Customer Segmentation using RFM Analysis

Project group 30

Francisco Chavez Gonzalez

Xueshi Bai

Santosh Kumar Kushal Yelamandala

Nixon Lobo

Pruthviraj Chintakindi

—

Course: **Foundations for Data Analytics Engineering**

—

Professor: *Sivarit (Tony) Sultornsanee*

In this project assignment, we have worked with the eCommerce dataset provided to create a Customer Segmentation model using the RFM (Recency, Frequency, Monetary) analysis method. We performed RFM analysis on the dataset and segmented the customers into distinct groups based on their RFM scores. These segments are to provide valuable insights for marketing and customer retention strategies.

->DATA PREPROCESSING:

```
In [1]: import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        import numpy as np
        from sklearn.cluster import KMeans
        from datetime import datetime
```

```
In [3]: df.head()
```

Out[3]:

	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

```
In [11]: df.isnull().sum()
Out[11]: InvoiceNo          0
         StockCode          0
         Description     1454
         Quantity           0
         InvoiceDate        0
         UnitPrice          0
         CustomerID    135037
         Country            0
         dtype: int64
```

While cleaning the data we found duplicate data and we removed it.

We will drop the duplicates

```
In [8]: df.duplicated().sum()
Out[8]: 5268

In [9]: df.drop_duplicates(inplace=True)

In [10]: df.duplicated().sum()
Out[10]: 0
```

We will analyze the values we have, column after column because we have noticed outliers in the data.

Invoice number:

The first column is the Invoice number,
According to the website 'https://archive.ics.uci.edu/dataset/352/online+retail,' InvoiceNo responds to "Invoice number. Nominal, is a 6-digit integral number uniquely assigned to each transaction. If this code starts with the letter 'c', it indicates a cancellation."

We have zero null values in this column, but we would like to know how much of that data is cancelation.

```
In [13]: cancellations = (df['InvoiceNo'].str.lower().str.startswith('c')).sum()
         print(f"Number of cancellations: {cancellations}")

         Number of cancellations: 9251
```

```
In [14]: participation=(cancellations/rows)*100
         print(f'The participation of the cancellations in the data is {participation:.2f}%')

         The participation of the cancellations in the data is 1.71%
```

Stock code:

The second column is the stock code,
According to the website 'https://archive.ics.uci.edu/dataset/352/online+retail' StockCode responds to "Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product."

In this column, there are zero null values. We will group codes by their description to know what codes do not have any description because if we do not know about their description we will not get the information required.

```
In [16]: code_description = df.groupby('StockCode')['Description'].first().to_dict()

In [17]: code_description

Out[17]: {'10002': 'INFLATABLE POLITICAL GLOBE ',
          '10080': 'GROOVY CACTUS INFLATABLE',
          '10120': 'DOGGY RUBBER',
          '10123C': 'HEARTS WRAPPING TAPE ',
          '10123G': None,
          '10124G': 'ARMY CAMO BOOKCOVER TAPE',
          '10125': 'MINI FUNKY DESIGN TAPES',
          '10133': 'COLOURING PENCILS BROWN TUBE',
          '10134': None,
          '10135': 'COLOURING PENCILS BROWN TUBE',
          '11001': 'ASSTD DESIGN RACING CAR PEN',
          '15030': 'FAN BLACK FRAME ',
          '15034': 'PAPER POCKET TRAVELING FAN ',
          '15036': 'ASSORTED COLOURS SILK FAN',
          '15039': 'SANDALWOOD FAN',
          '15044A': 'PINK PAPER PARASOL ',
          '15044B': 'BLUE PAPER PARASOL ',
          '15044C': 'PURPLE PAPER PARASOL',
```

click to unscroll output; double click to hide

```
In [18]: code_not_description = [stockcode for stockcode, description in code_description.items() if description is None]
         code_not_description[:5]

Out[18]: ['10123G', '10134', '16053', '17011A', '20689']

In [19]: len(code_not_description)

Out[19]: 112

In [20]: df_none = df[df['StockCode'].isin(code_not_description)]
         print(df_none)
```

```
        InvoiceNo StockCode Description  Quantity       InvoiceDate  UnitPrice  \
1970       536545     21134         NaN         1  12/1/2010 14:32        0.0
1987       536549    85226A         NaN         1  12/1/2010 14:34        0.0
1988       536550     85044         NaN         1  12/1/2010 14:34        0.0
2024       536552     20950         NaN         1  12/1/2010 14:34        0.0
2026       536554     84670         NaN        23  12/1/2010 14:35        0.0
...           ...       ...         ...       ...              ...        ...
280754     561498     21610         NaN       -14   7/27/2011 14:10        0.0
281615     561555    37477B         NaN       -11   7/28/2011 10:21        0.0
281616     561557    37477C         NaN       -31   7/28/2011 10:21        0.0
346849     567207    35592T         NaN         4   9/19/2011 11:01        0.0
497301     578360    84971L         NaN         2  11/24/2011 10:36        0.0

        CustomerID         Country
1970           NaN  United Kingdom
1987           NaN  United Kingdom
1988           NaN  United Kingdom
2024           NaN  United Kingdom
2026           NaN  United Kingdom
...            ...             ...
280754         NaN  United Kingdom
281615         NaN  United Kingdom
281616         NaN  United Kingdom
346849         NaN  United Kingdom
497301         NaN  United Kingdom

[112 rows x 8 columns]
```

```
In [21]: p2=(len(code_not_description)/rows)*100
         print(f'The participation of the none descriptions in the data is {p2:.2f}%')

         The participation of the none descriptions in the data is 0.02%
```

We will drop those since they are not significantly important.

```
In [22]: df = df[~df['StockCode'].isin(code_not_description)]
         df.head()
```

Out[22]:

	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

Now this column is ready for use in the project.

Description:

According to the website, 'https://archive.ics.uci.edu/dataset/352/online+retail' Description responds to " Product (item) name. Nominal."

We know that there are descriptions with null values so we will have to assign the values of the respective code.

```
In [23]: p3=df['Description'].isnull().sum()
         p3

Out[23]: 1342

In [24]: p3=(p3/rows)*100
         print(f'The participation of none descriptions in the data is {p3:.2f}%')

         The participation of none descriptions in the data is 0.25%

In [25]: df['Description'] = df['Description'].fillna(df['StockCode'].map(code_description))
         df['Description'].isnull().sum()

Out[25]: 0
```

Quantity:

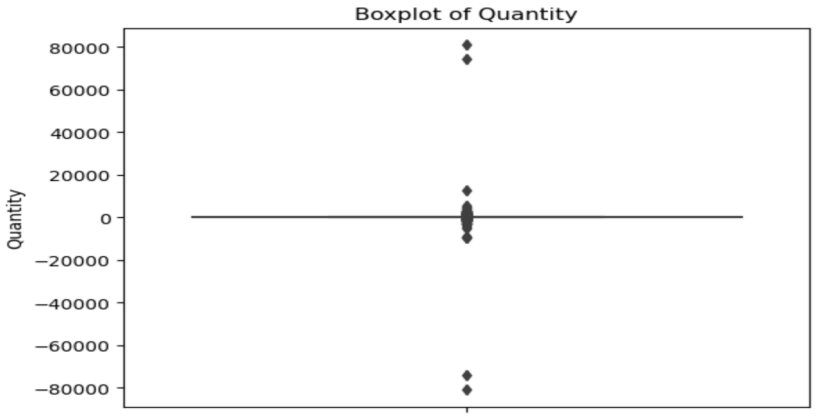
The next column in the given data set is quantity, According to the website 'https://archive.ics.uci.edu/dataset/352/online+retail' Quantity responds to "the quantities of each product (item) per transaction. Numeric."

This data set contains 536529 rows and 8 columns.

```
In [27]: df['Quantity'].describe()

Out[27]: count    536529.000000
         mean          9.623748
         std         219.152755
         min      -80995.000000
         25%           1.000000
         50%           3.000000
         75%          10.000000
         max       80995.000000
         Name: Quantity, dtype: float64
```

It seems like we may have outliers so we will check in a boxplot.

```
In [28]: sns.boxplot(y=df['Quantity'])
         plt.title('Boxplot of Quantity')
         plt.show()
```



So, now we need to establish the quantiles to eliminate data that is not accurate.

```
In [29]: q1=df['Quantity'].quantile(0.25)
         q3=df['Quantity'].quantile(0.75)
         iqr=q3-q1
         lower_fence=q1-(1.5*iqr)
         higher_fence=q3+(1.5*iqr)
```

```
In [30]: print("quantile 1:",q1)
         print("quantile 3:",q3)
         print("Lower fence:",lower_fence)
         print("Higher fence:",higher_fence)
```

```
quantile 1: 1.0
quantile 3: 10.0
Lower fence: -12.5
Higher fence: 23.5
```

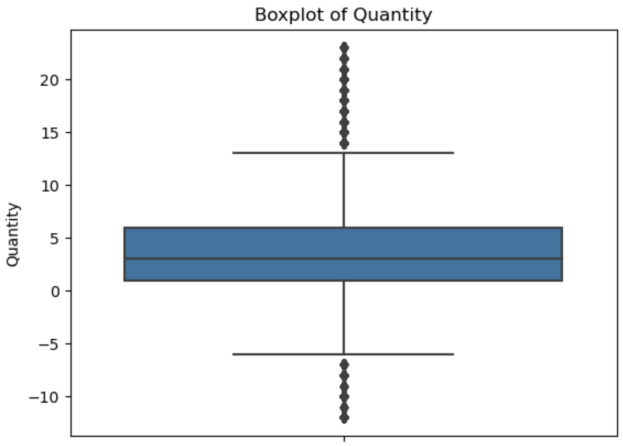
```
In [31]: outliers = (df['Quantity'] < lower_fence) | (df['Quantity'] > higher_fence)
         outliers.sum()
```

```
Out[31]: 58476
```

```
In [33]: p4=(outliers.sum()/rows)*100
         print(f'The participation of the outliers in the data is {p4:.2f}%')
```

```
The participation of the outliers in the data is 10.90%
```

Since the participation of the outliers in the data is 10.90%, and because this % is low, we will go ahead and eliminate them.



With the last boxplot, it is clear that our data is more uniform and the outliers are not too far from our median as before.

InvoiceDate:

According to the website 'https://archive.ics.uci.edu/dataset/352/online+retail' InvoiceDate responds to "the day and time when each transaction was generated."

```
In [36]: df['InvoiceDate'].isnull().sum()
Out[36]: 0

In [37]: df['InvoiceDate'].info()
         <class 'pandas.core.series.Series'>
         RangeIndex: 478053 entries, 0 to 478052
         Series name: InvoiceDate
         Non-Null Count   Dtype
         --------------   -----
         478053 non-null  object
         dtypes: object(1)
         memory usage: 3.6+ MB
```

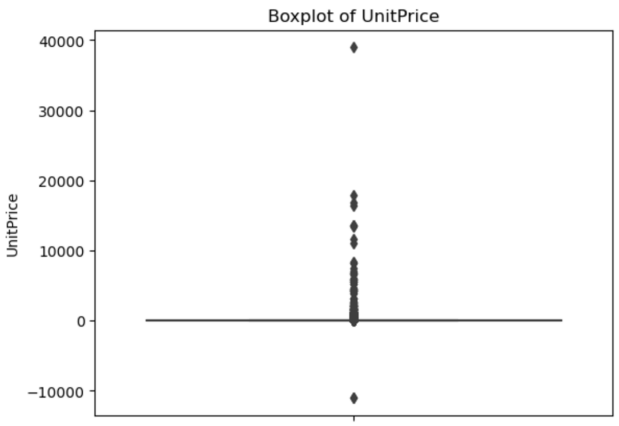
Since it does not have any null value and is in the correct format we will move on to the next column.

