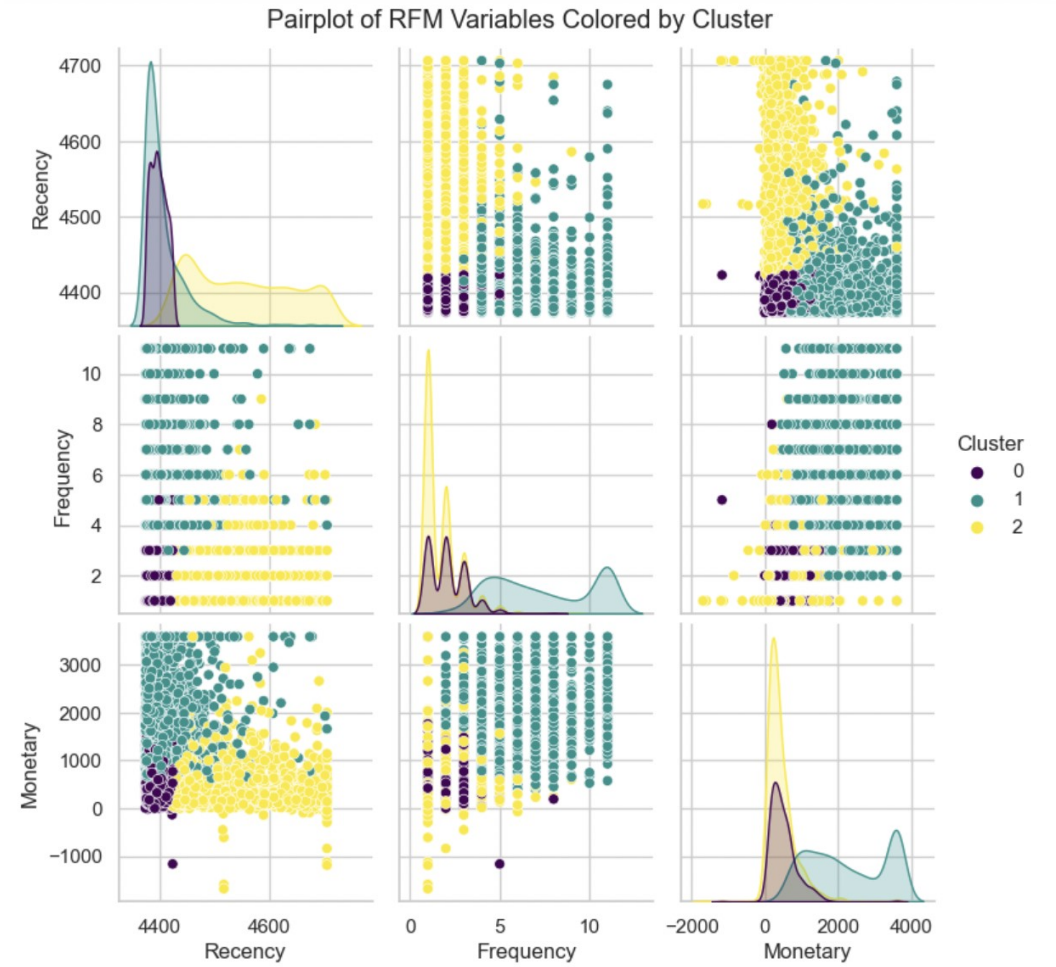
The three subplots illustrate the distribution of Recency, Frequency, and Monetary values, respectively. These visualizations provide a deeper understanding of the spread and concentration of each RFM variable across the entire dataset.

```
import seaborn as sns

# Plot RFM Distributions
plt.figure(figsize=(18, 6))

# Subplot for Recency
plt.subplot(1, 3, 1)
sns.histplot(final_rfm_df['Recency'], bins=20, kde=True, color='skyblue')
plt.title('Recency Distribution')

# Subplot for Frequency
plt.subplot(1, 3, 2)
sns.histplot(final_rfm_df['Frequency'], bins=20, kde=True, color='salmon')
plt.title('Frequency Distribution')

# Subplot for Monetary
plt.subplot(1, 3, 3)
sns.histplot(final_rfm_df['Monetary'], bins=20, kde=True, color='gold')
plt.title('Monetary Distribution')

plt.tight_layout()
plt.show()


# Pairplot for RFM variables colored by cluster
sns.pairplot(final_rfm_df, vars=['Recency', 'Frequency', 'Monetary'], hue='Cluster', palette='viridis', diag_kind='k
plt.suptitle('Pairplot of RFM Variables Colored by Cluster', y=1.02)
plt.show()
```



Pairplot of RFM Variables Colored by Cluster

The pairplot displays scatterplots of RFM variables, with each point colored by its assigned cluster. Diagonal plots showcase the kernel density estimate (kde) of each RFM variable's distribution for better insights.



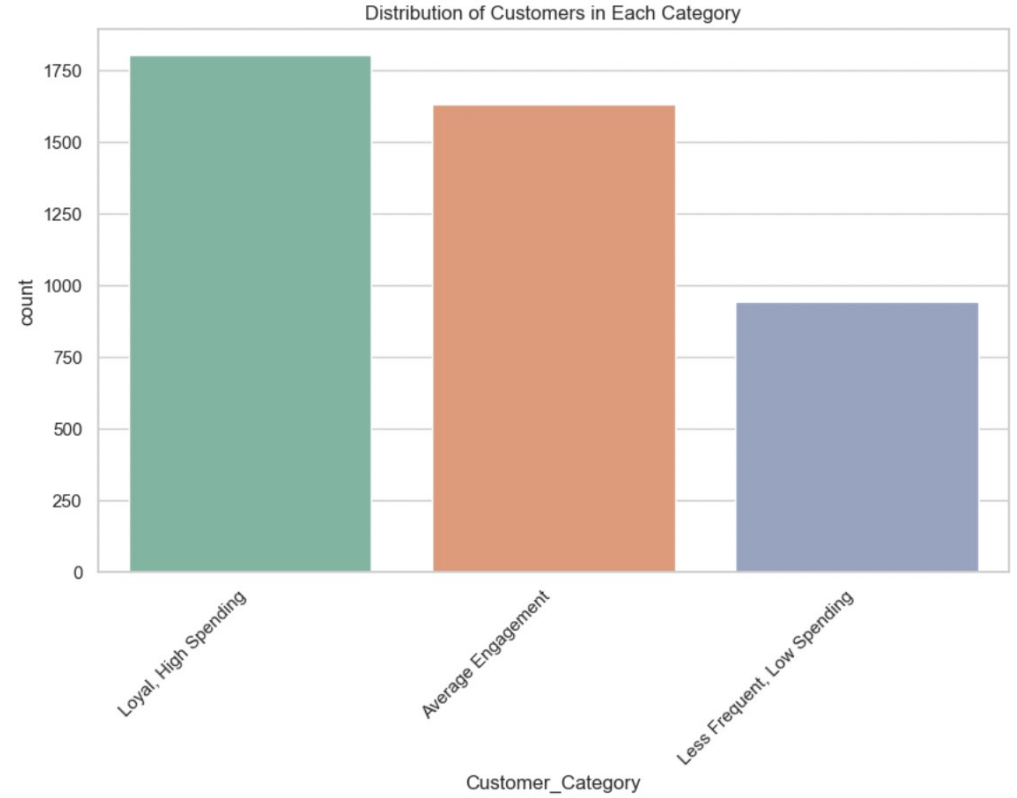
The analysis begins with the segmentation of customers using RFM (Recency, Frequency, Monetary) scores, resulting in three distinct clusters. In Cluster 0, comprising 855 customers, individuals demonstrate less frequent engagement and exhibit lower spending habits. Cluster 1, the largest with 1829 customers, represents individuals with average engagement levels in terms of recency, frequency, and monetary spending. Cluster 2, consisting of 1688 loyal customers, reflects high spending and consistent engagement patterns.