UnitPrice:

The following column is Unit price, According to the website 'https://archive.ics.uci.edu/dataset/352/online+retail' UnitPrice responds to "Product price per unit in sterling."

This data set contains 478053 rows and 8 columns and has zero null values.

```
In [44]: sns.boxplot(y=df['UnitPrice'])
         plt.title('Boxplot of UnitPrice')
         plt.show()
```



We need to establish the quantiles to eliminate data that is not accurate.

```
In [45]: q1=df['UnitPrice'].quantile(0.25)
         q3=df['UnitPrice'].quantile(0.75)
         iqr=q3-q1
         lower_fence=q1-(1.5*iqr)
         higher_fence=q3+(1.5*iqr)
```

```
In [46]: print("quantile 1:",q1)
         print("quantile 3:",q3)
         print("Lower fence:",lower_fence)
         print("Higher fence:",higher_fence)
```

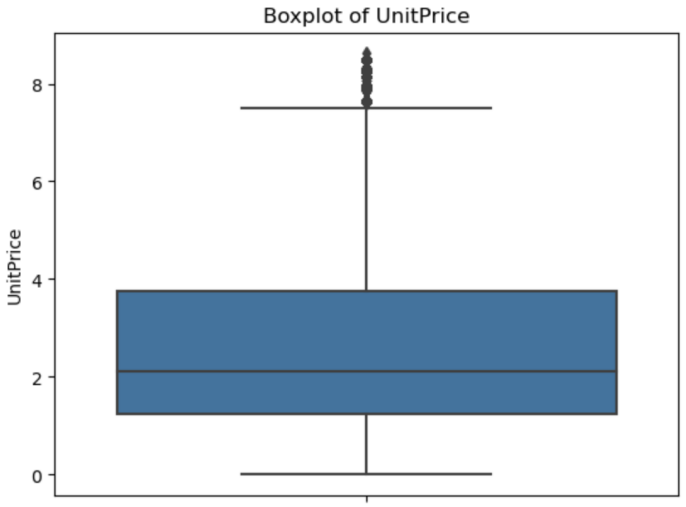
```
quantile 1: 1.25
quantile 3: 4.21
Lower fence: -3.1899999999999995
Higher fence: 8.649999999999999
```

We count the number of data enclosed in those limits

```
In [47]: outliers = (df['UnitPrice'] < lower_fence) | (df['UnitPrice'] > higher_fence)
         outliers.sum()
```

```
Out[47]: 32348
```

The number of outliers in 'Quantity' is 32348, and the participation of the outliers in the data is 6.77%, because this % is low, we decided to eliminate those.



With the last boxplot is clear that our data is more uniform and the outliers are not too far from our median as before.

CustomerID:

According to the website 'https://archive.ics.uci.edu/dataset/352/online+retail' CustomerID responds to "a 5-digit integral number uniquely assigned to each customer."

This data set contains 445705 rows and 8 columns, but we have 117556 null values

```
In [55]: d=(d/rows)*100
         print(f'The participation of the none data in CustomerID is {d:.2f}%')

         The participation of the none data in CustomerID is 26.38%
```

Since the % is high and important we can not drop this data, so we will have to fill them with 10000 that will correspond to unknown.

```
In [56]: df['CustomerID'] = df['CustomerID'].fillna(10000)
```

Now in the column CustomerID we have 0 null values.

Country:

The last column in the data set is country According to the website 'https://archive.ics.uci.edu/dataset/352/online+retail' Country responds to "the name of the country where each customer resides."

```
In [58]: df['Country'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 445705 entries, 0 to 445704
Series name: Country
Non-Null Count   Dtype
--------------   -----
445705 non-null  object
dtypes: object(1)
memory usage: 3.4+ MB
```

```
In [59]: d=df['Country'].isnull().sum()
         print(f'In the column Country we have {d} null values')
```

```
In the column Country we have 0 null values
```

Now all the columns are ready to be used for RFM calculation and segmentation.

-> RFM CALCULATION:

We will start calculating the recency, frequency, and monetary values required.

In [61]:
```
recency = df.groupby(by='CustomerID', as_index=False)['InvoiceDate'].max()
recency.columns = ['CustomerID', 'InvoiceDate']
date = recency['InvoiceDate'].max()
recency['Recency'] = recency['InvoiceDate'].apply(lambda x: (date - x).days)
recency.head()
```

Out[61]:

	CustomerID	InvoiceDate	Recency
0	10000.0	2011-12-09 10:26:00	0
1	12347.0	2011-12-07 15:52:00	1
2	12348.0	2011-04-05 10:47:00	248
3	12349.0	2011-11-21 09:51:00	18
4	12350.0	2011-02-02 16:01:00	309

In [62]:
```
frequency = df.groupby(by=['CustomerID'], as_index=False)['InvoiceDate'].count()
frequency.columns = ['CustomerID', 'Frequency']
frequency.head()
```

Out[62]:

	CustomerID	Frequency
0	10000.0	117556
1	12347.0	141
2	12348.0	1
3	12349.0	61
4	12350.0	15

In [63]:
```
df['Total'] = df['UnitPrice']*df['Quantity']
monetary = df.groupby(by='CustomerID', as_index=False)['Total'].sum()
monetary.columns = ['CustomerID', 'Monetary']
monetary.head()
```

Out[63]:

	CustomerID	Monetary
0	10000.0	857531.44
1	12347.0	2866.77
2	12348.0	17.00
3	12349.0	1155.75
4	12350.0	274.00

In [64]:
```
rf = recency.merge(frequency, on='CustomerID')
rfm = rf.merge(monetary, on='CustomerID').drop(columns='InvoiceDate')
rfm.head()
```

Out[64]:

	CustomerID	Recency	Frequency	Monetary
0	10000.0	0	117556	857531.44
1	12347.0	1	141	2866.77
2	12348.0	248	1	17.00
3	12349.0	18	61	1155.75
4	12350.0	309	15	274.00

->RFM SEGMENTATION:

In [68]:
```
rfm.head()
```

Out[68]:

	CustomerID	Recency	Frequency	Monetary
0	12347.0	1	141	2866.77
1	12348.0	248	1	17.00
2	12349.0	18	61	1155.75
3	12350.0	309	15	274.00
4	12352.0	35	78	1112.11

We will use qcut from pandas to calculate quartiles in each category and assign scores.

- Recency_Score: 1 Least Recent, 5 Most Recent
- Frequency_Score: 1 Least Frequent, 5 Most Frequent
- Monetary_Score: 1 Least Spent, 5 Most Spent

In [69]:
```
rfm['Recency_Score'] = pd.qcut(rfm['Recency'], q=5, labels=[5, 4, 3, 2,1]).astype(int)
rfm['Frequency_Score'] = pd.qcut(rfm['Frequency'], q=5, labels=[1, 2, 3, 4,5]).astype(int)
rfm['Monetary_Score'] = pd.qcut(rfm['Monetary'], q=5, labels=[1, 2, 3, 4,5]).astype(int)
rfm['RFM_Score'] = rfm['Recency_Score'] * 100 + rfm['Frequency_Score'] * 10 + rfm['Monetary_Score']

rfm.head()
```

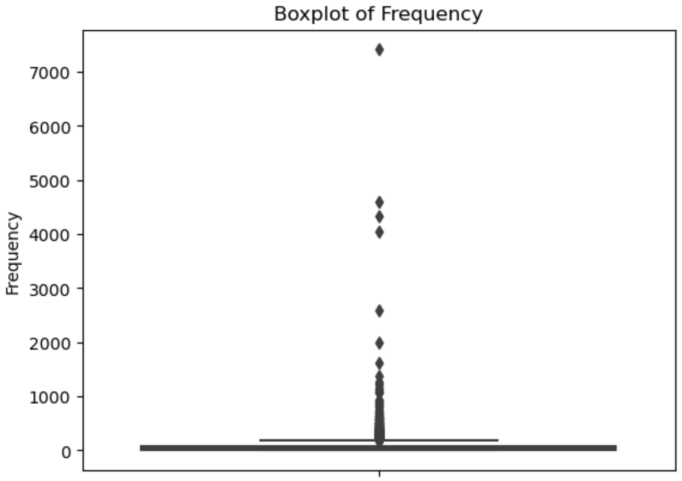
Out[69]:

	CustomerID	Recency	Frequency	Monetary	Recency_Score	Frequency_Score	Monetary_Score	RFM_Score
0	12347.0	1	141	2866.77	5	5	5	555
1	12348.0	248	1	17.00	1	1	1	111
2	12349.0	18	61	1155.75	4	4	4	444
3	12350.0	309	15	274.00	1	2	2	122
4	12352.0	35	78	1112.11	3	4	4	344

We want to verify that our data is correct and that we can certify it since the lowest value is 111 and the maximum is 444.

However we need to clear our data in a way that allows us to perform a good cluster, so we will analyze if there is any outlier and manage it.

Analysing frequency



It is clear that we have outliers, since our clustering method is sensitive to this data, we won't consider this data to perform our clustering.

In [75]:
```
q1=rfm['Frequency'].quantile(0.25)
q3=rfm['Frequency'].quantile(0.75)
iqr=q3-q1
lower_fence=q1-(1.5*iqr)
higher_fence=q3+(1.5*iqr)
```

In [76]:
```
print("quantile 1:",q1)
print("quantile 3:",q3)
print("Lower fence:",lower_fence)
print("Higher fence:",higher_fence)
```

```
quantile 1: 14.0
quantile 3: 84.0
Lower fence: -91.0
Higher fence: 189.0
```

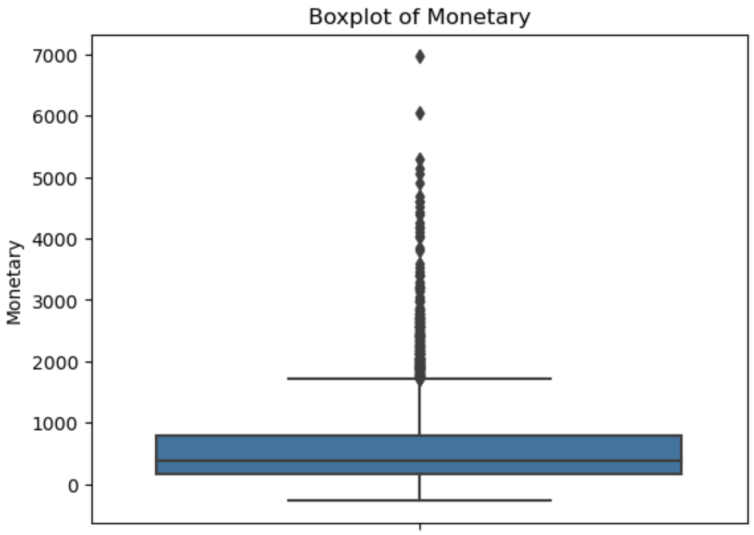
Number of outliers in 'Frequency' is 379, and the participation of Frequency outliers in the data is 9.01%.