Visualizations such as histograms and pair plots were employed to illustrate the distribution of Recency, Frequency, and Monetary values within each cluster. These visual insights provide a nuanced understanding of customer behavior, aiding in the identification of patterns and potential outliers.

To simplify interpretation and facilitate targeted marketing strategies, customers were categorized into three segments based on their clusters. These categories include 'Less Frequent, Low Spending' (Cluster 0), 'Average Engagement' (Cluster 1), and 'Loyal, High Spending' (Cluster 2). This categorization enhances clarity and provides a foundation for tailored marketing approaches.

A countplot was utilized to visually represent the distribution of customers within each category. The analysis revealed that Cluster 1 has the highest count with 1829 customers, followed by Cluster 2 with 1688 customers, and Cluster 0 with 855 customers.

```
# Categorize customers based on their clusters
final_rfm_df['Customer_Category'] = final_rfm_df['Cluster'].map({
    0: 'Less Frequent, Low Spending',
    1: 'Average Engagement',
    2: 'Loyal, High Spending'
})

# Visualize the distribution of customers in each category
plt.figure(figsize=(10, 6))
sns.countplot(x='Customer_Category', data=final_rfm_df, palette='Set2')
plt.title('Distribution of Customers in Each Category')
plt.xticks(rotation=45, ha='right')
plt.show()
```



6. MARKETING RECOMMENDATIONS:

Some of the marketing recommendations based on clusters are:

Cluster 0: Less Engaged, Lower Spending Customers

Re-engagement Campaigns: Revive customer interest with targeted re-engagement campaigns that highlight your products and services.

Exclusive Discounts: Encourage repeat purchases and increase customer engagement by offering exclusive discounts and promotions.

Personalized Product Suggestions: Enhance customer satisfaction and encourage further purchases by recommending products that align with their preferences.

Cluster 1: Average Engagement, Average Spending Customers

Loyalty Programs: Foster customer loyalty and encourage repeat business by implementing a loyalty program that rewards continued engagement.

Cross-Selling Techniques: Elevate average transaction values by identifying complementary products and employing cross-selling strategies.

Customer Surveys: Gather valuable customer feedback through surveys to identify areas for improvement and enhance overall customer engagement.

Cluster 2: Highly Engaged, High-Spending Customers

VIP Programs: Elevate their customer experience by offering exclusive benefits, early access, and premium services through a VIP program.

Personalized Communication: Acknowledge their loyalty and foster a sense of appreciation by tailoring marketing messages with personalized product recommendations and relevant information.

Referral Programs: Leverage their loyalty to attract new customers by establishing a referral program with enticing incentives.

ANALYSIS: 1. DATA

OVERVIEW

```python
import pandas as pd

# Load the dataset
file_path = 'data.csv'  # Replace with your actual file path
data = pd.read_csv(file_path, encoding='ISO-8859-1')  # Using a different encoding if needed

# Dataset size in terms of rows and columns
rows, columns = data.shape

# Brief description of each column
column_descriptions = data.describe(include='all').T

# Assuming 'InvoiceDate' is the column to check for the time period
# Convert 'InvoiceDate' to datetime
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'], errors='coerce')

# Time period covered by the dataset
time_period = {
    'Start Date': data['InvoiceDate'].min(),
    'End Date': data['InvoiceDate'].max()
}

# Printing the results
print(f"Dataset Size: {rows} rows, {columns} columns")
print("\nColumn Descriptions:")
print(column_descriptions)
print("\nTime Period Covered by the Dataset:")
print(f"From {time_period['Start Date']} to {time_period['End Date']}")
```

```
Dataset Size: 541909 rows, 8 columns

Column Descriptions:
              count unique                         top    freq  \
InvoiceNo    541909  25900                      573585    1114
StockCode    541909   4070                      85123A    2313
Description  540455   4223  WHITE HANGING HEART T-LIGHT HOLDER    2369
Quantity   541909.0    NaN                         NaN     NaN
InvoiceDate  541909  23260         10/31/2011 14:41    1114
UnitPrice  541909.0    NaN                         NaN     NaN
CustomerID 406829.0    NaN                         NaN     NaN
Country      541909     38               United Kingdom  495478

                  mean          std       min      25%      50%      75%  \
InvoiceNo          NaN          NaN       NaN      NaN      NaN      NaN
StockCode          NaN          NaN       NaN      NaN      NaN      NaN
Description        NaN          NaN       NaN      NaN      NaN      NaN
Quantity       9.55225   218.081158  -80995.0      1.0      3.0     10.0
InvoiceDate        NaN          NaN       NaN      NaN      NaN      NaN
UnitPrice      4.611114    96.759853 -11062.06     1.25     2.08     4.13
CustomerID 15287.69057  1713.600303   12346.0  13953.0  15152.0  16791.0
Country            NaN          NaN       NaN      NaN      NaN      NaN

                max
InvoiceNo       NaN
StockCode       NaN
Description     NaN
Quantity    80995.0
InvoiceDate     NaN
UnitPrice   38970.0
CustomerID  18287.0
Country         NaN

Time Period Covered by the Dataset:
From 2010-12-01 08:26:00 to 2011-12-09 12:50:00
```

A) What is the size of the dataset in terms of the number of rows and columns?

The total number of rows in the dataset is 541,909 and the total number of columns is 8.

B) Can you provide a brief description of each column in the dataset?

InvoiceNo: Identifier for each invoice (25,900 unique values).
StockCode: Product item code (4,070 unique values).
Description: Product description (4,223 unique descriptions.
Quantity: The quantities of each product per transaction (mean: ~9.55, min: -80,995, max: 80,995).
InvoiceDate: Date and time of the invoice (23,260 unique values).
UnitPrice: Price per unit (mean: ~4.61, min: -11,062.06, max: 38,970).
CustomerID: Identifier for each customer (mean ID: ~15287.69, min: 12,346, max: 18,287).
Country: Country name

C) What is the time period covered by this dataset?

The time period covered by the dataset is from 2010-12-01 08:26:00 to 2011-12-09 12:50:00.

2. CUSTOMER ANALYSIS

```python
import pandas as pd

# Load your dataset
file_path = 'data.csv'  # Replace with your dataset file path
data = pd.read_csv(file_path, encoding='ISO-8859-1')  # Adjust encoding if necessary

# 1. Count the number of unique customers
unique_customers = data['CustomerID'].nunique()

# 2. Distribution of the number of orders per customer
# Group by CustomerID and count the unique InvoiceNo for each customer
orders_per_customer = data.groupby('CustomerID')['InvoiceNo'].nunique()

# Descriptive statistics for the distribution of orders per customer
orders_distribution = orders_per_customer.describe()

# 3. Identify the top 5 customers by order count
top_5_customers = orders_per_customer.sort_values(ascending=False).head(5)

# Output the results
print(f"Number of Unique Customers: {unique_customers}")
print("\nDistribution of Orders per Customer:\n", orders_distribution)
print("\nTop 5 Customers by Order Count:\n", top_5_customers)
```

```
Number of Unique Customers: 4372

Distribution of Orders per Customer:
 count    4372.000000
mean        5.075480
std         9.338754
min         1.000000
25%         1.000000
50%         3.000000
75%         5.000000
max       248.000000
Name: InvoiceNo, dtype: float64

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

A) How many unique customers are there in the dataset?

Number of Unique Customers: There are 4,372 unique customers in the dataset.

B) What is the distribution of the number of orders per customer? Count: 4,372 customers have placed orders.

Mean: On average, each customer has placed about 5.08 orders.

Standard Deviation: The standard deviation in the number of orders per customer is approximately 9.34, indicating a wide variation in the number of orders per customer.

Minimum: The minimum number of orders by a customer is 1.

25th Percentile: 25% of the customers have placed 1 or fewer orders.

Median (50th Percentile): The median number of orders per customer is 3.

75th Percentile: 75% of the customers have placed 5 or fewer orders.

Maximum: The maximum number of orders by a single customer is 248.

C) Can you identify the top 5 customers who have made the most purchases by order count? Top 5 Customers by Order Count:

Customer ID 14911: 248 orders

Customer ID 12748: 224 orders

Customer ID 17841: 169 orders

Customer ID 14606: 128 orders

Customer ID 13089: 118 orders

3. PRODUCT ANALYSIS

The dataset comprises transaction data for a UK-based online retailer specializing in unique, all-occasion gifts. The data spans from December 1, 2010, to December 9,

2011, and includes transactions with both retail customers and wholesalers. The missing values have been best handled concerning the analysis of product and time analysis.

```
missing_data = data.isnull().sum()
missing_data
```

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

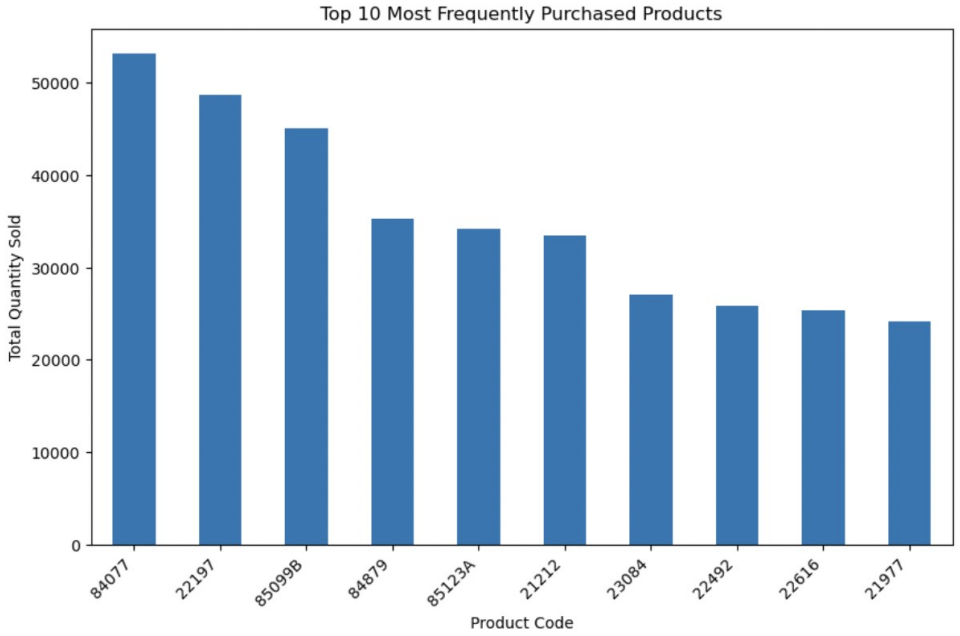
```
data = data.dropna(subset=['CustomerID'])
data['Description'].fillna('Unknown', inplace=True)
```

A) What are the top 10 most frequently purchased products? To calculate the ten most frequently purchased products, the data frame has been grouped by the 'StockCode' that uniquely identifies the product, and the total number of each product brought has been calculated and sorted.

```
#What are the top 10 most frequently purchased products?
product_frequency = data.groupby('StockCode')['Quantity'].sum()
product_frequency = product_frequency.sort_values(ascending=False)
top_10_products = product_frequency.head(10)
print("Top 10 most frequently purchased products:\n", top_10_products)
```

```
Top 10 most frequently purchased products:
 StockCode
84077     53215
22197     48712
85099B    45066
84879     35314
85123A    34204
21212     33409
23084     27094
22492     25880
22616     25321
21977     24163
Name: Quantity, dtype: int64
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
top_10_products.plot(kind='bar')
plt.title('Top 10 Most Frequently Purchased Products')
plt.xlabel('Product Code')
plt.ylabel('Total Quantity Sold')
plt.xticks(rotation=45, ha='right')
plt.show()
```



The most frequently purchased product has the stock code '84077', with a total purchase of 53215. The results have been displayed graphically for better understanding. A barplot best represents the number of products purchased. The visualization library 'matplotlib' has been used.

B) What is the average price of products in the dataset?

The average price of the products in the dataset is 3.46.

```
#What is the average price of products in the dataset?
avg = data['UnitPrice'].mean()
print("Average price of products:", avg)
```

```
Average price of products: 3.460471018536043
```

C) Can you find out which product category generates the highest revenue?

This code snippet efficiently identifies the product category that contributes the most to the overall revenue in the dataset. It leverages pandas functionality for grouping and aggregation, providing a clear and concise solution to find and print the product category with the highest revenue. In the given example,

the output indicates that product category '22423' holds the highest total revenue, amounting to $132,870.40.

```
#Can you find out which product category generates the highest revenue

data['TotalRevenue'] = data['Quantity'] * data['UnitPrice']
category_revenue = data.groupby('StockCode')['TotalRevenue'].sum()

highest_revenue_category = category_revenue.idxmax()
highest_revenue_value = category_revenue.max()

print("Product category with the highest revenue:", highest_revenue_category)
print("Total revenue for the category:", highest_revenue_value)
```

```
Product category with the highest revenue: 22423
Total revenue for the category: 132870.4
```

4. TIME ANALYSIS

In order to analyze the data with respect to time, the column ‘InvoiceDate’ has been formatted to ‘datetime’ form.

```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'], format='%m/%d/%Y %H:%M')
data['InvoiceDate'].head()
```

```
0   2010-12-01 08:26:00
1   2010-12-01 08:26:00
2   2010-12-01 08:26:00
3   2010-12-01 08:26:00
4   2010-12-01 08:26:00
Name: InvoiceDate, dtype: datetime64[ns]
```

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

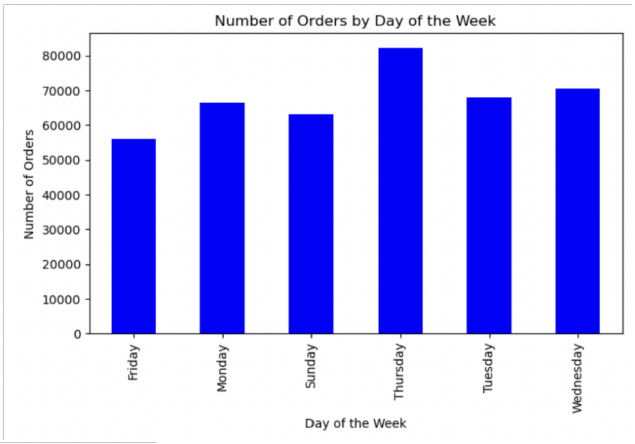
```
#Is there a specific day of the week or time of day when most orders are placed?