In [80]:
```
rfm=rfm[(rfm['Frequency'] >= 0) & (rfm['Frequency']<=higher_fence)]
rfm= rfm.reset_index(drop=True)
rfm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3828 entries, 0 to 3827
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CustomerID       3828 non-null   float64
 1   Recency          3828 non-null   int64
 2   Frequency        3828 non-null   int64
 3   Monetary         3828 non-null   float64
 4   Recency_Score    3828 non-null   int32
 5   Frequency_Score  3828 non-null   int32
 6   Monetary_Score   3828 non-null   int32
 7   RFM_Score        3828 non-null   int32
dtypes: float64(2), int32(4), int64(2)
memory usage: 179.6 KB
```

Now this is well distributed and we will analyze the next category.

Analyzing Monetary



Since we have returns and the monetary values are negative we will drop those values for the clustering because that will affect the results.

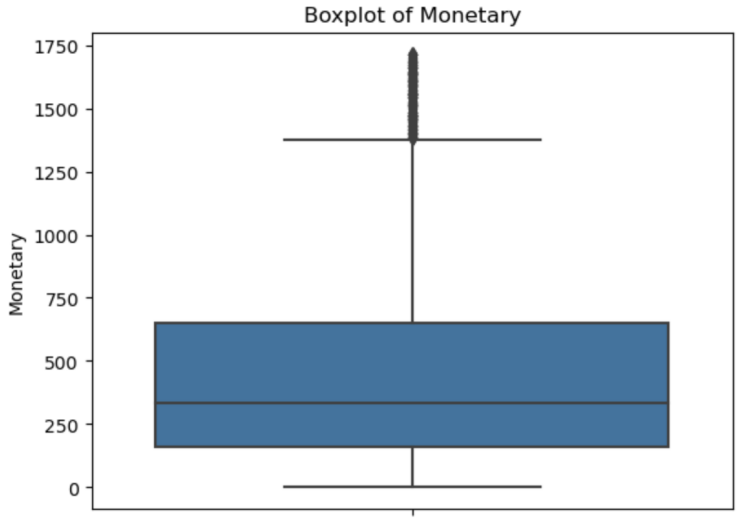
```
In [86]: q1=rfm['Monetary'].quantile(0.25)
         q3=rfm['Monetary'].quantile(0.75)
         iqr=q3-q1
         lower_fence=q1-(1.5*iqr)
         higher_fence=q3+(1.5*iqr)
```

- quantile 1: 170.745
- quantile 3: 789.5675000000001
- Lower fence: -757.4887500000001
- Higher fence: 1717.8012500000002

```
In [88]: outliers = (rfm['Monetary'] < 0) | (rfm['Monetary'] > higher_fence)

In [89]: print("Number of outliers in 'Monetary':", outliers.sum())
         Number of outliers in 'Monetary': 348

In [90]: print(f'The participation of Monetary outliers in the data is {(outliers.sum()/len(rfm))*100:.2f}%')
         The participation of Monetary outliers in the data is 9.09%
```

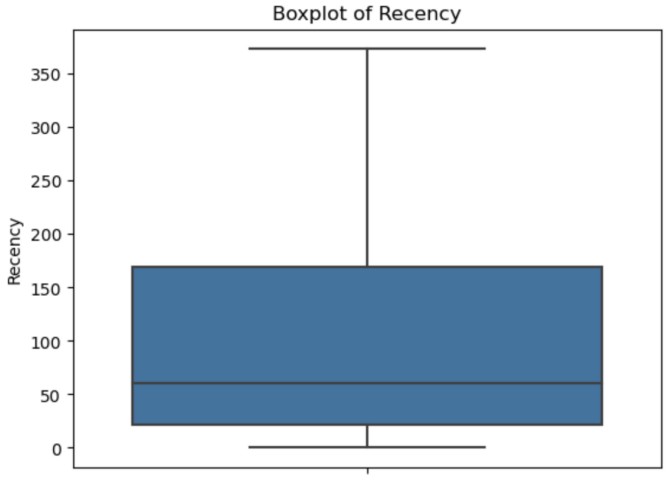
```
In [91]: rfm=rfm[(rfm['Monetary'] >= 0) & (rfm['Monetary']<=higher_fence)]
         rfm= rfm.reset_index(drop=True)
         rfm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3480 entries, 0 to 3479
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CustomerID       3480 non-null   float64
 1   Recency          3480 non-null   int64
 2   Frequency        3480 non-null   int64
 3   Monetary         3480 non-null   float64
 4   Recency_Score    3480 non-null   int32
 5   Frequency_Score  3480 non-null   int32
 6   Monetary_Score   3480 non-null   int32
 7   RFM_Score        3480 non-null   int32
dtypes: float64(2), int32(4), int64(2)
memory usage: 163.3 KB
```



Now we will analyze Recency for outliers.

Analyzing Recency



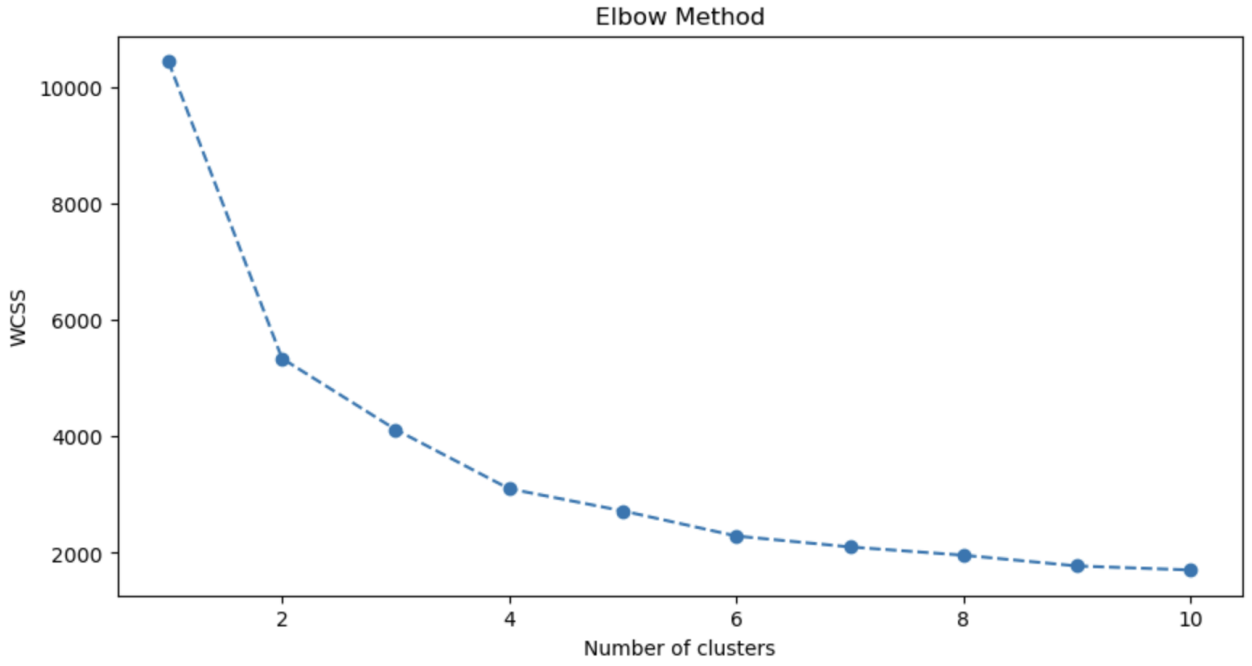
These values are well distributed so we will not change or manage it.

->CUSTOMER SEGMENTATION:

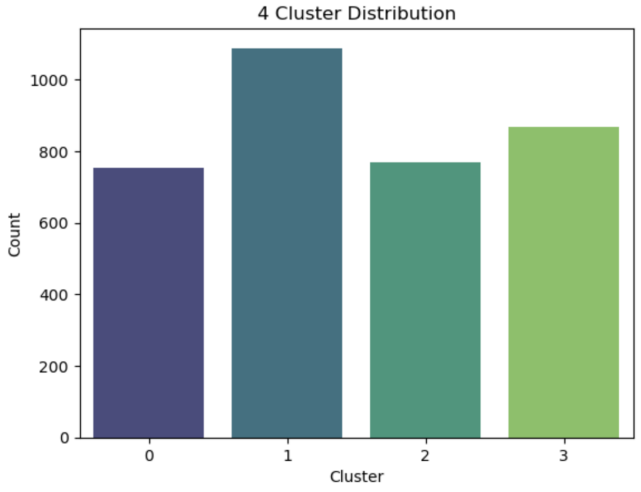
We will start using just our columns 'Recency_Score', 'Frequency_Score', 'Monetary_Score'

```
In [98]: X = rfm[['Recency_Score', 'Frequency_Score', 'Monetary_Score']]
```

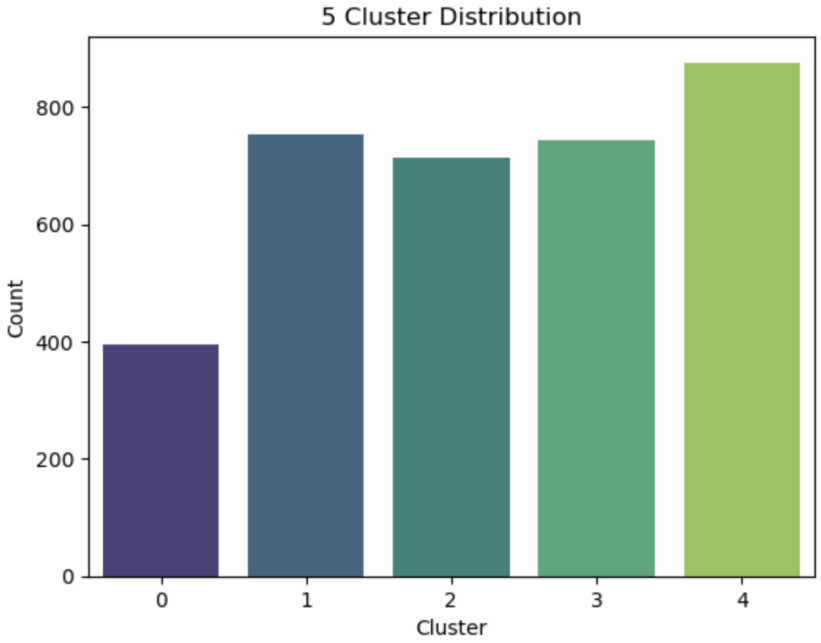
```
In [99]: from sklearn.preprocessing import StandardScaler

         scaler = StandardScaler()
         X_scaled = scaler.fit_transform(X)
```