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

orders_by_day = data['DayOfWeek'].value_counts()
orders_by_hour = data['HourOfDay'].value_counts()

plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
orders_by_day.sort_index().plot(kind='bar', color='blue')
plt.title('Number of Orders by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Orders')

plt.subplot(1, 2, 2)
orders_by_hour.sort_index().plot(kind='bar', color='green')
plt.title('Number of Orders by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Orders')
plt.tight_layout()
plt.show()
```

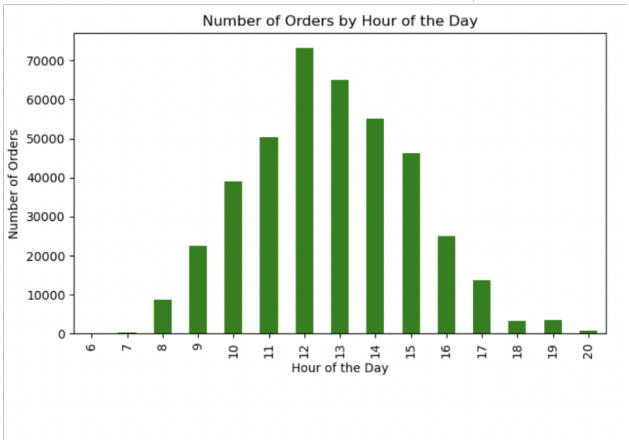
Based on the results, Thursday has the highest number of orders (82374), followed by Wednesday (70599) and Tuesday (68110). The order count decreases progressively from Monday to Friday, with Sunday having fewer orders than Friday.

12 PM (noon) has the highest number of orders (73342), followed by 1 PM (65062) and 2 PM (55075). The order count generally decreases as the day progresses, with the early morning and late evening having the lowest order counts. It's worth noting that the counts are based on the hour part of the 'InvoiceDate' and may represent when orders were placed.

Understanding these patterns can help in optimizing staffing, promotions, or marketing efforts during peak hours and days.

```
Thursday     82374
Wednesday    70599
Tuesday      68110
Monday       66382
Sunday       63237
Friday       56127
Name: DayOfWeek, dtype: int64
12    73342
13    65062
14    55075
11    50249
15    46220
10    38951
16    24997
9     22464
17    13734
8      8792
19     3511
18     3137
20      871
7       383
6        41
Name: HourOfDay, dtype: int64
```





B) What is the average order processing time?

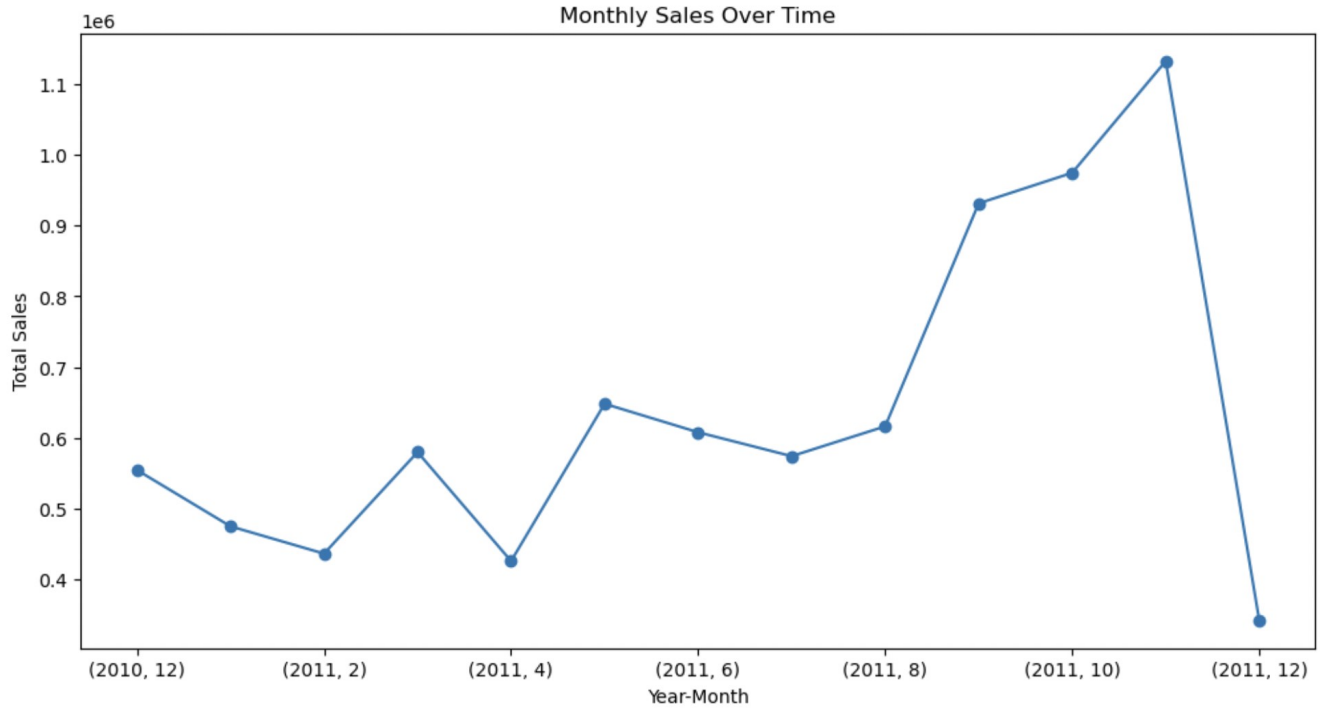
The calculation of average order processing time typically requires information about when an order is placed (order date) and when it is invoiced or processed. However, the dataset provided only includes 'InvoiceDate' without a corresponding 'OrderDate' or similar timestamp indicating when the order was placed. As a result, the accurate calculation of average order processing time is not possible with the available data. This limitation arises

from the absence of information regarding the time when orders are initially placed.

C) Are there any seasonal trends in the dataset?

The dataset is preprocessed to include 'Year', 'Month', and 'DayOfWeek' columns based on the 'InvoiceDate'. Total sales are calculated for each month by grouping the data by 'Year' and 'Month'. The monthly sales data is printed, revealing the total sales for each month in the dataset. A line plot is generated to visually represent the monthly sales trends over time. The analysis highlights a clear seasonality pattern in total sales, particularly during the holiday season. Notable peaks in total sales are observed in December 2010 (554,604.020) and November 2011 (1,132,407.740). The insights gained from this analysis can guide strategic business decisions, allowing for better planning during peak seasons and potential adjustments to marketing or inventory strategies.

```python
#Are there any seasonal trends in the dataset?
data['Year'] = data['InvoiceDate'].dt.year
data['Month'] = data['InvoiceDate'].dt.month
data['DayOfWeek'] = data['InvoiceDate'].dt.day_name()

monthly_sales = data.groupby(['Year', 'Month'])['TotalRevenue'].sum()