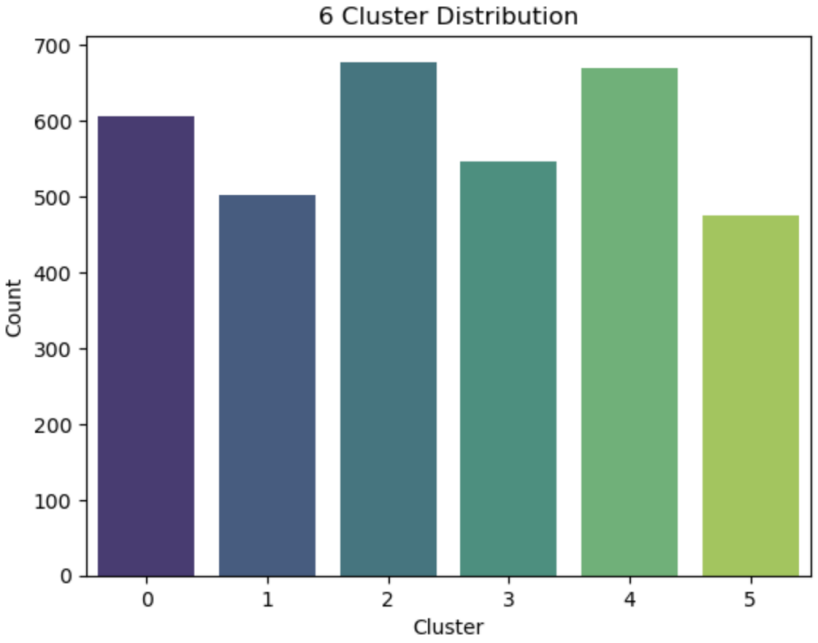
We will use KMeans method for our clustering. We will start with 4 and we will perform it with 5 and 6 too.
Starting with 4 clusters:



5 Clusters:

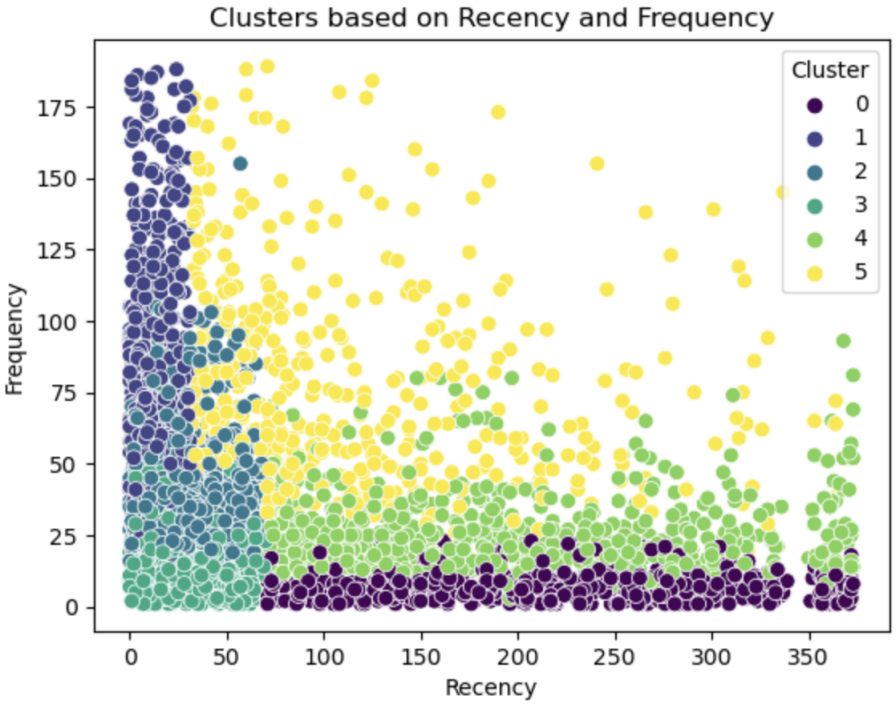


6 Clusters:

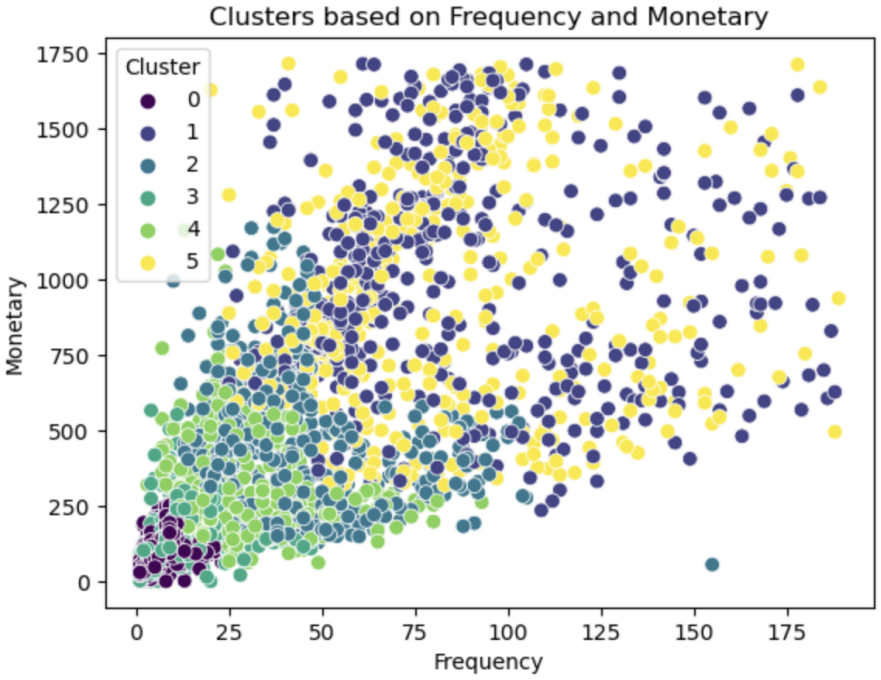


After checking the distribution of values we consider that the best cluster would be 6.

We want to visualize our clusters based on Recency and Frequency.

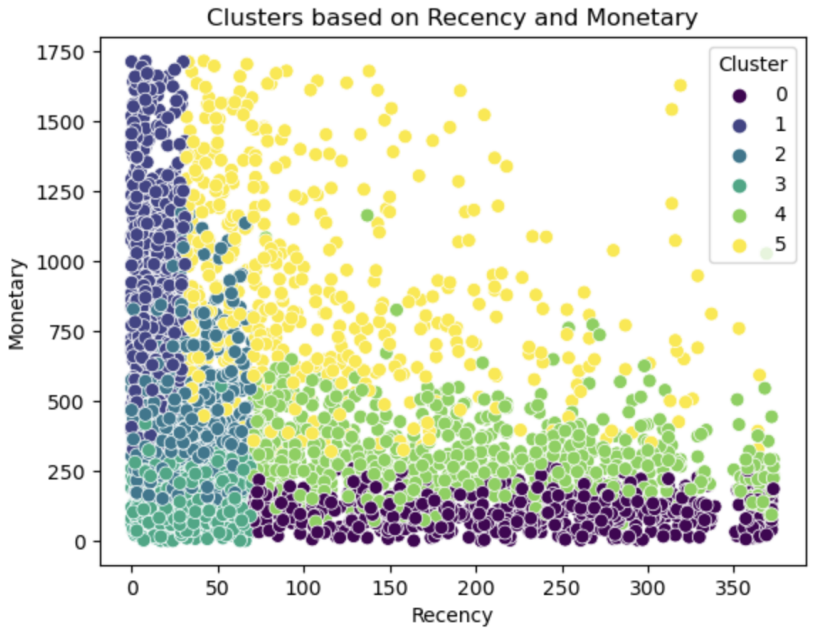


Even though some values are overlapped, is clear that we have 6 groups of data. Now we will check our clustering but for the Frequency and Monetary.



This cluster is not as clear as the others, but we can establish some layer from point 0.

Now we will check our clustering but for the Recency and Monetary.



This graph shows us that we have gotten our 6 clusters and appear to be well distributed.

-> SEGMENT PROFILING:

We will create our segmentation based on different studies.

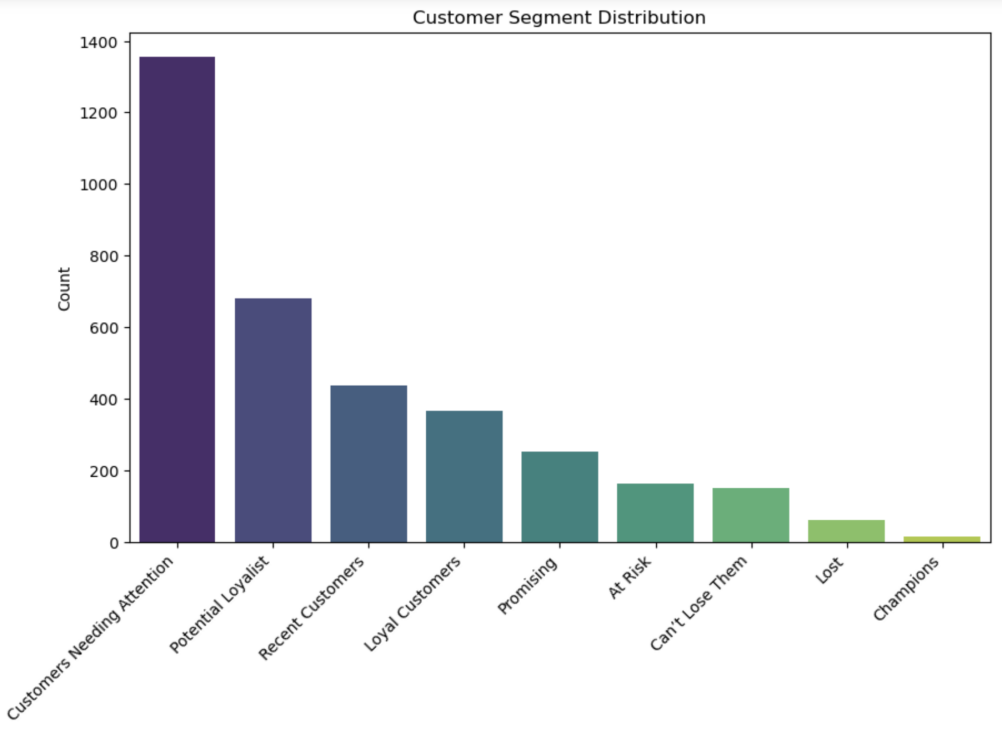
- https://www.everydaymarketing.co/business-and-marketing-case-study/data/rfm-analysis-fo-customer-segmentation-from-behavior-transaction-data/
- https://www.geeksforgeeks.org/rfm-analysis-analysis-using-python/
- https://thecleverprogrammer.com/2023/06/12/rfm-analysis-using-python/
- https://clevertap.com/blog/rfm-analysis/