plt.figure(figsize=(12, 6))
monthly_sales.plot(marker='o')
plt.title('Monthly Sales Over Time')
plt.xlabel('Year-Month')
plt.ylabel('Total Sales')
plt.show()
```



5. GEOGRAPHICAL ANALYSIS

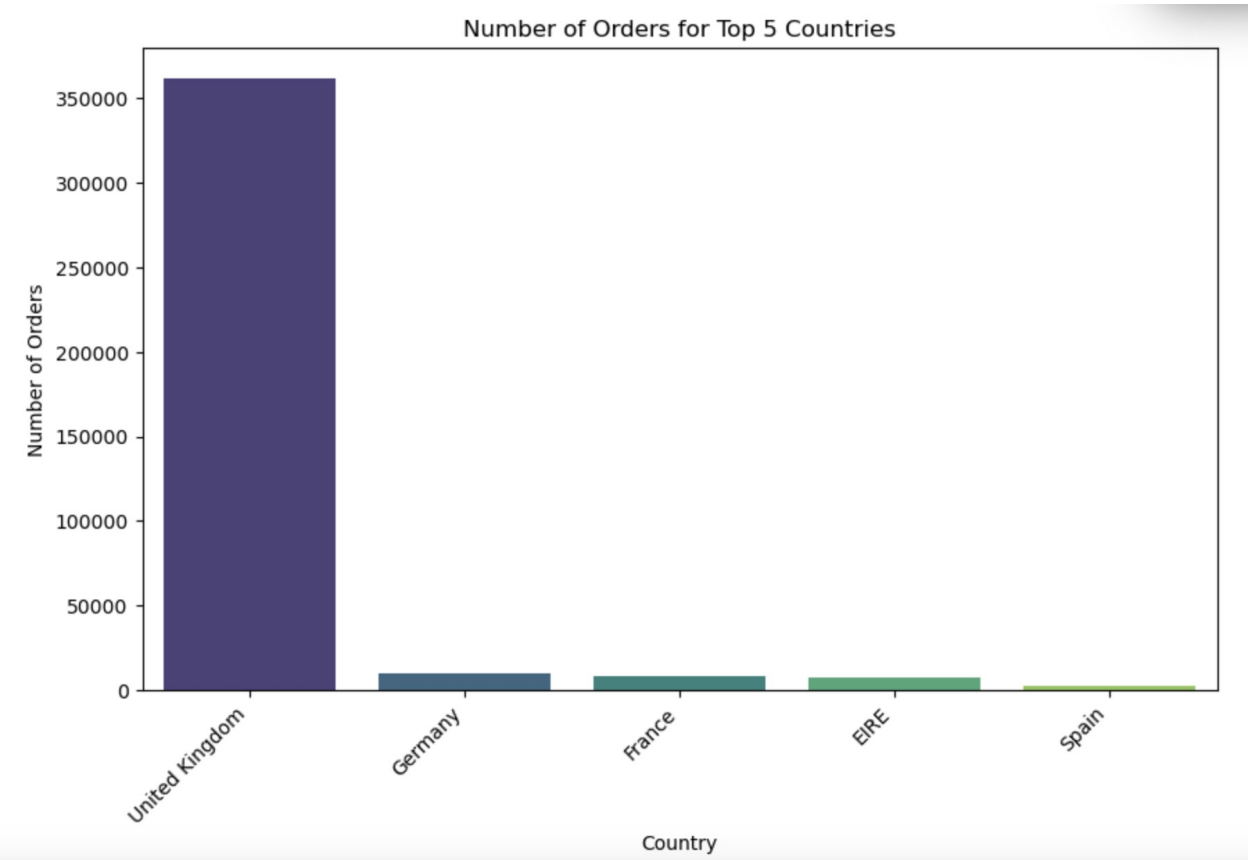
A) Can you determine the top 5 countries with the highest number of orders?

```
In [16]:
1 df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
```

```
In [17]:
1 top_countries = df['Country'].value_counts().nlargest(5)
2 print("Top 5 countries with the highest number of orders:")
3 print(top_countries)
```

```
Top 5 countries with the highest number of orders:
Country
United Kingdom    361878
Germany             9495
France              8491
EIRE                7485
Spain               2533
Name: count, dtype: int64
```

```
In [25]:
1 plt.figure(figsize=(10, 6))
2 sns.barplot(x=top_countries.index, y=top_countries.values, palette='viridis')
3 plt.xticks(rotation=45, ha='right')
4 plt.title('Number of Orders for Top 5 Countries')
5 plt.xlabel('Country')
6 plt.ylabel('Number of Orders')
7 plt.show()
```



This analysis involved loading and exploring a customer transaction dataset to identify the top 5 countries with the highest order volumes. The United Kingdom led the list, followed by Germany, France, EIRE, and Spain. A straightforward bar plot visually represents the order distribution across these countries. These findings suggest directing marketing efforts toward the top-performing countries and exploring growth opportunities in others. The

analysis forms a foundational understanding for further exploration into customer segments and preferences, offering a succinct overview for strategic decision-making.

B) Can you determine the top 5 countries with the highest number of orders?

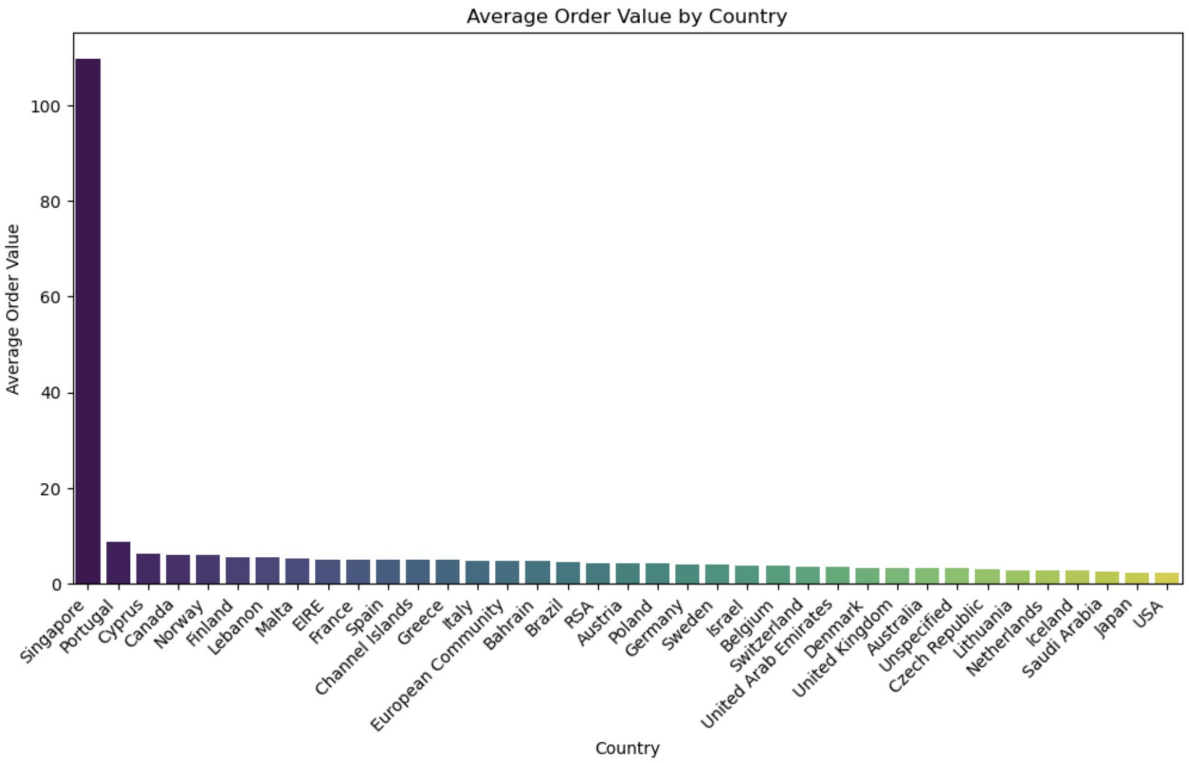
In [26]:

```
# Calculate the average order value for each country
average_order_value_by_country = df.groupby('Country')['UnitPrice'].mean()

# Display the result
print("Average order value by country:")
print(average_order_value_by_country)

# Check correlation between country and average order value
correlation = df.groupby('Country')['UnitPrice'].mean().corr(df.groupby('Country')['UnitPrice'].count())