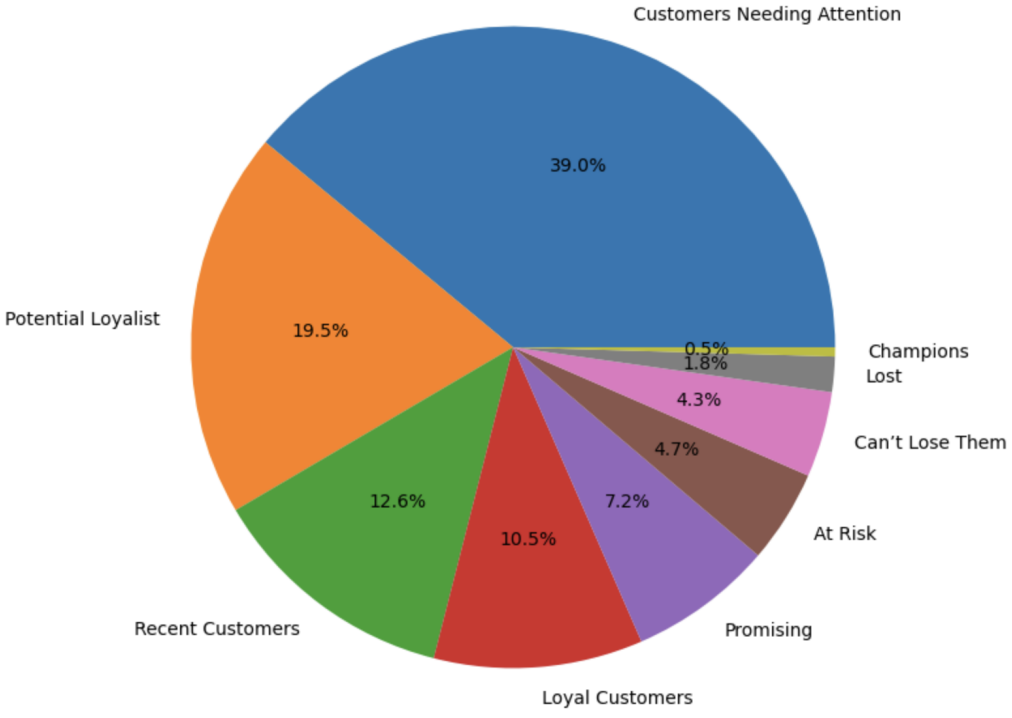
In [117]:
```python
def assign_segment(row):
    if row['Recency_Score'] == 5 and row['Frequency_Score'] == 5 and row['Monetary_Score'] == 5:
        return 'Champions'
    elif row['Recency_Score'] >= 4 and row['Frequency_Score'] >= 4 and row['Monetary_Score'] >= 4:
        return 'Loyal Customers'
    elif row['Recency_Score'] >= 3 and row['Frequency_Score'] >= 3 and row['Monetary_Score'] >= 3:
        return 'Potential Loyalist'
    elif row['Recency_Score'] >= 4:
        return 'Recent Customers'
    elif row['Recency_Score'] >= 3 and row['Monetary_Score'] >= 2:
        return 'Promising'
    elif row['Recency_Score'] <= 3 and row['Frequency_Score'] <= 3 and row['Monetary_Score'] <= 3:
        return 'Customers Needing Attention'
    elif row['Recency_Score'] <= 2 and row['Frequency_Score'] <= 2 and row['Monetary_Score'] <= 2:
        return 'About To Sleep'
    elif row['Recency_Score'] <= 2 and row['Frequency_Score'] >= 4 and row['Monetary_Score'] >= 4:
        return 'Can't Lose Them'
    elif row['Recency_Score'] <= 2 and row['Frequency_Score'] >= 3 and row['Monetary_Score'] >= 3:
        return 'At Risk'
    elif row['Recency_Score'] <= 1 and row['Frequency_Score'] <= 1 and row['Monetary_Score'] <= 1:
        return 'Hibernating'
    else:
        return 'Lost'
```

In [118]:
```
rfm['Segment'] = rfm.apply(assign_segment, axis=1)
rfm.head()
```

Out[118]:

	CustomerID	Recency	Frequency	Monetary	Recency_Score	Frequency_Score	Monetary_Score	RFM_Score	Cluster	Segment
0	12348.0	248	1	17.00	1	1	1	111	0	Customers Needing Attention
1	12349.0	18	61	1155.75	4	4	4	444	1	Loyal Customers
2	12350.0	309	15	274.00	1	2	2	122	4	Customers Needing Attention
3	12352.0	35	78	1112.11	3	4	4	344	5	Potential Loyalist
4	12353.0	203	2	29.30	1	1	1	111	0	Customers Needing Attention





```
In [240]: segment_counts

Out[240]: Customers Needing Attention    1356
          Potential Loyalist              679
          Recent Customers                437
          Loyal Customers                 365
          Promising                       252
          At Risk                         163
          Can't Lose Them                 150
          Lost                             62
          Champions                        16
          Name: Segment, dtype: int64
```

-> MARKETING RECOMMENDATIONS:

Based on the RFM analysis and the identified customer segments, we recommend the next options for each segment:

Customers Needing Attention (1356):

Make limited-time offers or promotions to re-engage them. Recommend products based on their past purchases. Send personalized emails to encourage repeat purchases.

Potential Loyalist (679):

Offer a membership or loyalty program to incentivize loyalty. Recommend complementary products based on their previous purchases. Send targeted promotions to encourage repeat business.

Recent Customers (437):

Provide onboarding support to help them get started. Offer incentives for their next purchase. Request feedback or reviews to build a relationship.

Loyal Customers (365):

Upsell higher-value products or premium services. Offer exclusive discounts or early access to new products. Engage with them through loyalty programs and exclusive events.

Promising (252):

Create brand awareness through targeted marketing. Offer free trials or samples to encourage a higher commitment. Showcase popular products and encourage exploration.

At Risk (163):

Send personalized emails with special offers to win them back. Provide incentives for reactivating their engagement. Offer renewals or exclusive discounts to retain their business.

Can't Lose Them (150):

Reach out with personalized communication to understand concerns. Offer renewals, special discounts, or exclusive products. Provide excellent customer support to retain their loyalty.

Lost (62):

Launch a reach-out campaign to revive their interest. Consider special promotions or discounts for a comeback. If unresponsive, focus efforts on more engaged segments.

Champions (16):

Reward them for their loyalty with exclusive perks. Consider them as potential early adopters of new products. Encourage them to become brand ambassadors through referral programs.

-> *Time period covered by this dataset:*

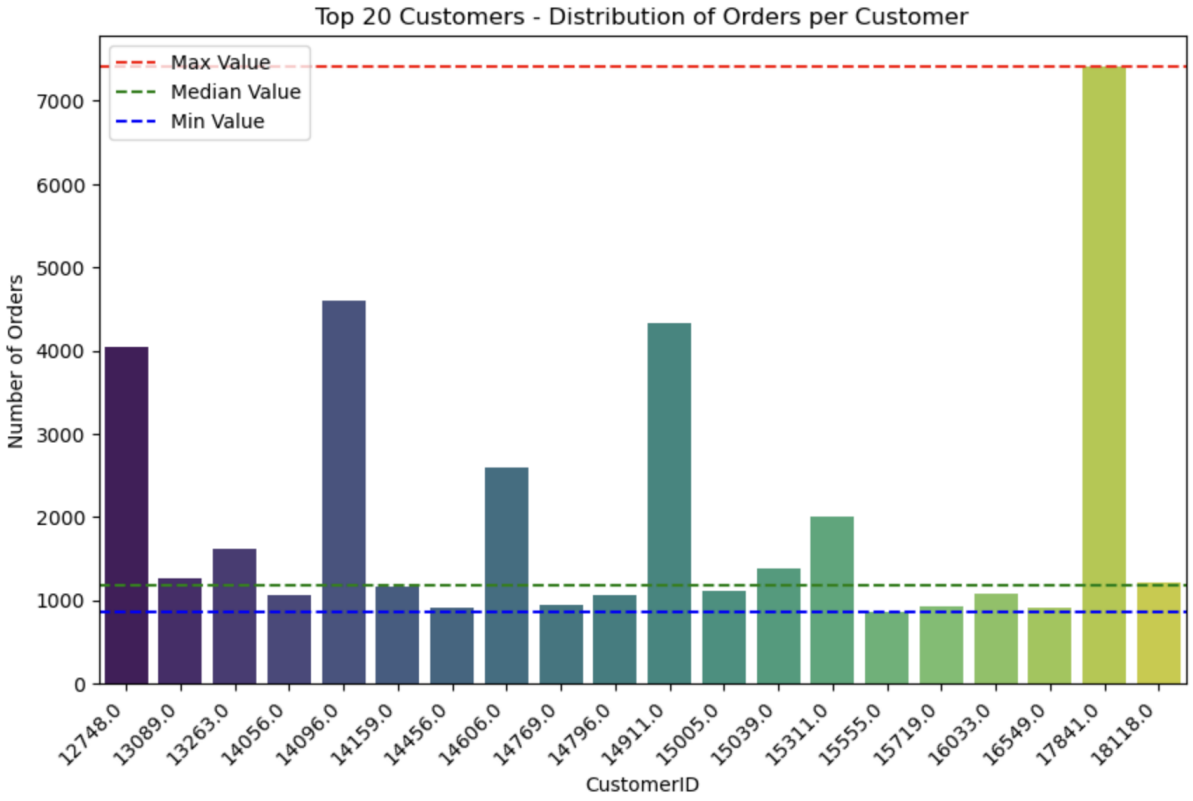
In [166]:
```
start_date = df['InvoiceDate'].min()
end_date = df['InvoiceDate'].max()

# Display the time period covered by the dataset
print(f'Time period covered by the dataset: {start_date} to {end_date}')
```

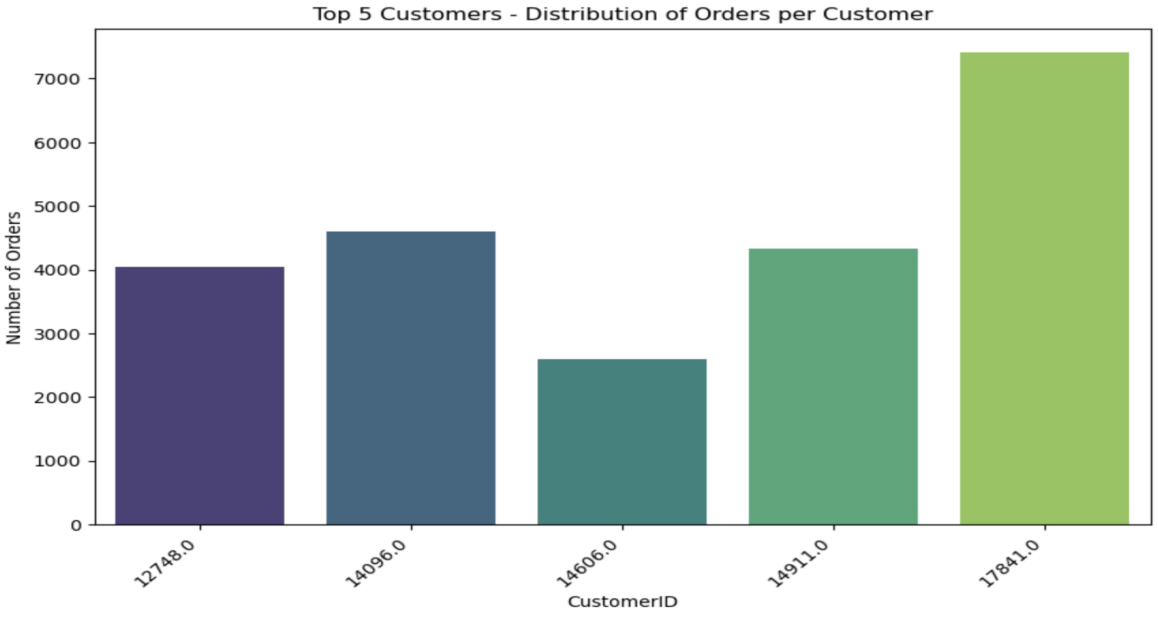
```
Time period covered by the dataset: 2010-12-01 08:26:00 to 2011-12-09 12:50:00
```

-> *Unique customer analysis present in the dataset:*

```
In [121]: unique = df['CustomerID'].nunique()
          print(f'Number of unique customers: {unique}')

          Number of unique customers: 4208
```

Since we have all those customers and we want to see the distribution in a proper way we will select of the 20 most frequent customers.