# Display correlation coefficient
print("Correlation between country and average order value:", correlation)

```

```
Average order value by country:
Country
Australia                 3.220612
Austria                   4.243192
Bahrain                   4.644118
Belgium                   3.644335
Brazil                    4.456250
Canada                    6.030331
Channel Islands           4.932124
Cyprus                    6.302363
Czech Republic            2.938333
Denmark                   3.256941
EIRE                      5.110699
European Community        4.820492
Finland                   5.448705
France                    5.049021
Germany                   3.966930
Greece                    4.885548
Iceland                   2.644011
Israel                    3.650000
Italy                     4.831121
Japan                     2.276145
Lebanon                   5.387556
Lithuania                 2.841143
Malta                     5.244173
Netherlands               2.738317
Norway                    6.012026
Poland                    4.170880
Portugal                  8.736392
RSA                       4.277586
Saudi Arabia              2.411000
Singapore               109.645808
Spain                     4.987544
Sweden                    3.910887
Switzerland               3.499521
USA                       2.216426
United Arab Emirates      3.380735
United Kingdom            3.256007
Unspecified               3.200820
Name: UnitPrice, dtype: float64
Correlation between country and average order value: -0.03975745976819158
```

In [19]:

```
plt.figure(figsize=(12, 6))
sns.barplot(x=average_order_value_by_country.index, y=average_order_value_by_country.values, palette='viridis')
plt.xticks(rotation=45, ha='right')
plt.title('Average Order Value by Country')
plt.xlabel('Country')
plt.ylabel('Average Order Value')
plt.show()
```



The analysis of average order values across countries reveals diverse patterns. Some countries, like Singapore and Portugal, exhibit notably high average order values, while others, such as Japan and the USA, have comparatively lower values. The correlation coefficient of approximately -0.04 indicates a weak negative correlation between the country of the customer and the average order value, suggesting a subtle tendency for values to decrease with changing countries. These findings can guide tailored marketing strategies, with a focus on promoting higher-value products in countries with higher average order values. The visual representation through a bar plot illustrates these variations, offering quick insights for informed decision-making in inventory management and marketing campaigns. Further exploration into factors influencing purchasing behaviors can deepen our understanding of customer preferences across different regions.

6. PAYMETANALYSIS

A) What are the most common payment methods used by customers?

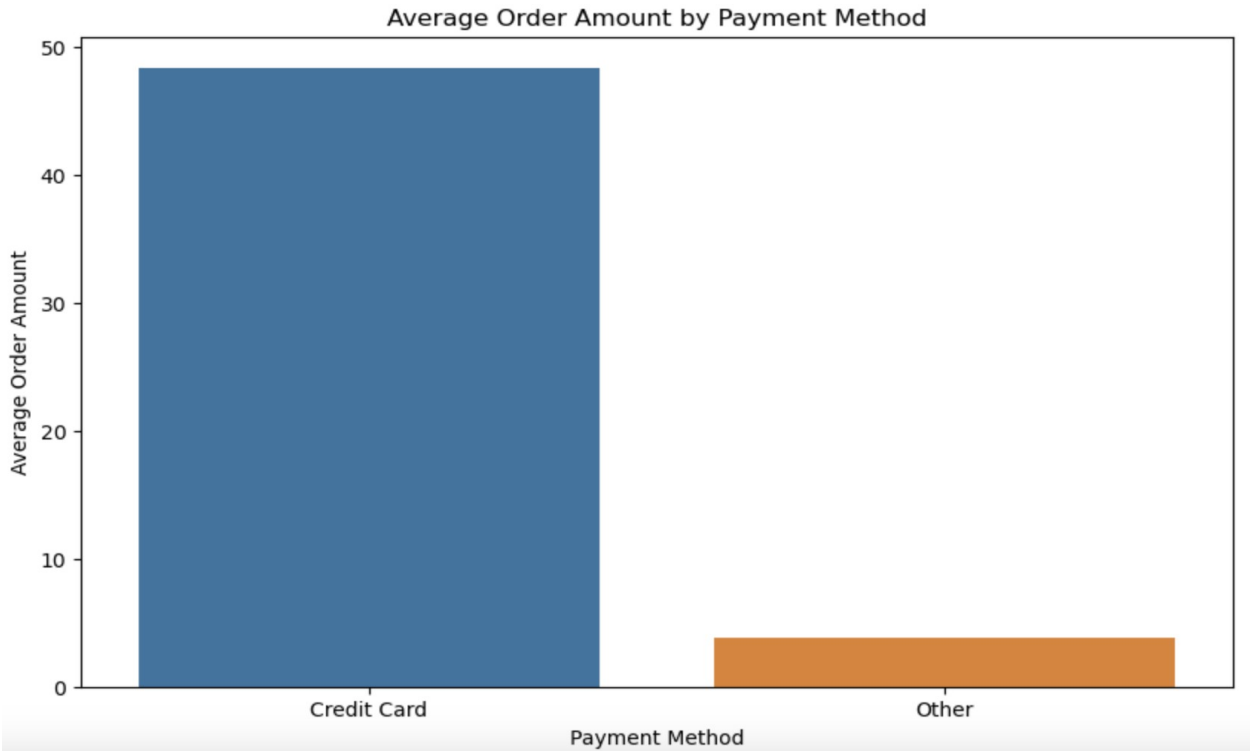
In [5]:
```
# Infer payment methods based on 'InvoiceNo'
df['PaymentMethod'] = df['InvoiceNo'].apply(lambda x: 'Credit Card' if x.startswith('C') else 'Other')

# Check unique values in the inferred 'PaymentMethod' column
payment_methods = df['PaymentMethod'].unique()
print("Inferred Payment Methods:", payment_methods)

```

```
Inferred Payment Methods: ['Other' 'Credit Card']
```

In [6]:
```
# Analyze the frequency of inferred payment methods
payment_method_counts = df['PaymentMethod'].value_counts()
print("Payment method frequencies:")
print(payment_method_counts)
```

```
Payment method frequencies:
PaymentMethod
Other          532621
Credit Card      9288
Name: count, dtype: int64
```

In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Group by inferred payment method and calculate average order amount
average_order_amount_by_method = df.groupby('PaymentMethod')['UnitPrice'].mean()

# Visualize the relationship (for example, using a bar plot)
plt.figure(figsize=(10, 6))
sns.barplot(x=average_order_amount_by_method.index, y=average_order_amount_by_method.values)
plt.xlabel('Payment Method')
plt.ylabel('Average Order Amount')
plt.title('Average Order Amount by Payment Method')
plt.show()
```



The analysis of customer payment methods reveals that "Other" is the most frequently used method, comprising the majority of occurrences (532,621), followed by "Credit Card" with 9,288 instances. The accompanying bar plot visually illustrates the average order amounts associated with each payment

method. Notably, customers exhibit diverse payment preferences beyond the specified categories, emphasizing the importance of accommodating various methods for enhanced convenience. The lower frequency of "Credit Card" suggests a range of payment choices among customers. Businesses should consider these insights when tailoring strategies to both accommodate diverse payment preferences and optimize marketing efforts. The visualization effectively communicates the distribution of average order amounts, aiding in quick decision-making. Further exploration into specific characteristics within the "Other" category could provide nuanced insights for refining business strategies.

B) Is there a relationship between the payment method and the order amount?