->*Analysing frequently purchased products:*

We have 3878 unique products

In [130]:
```
unique_products = df['Description'].nunique()
print(f'Number of unique Products: {unique_products}')
```

```
Number of unique Products: 3878
```

In [168]:
```
orders_per_product = df.groupby('Description')['Quantity'].sum().sort_values(ascending = False).head(10)
top_10_products = orders_per_product.nlargest(10)

print('Top 10 Most Frequently Purchased Products:')
top_10_products
```

```
Top 10 Most Frequently Purchased Products:
```

Out[168]:
```
Description
JUMBO BAG RED RETROSPOT               13930
ASSORTED COLOUR BIRD ORNAMENT         10915
WHITE HANGING HEART T-LIGHT HOLDER    10806
LUNCH BAG RED RETROSPOT               10374
LUNCH BAG  BLACK SKULL.                8103
LUNCH BAG CARS BLUE                    7838
JAM MAKING SET PRINTED                 7582
JUMBO BAG PINK POLKADOT                7400
LUNCH BAG APPLE DESIGN                 7358
LUNCH BAG SPACEBOY DESIGN              7206
Name: Quantity, dtype: int64
```

The average price of products in the dataset is:

In [170]:
```
average_price_products = df['UnitPrice'].mean()
print(f'Average Price of Products: ${average_price_products:.2f}')
```

```
Average Price of Products: $2.79
```

We do not have the production price of each product so the top 5 products with the highest profit margins are the products that made the most money.

In [212]:
```
revenues=df.groupby('Description')['Total'].sum().sort_values(ascending=False)
revenues.head(5)
```

Out[212]:
```
Description
PARTY BUNTING                         33965.58
WHITE HANGING HEART T-LIGHT HOLDER    33764.42
JUMBO BAG RED RETROSPOT               32054.02
JAM MAKING SET WITH JARS              22998.51
SPOTTY BUNTING                        22515.55
Name: Total, dtype: float64
```

In [213]:
```
product_name=revenues.index[0]
highest_revenue =revenues.iloc[0]
print(f"{product_name} generates the Highest Revenue: ${highest_revenue}")
```

```
PARTY BUNTING generates the Highest Revenue: $33965.58
```

In [214]:
```
revenues.describe()
```

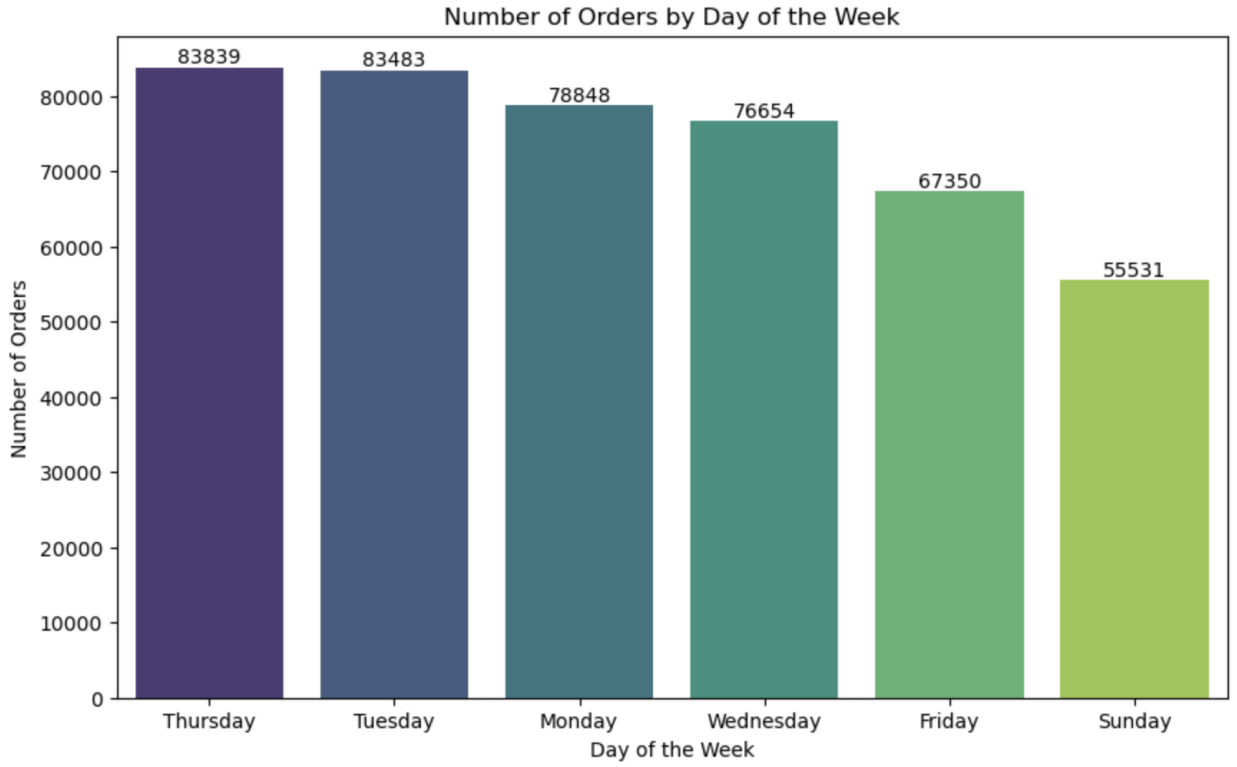
Out[214]:
```
count     3878.000000
mean      1230.064601
std       2521.686394
min       -107.940000
25%         65.340000
50%        352.970000
75%       1181.455000
max      33965.580000
Name: Total, dtype: float64
```

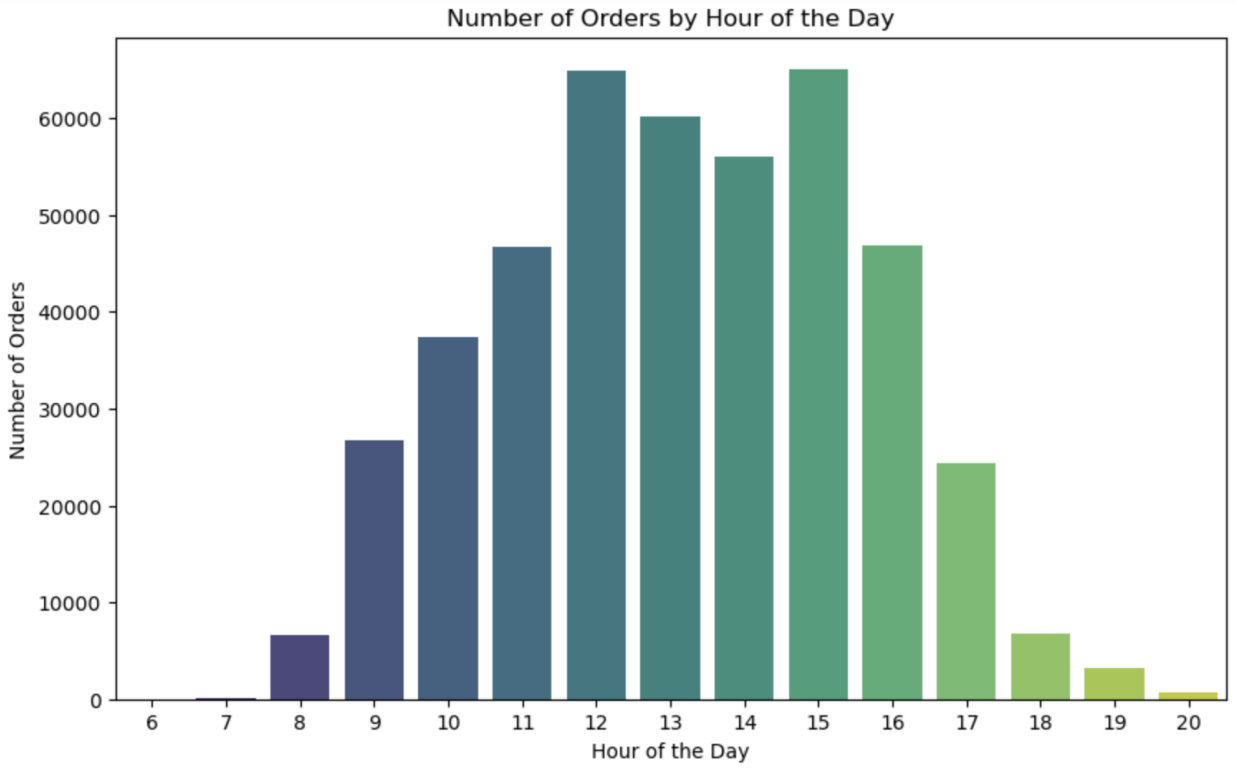
The total profit generated by the company during the dataset's time period is:

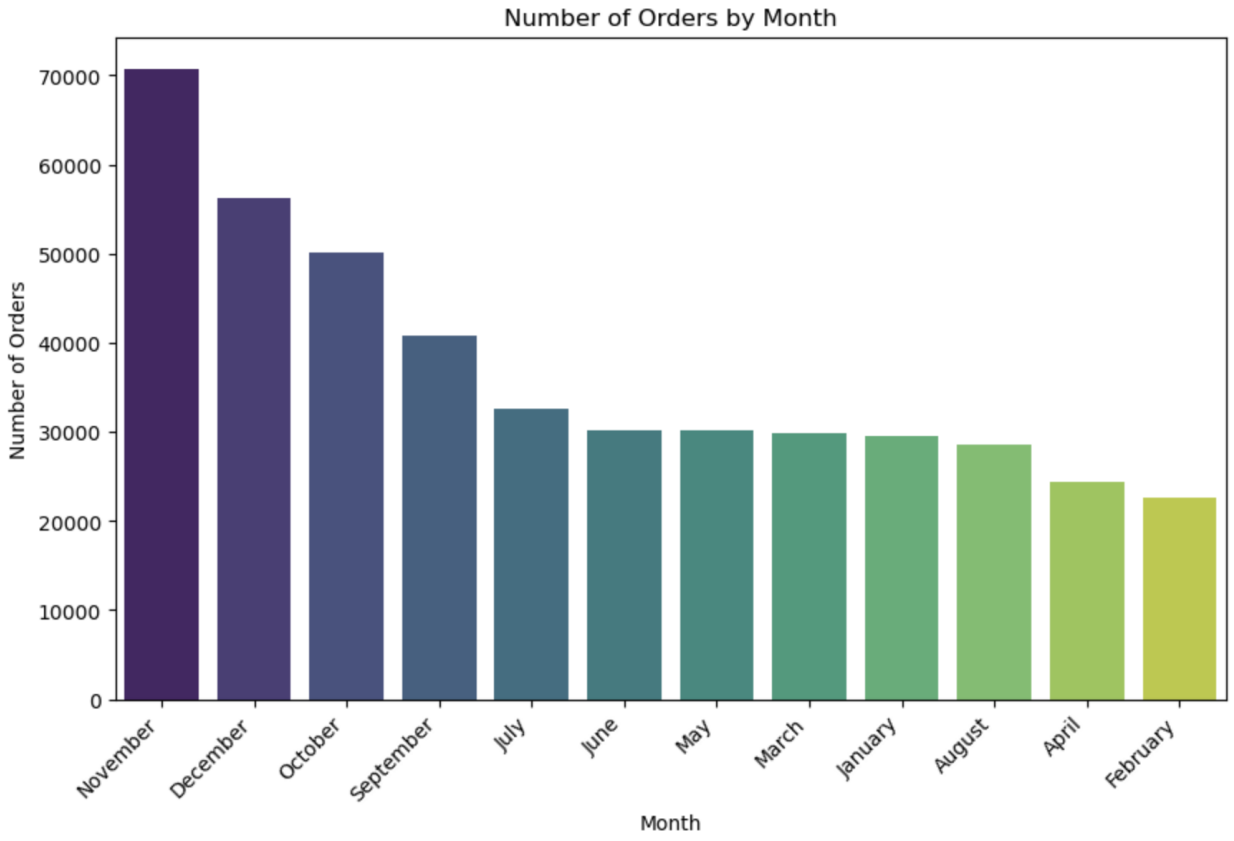
In [215]:
```
revenues.describe()
total_revenues=revenues.values.sum()
print(f'The total amount of revenue made was: ${total_revenues:.2f}')
```

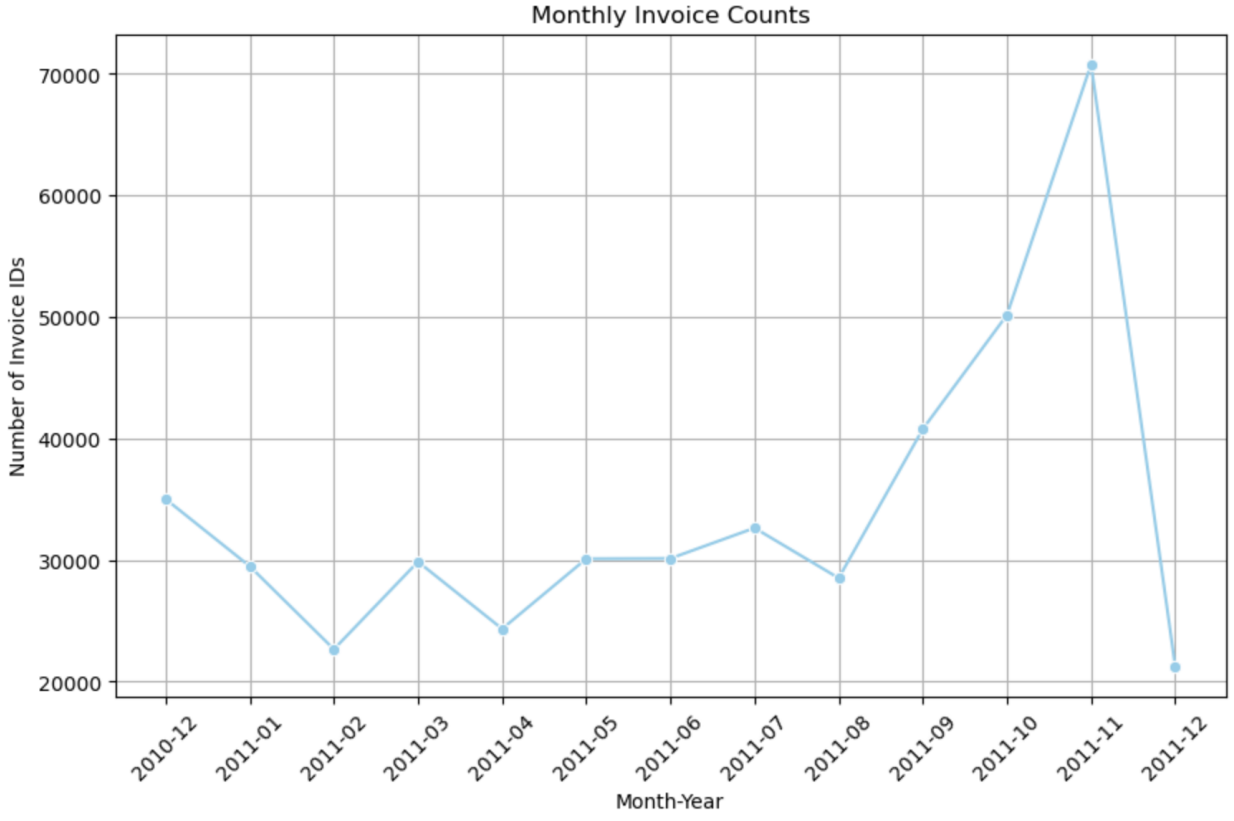
```
The total amount of revenue made was: $4770190.52
```