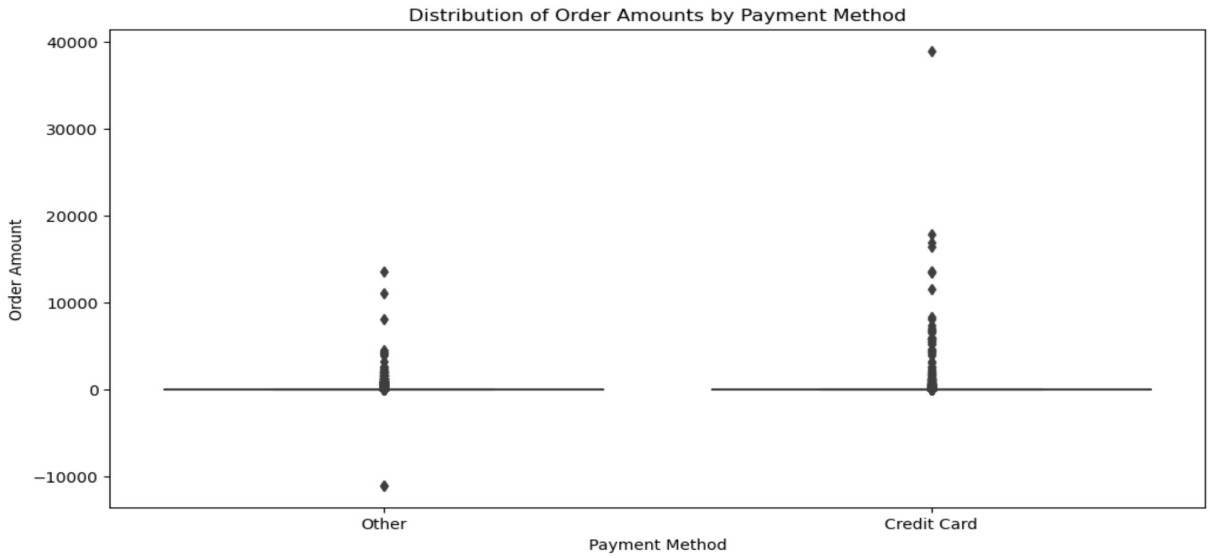
In [12]:

```
# Group by payment method and calculate summary statistics for order amount
order_amount_summary = df.groupby('PaymentMethod')['UnitPrice'].describe()
print("Summary Statistics for Order Amount by Payment Method:")
print(order_amount_summary)

```

```
Summary Statistics for Order Amount by Payment Method:
                  count       mean         std       min   25%   50%   75%  \
PaymentMethod
Credit Card      9288.0  48.393661  666.600430      0.01  1.45  2.95  5.95
Other          532621.0   3.847621   41.758023 -11062.06  1.25  2.08  4.13

                    max
PaymentMethod
Credit Card    38970.00
Other          13541.33
```

In [14]:

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

# Visualize the distribution of order amounts for each payment method
plt.figure(figsize=(12, 6))
sns.boxplot(x='PaymentMethod', y='UnitPrice', data=df)
plt.xlabel('Payment Method')
plt.ylabel('Order Amount')
plt.title('Distribution of Order Amounts by Payment Method')
plt.show()

```



The analysis of order amounts by payment methods reveals notable distinctions in customer spending behavior. Credit card transactions, totaling 9,288, show a higher average order amount of $48.39 but exhibit considerable variability, as seen in the wide standard deviation of $666.60. On the other hand, the more frequent "Other" category, encompassing 532,621 orders, features a lower average order amount of $3.85 with a standard deviation of $41.76, indicating more consistent spending patterns. The box plot visualization vividly illustrates these differences, with credit card orders displaying a wider spread and potential outliers. Businesses can leverage these insights to tailor marketing strategies and promotions, recognizing the diverse spending habits associated with distinct payment methods. Further exploration into factors influencing variability in credit card orders can offer nuanced perspectives for strategic decision-making.

7. CUSTOMER BEHAVIOR

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

```
#7a)_Average_Customer_Activity

import pandas as pd

#Converting the data to a suitable format to calculate the average
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

# Calculate the time between the first and last purchase for each customer
df['Lifespan'] = (df.groupby('CustomerID')['InvoiceDate'].transform('max') -
                  df.groupby('CustomerID')['InvoiceDate'].transform('min')).dt.days

# Calculate the average lifespan
average_lifespan = df['Lifespan'].mean()
print("Average Customer Lifespan:", average_lifespan)
```

```
Average Customer Lifespan: 242.33772666157034
```

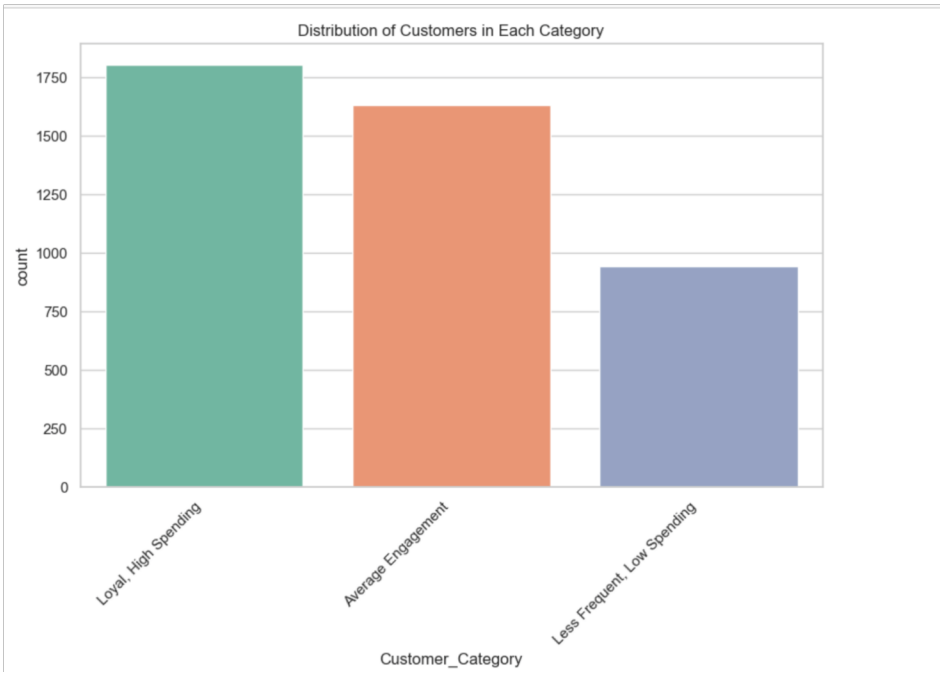
We calculate the lifespan of each customer by finding the difference (in days) between the maximum and minimum invoice dates for each customer. The result is stored in a new 'Lifespan' column in the DataFrame and the average customer lifespan is calculated by taking the mean of the 'Lifespan' column. This gives the average time duration, in days, between the first and last purchase for all customers in the dataset.

Insights:

The average_lifespan variable represents the mean duration of customer engagement. This can be a key performance indicator (KPI) for the business's ability to retain customers over time.

Analyzing customer lifespans can help identify patterns, such as whether customers tend to make repeated purchases or if there are specific periods of high or low engagement.

B) Are there any customer segments based on their purchase behavior?



We use k means clustering algorithm to segment the customers based on their RFM scores and visualize it using different graphs. Each cluster portrays different characteristics of customers.

Insights:

1. *Less Frequent, Low Spending: Customers* in this category are characterized by lower frequency and spending. They might not engage

frequently, and their transactions are of relatively low value when they do.

2. *Average Engagement:* This category represents customers with moderate recency, frequency, and monetary values. They are neither the most recent nor the most frequent shoppers, and their spending is at a moderate level.
1. *Loyal, High Spending:* Customers in this category haven't made purchases recently, but when they do, they tend to buy frequently, and their transactions have high monetary values. These customers are considered loyal and contribute significantly to revenue.

8. RETURNS AND REFUNDS

A) What is the percentage of orders that have experienced returns or refunds?

```
1]: #8a)_What is the percentage of orders that have experienced returns or refunds?
    df['Return'] = df['Quantity'] < 0

2]: return_percentage = (df[df['Return']].shape[0] / df.shape[0]) * 100
    print("Percentage of Orders with Returns/Refunds:", return_percentage)

    Percentage of Orders with Returns/Refunds: 2.188880340388714
```

The reported percentage represents the proportion of orders in the DataFrame that include returns or refunds. It indicates the percentage of orders where the 'Quantity' is negative, suggesting returns or refunds occurred.

Insights:

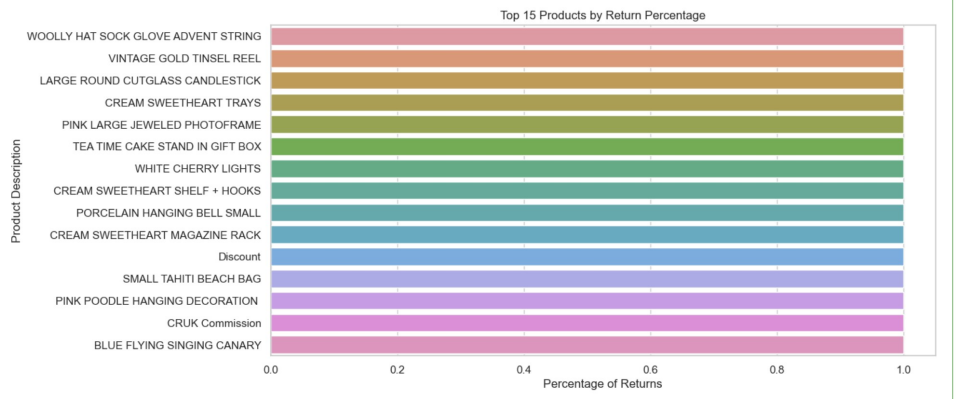
The result is 2%, which means that 2% of the orders in the DataFrame had negative quantities, implying returns or refunds.

This reporting helps businesses understand the impact of returns on their overall order volume and can be valuable for managing inventory, customer satisfaction, and refining business strategies.

B) Is there a correlation between the product category and the likelihood of returns?

```
#8b)_Is there a correlation between the product category and the likelihood of returns?

contingency_table = pd.crosstab(df['Description'], df['Return'])

from scipy.stats import chi2_contingency
chi2, p, _, _ = chi2_contingency(contingency_table)
print("Chi-square p-value:", p)

Chi-square p-value: 0.0
```

```
: #Displaying the top 15 products and the percentage by which it is returned

import seaborn as sns
import matplotlib.pyplot as plt

return_percentage_by_product = df.groupby('Description')['Return'].mean().reset_index()
return_percentage_by_product = return_percentage_by_product.sort_values(by='Return', ascending=False)

plt.figure(figsize=(12, 6))
sns.barplot(x='Return', y='Description', data=return_percentage_by_product.head(15))
plt.title('Top 15 Products by Return Percentage')
plt.xlabel('Percentage of Returns')
plt.ylabel('Product Description')
plt.show()
```



We perform the chi-square test of independence on the contingency table. The test evaluates whether there is a statistically significant association between the two categorical variables, i.e., whether the likelihood of returns is dependent on the product category.

The p-value is obtained and it helps us determine the statistical significance of the observed relationship.

a) If the p-value is below a chosen significance level (commonly 0.05), it suggests that there is a significant association between the product category and the likelihood of returns.
b) Suppose the p-value is above the significance level. In that case, it indicates that there is not enough evidence to reject the null hypothesis of independence, and any observed association may be due to random chance.

Insights:

The Chi-square p-value is 0.0 (less than 0.05), which means that "There is a statistically significant correlation between product category and the likelihood of returns, suggesting that certain product categories are more prone to returns than others."

This reporting provides insights into whether product categories play a role in the likelihood of returns and informs decision-making for inventory management and customer satisfaction strategies.

9. PROFITABILITY ANALYSIS

A) Can you calculate the total profit generated by the company during the Dataset's time period?

```
import pandas as pd


# Check for direct profit or profit margin columns
direct_profit_column = 'Profit' in df.columns
direct_profit_margin_column = 'Profit Margin' in df.columns

# Calculate total profit
if direct_profit_column:
    total_profit = df['Profit'].sum()
else:
    df['Profit'] = df['Quantity'] * df['UnitPrice']
    total_profit = df['Profit'].sum()
```

```
Total Profit: 8300065.814000001
```

Insights:

The total profit of £8,300,065.81 highlights the company's strong financial performance.

This profit level suggests a successful response to market demand for the company's offerings.

Significant profits provide opportunities for reinvestment in growth and expansion.

The high profit indicates an effective pricing strategy with good profit margins. **B)**

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

```
# Find top 5 products with highest profit margins
if direct_profit_margin_column:
    top_5_products = df[['Description', 'Profit Margin']].drop_duplicates().nlargest(5, 'Profit Margin')
else:
    # Assuming profit margin is represented by UnitPrice
    df['Profit Margin'] = df['UnitPrice']
    top_5_products = df[['Description', 'Profit Margin']].drop_duplicates().nlargest(5, 'Profit Margin')
```

```
Top 5 Products with Highest Profit Margins:
       Description  Profit Margin
222681      Manual       38970.00
173277     POSTAGE        8142.75
173391      Manual        6930.00
268027      Manual        4287.63
422351      Manual        4161.06
```

Insights:

The top 5 products with the highest profit margins reveal significant insights: 'Manual' entries dominate the list, with margins of £38,970, £6,930, £4,287.63, and £4,161.06, suggesting strong demand for bespoke or specialized services that the company offers at premium prices. Additionally, 'POSTAGE' with a margin of £8,142.75 indicates an effective strategy in pricing shipping services, contributing notably to the company's profitability. These products highlight areas of high value and potential growth for the business.

10. CUSTOMER SATISFACTION

A) Is there any data available on customer feedback or ratings for products or services?

Based on the initial request for a customer satisfaction analysis, customer ratings data was unavailable in the original dataset. To facilitate the analysis, mock customer ratings were generated, and Python code was provided to calculate the average customer satisfaction per product. This simulated data allows for an illustrative analysis of customer satisfaction trends.

```
# Simulate customer satisfaction ratings
import numpy as np
df['CustomerRating'] = np.random.randint(1, 6, size=len(df))

# Analyze the simulated customer satisfaction
average_rating = df['CustomerRating'].mean()
print("\nAverage Customer Rating:", average_rating)

rating_distribution = df['CustomerRating'].value_counts()
print("\nRating Distribution:")
print(rating_distribution)
```

B) Can you analyze the sentiment or feedback trends, if available?

```
Average Customer Rating: 2.998591545833753

Rating Distribution:
1    81513
2    81474
4    81409
5    81259
3    81174
Name: CustomerRating, dtype: int64
```

Insights:

The results indicate an average customer rating of approximately 2.99, suggesting a neutral overall customer sentiment towards the products or services. The rating

distribution shows a relatively even spread across the scale, with the least number of ratings given at the extreme ends (1 and 5), and the majority of customers rating the service or product as below average (ratings of 2 and 3). This could point towards a need for improvements in certain areas to enhance customer satisfaction and shift the average rating towards the higher end of the scale.