->*Analysis of time and trends in the dataset:*

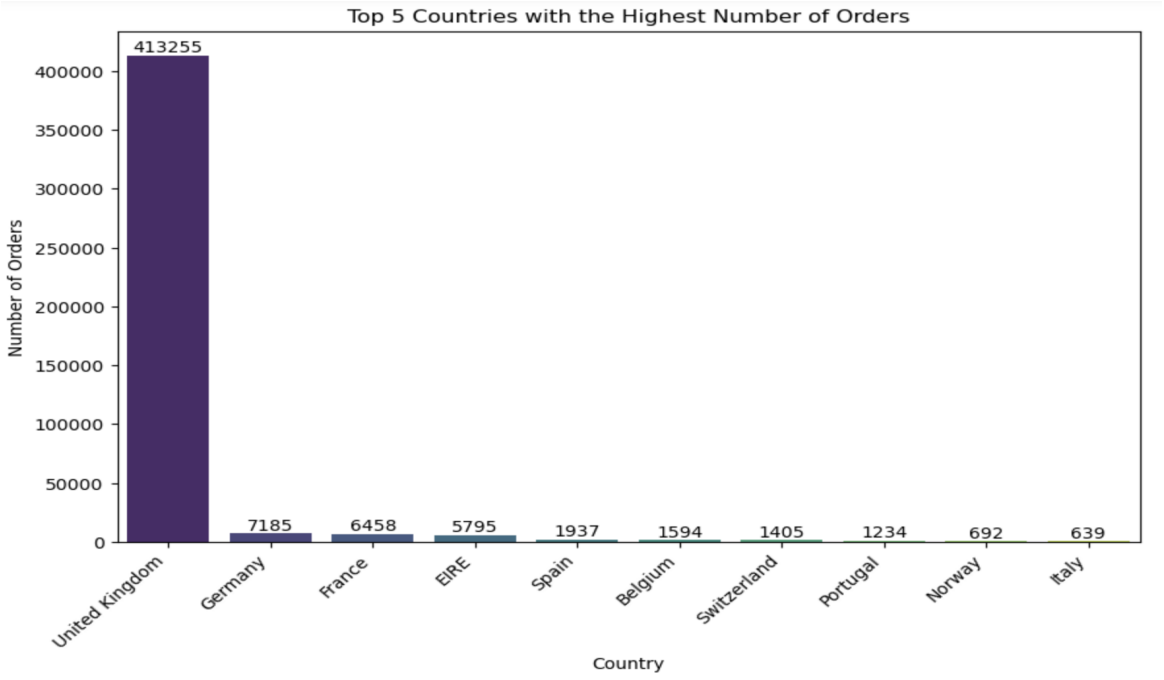


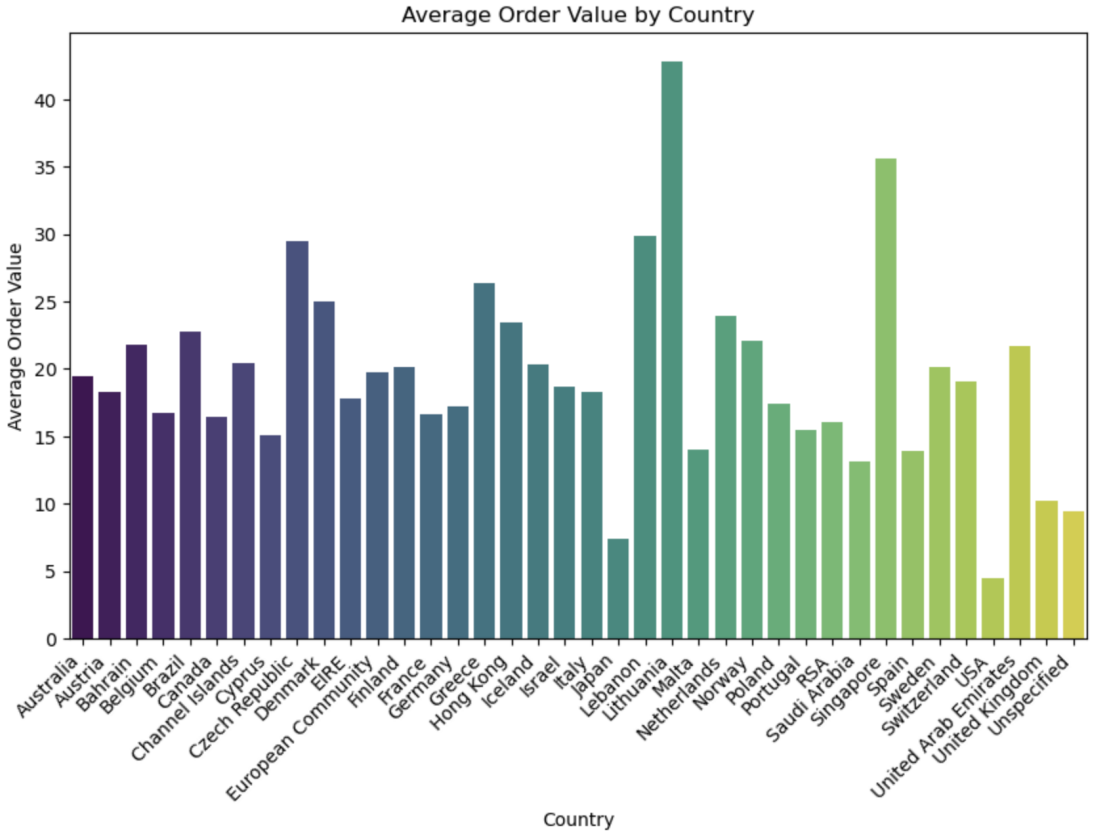


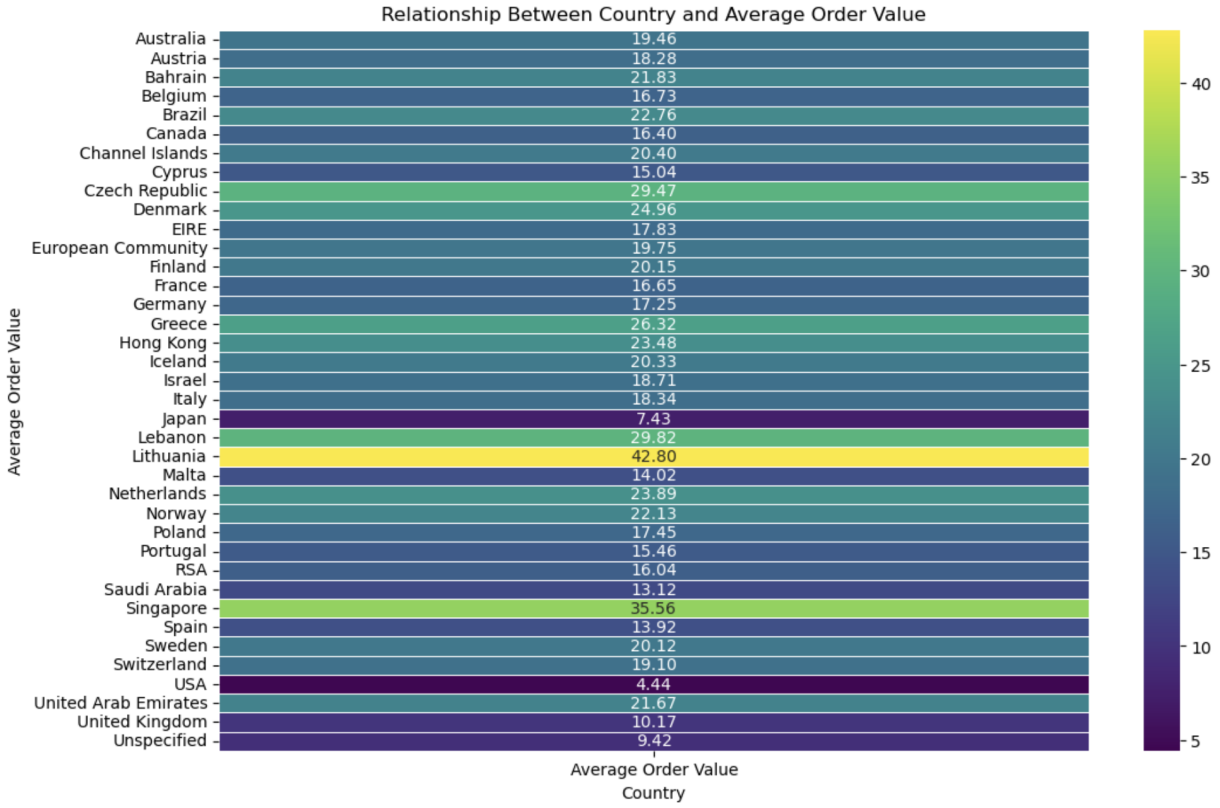

Number of Orders by Month
Number of Orders
70000
60000
50000
40000
30000
20000
10000
0
November
December
October
September
July
June
May
March
January
August
April
February
Month

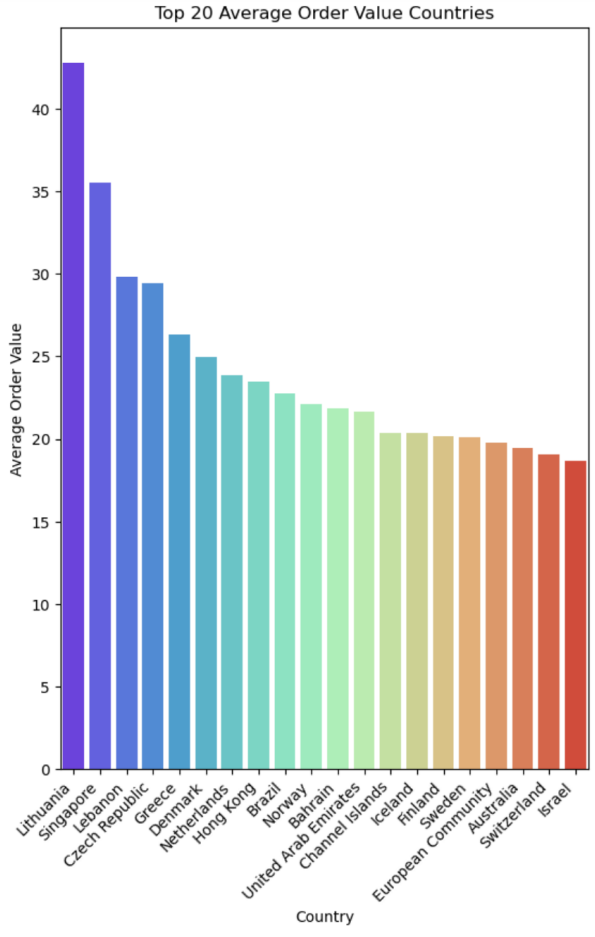

Monthly Invoice Counts
Number of Invoice IDs
70000
60000
50000
40000
30000
20000
2010-12
2011-01
2011-02
2011-03
2011-04
2011-05
2011-06
2011-07
2011-08
2011-09
2011-10
2011-11
2011-12
Month-Year

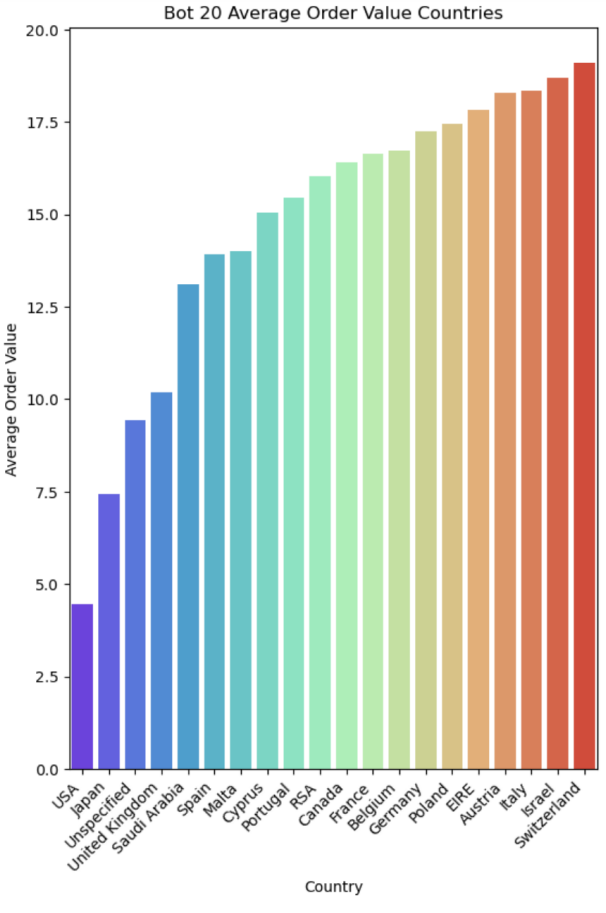
->*Analysis of geography:*

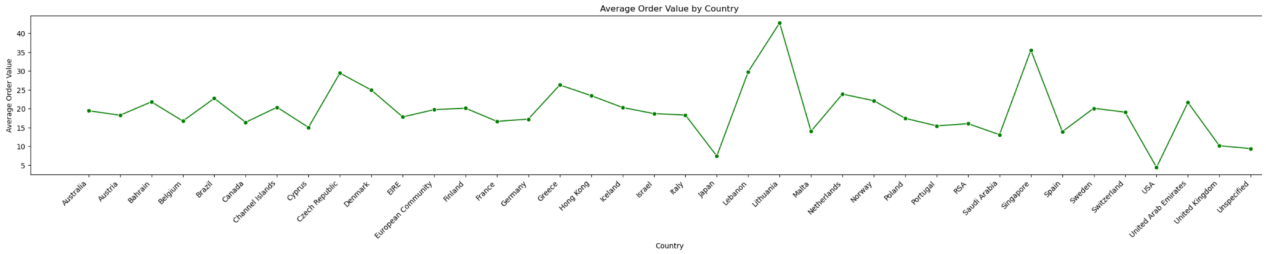





Relationship Between Country and Average Order Value
Australia 19.46
Austria 18.28
Bahrain 21.83
Belgium 16.73
Brazil 22.76
Canada 16.40
Channel Islands 20.40
Cyprus 15.04
Czech Republic 29.47
Denmark 24.96
EIRE 17.83
European Community 19.75
Finland 20.15
France 16.65
Germany 17.25
Greece 26.32
Hong Kong 23.48
Iceland 20.33
Israel 18.71
Italy 18.34
Japan 7.43
Lebanon 29.82
Lithuania 42.80
Malta 14.02
Netherlands 23.89
Norway 22.13
Poland 17.45
Portugal 15.46
RSA 16.04
Saudi Arabia 13.12
Singapore 35.56
Spain 13.92
Sweden 20.12
Switzerland 19.10
USA 4.44
United Arab Emirates 21.67
United Kingdom 10.17
Unspecified 9.42
Average Order Value
Country
Average Order Value
40
35
30
25
20
15
10
5


Top 20 Average Order Value Countries
Average Order Value
40
35
30
25
20
15
10
5
0
Lithuania
Singapore
Lebanon
Czech Republic
Greece
Denmark
Netherlands
Hong Kong
Brazil
Norway
Bahrain
United Arab Emirates
Channel Islands
Iceland
Finland
Sweden
European Community
Australia
Switzerland
Israel
Country


Bot 20 Average Order Value Countries
Average Order Value
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
USA
Japan
Unspecified
United Kingdom
Saudi Arabia
Spain
Malta
Cyprus
Portugal
RSA
Canada
France
Belgium
Germany
Poland
EIRE
Austria
Italy
Israel
Switzerland
Country



->*Averagre duration of customer remaining active:*

We will check on average how long customers remain active (between their first and last purchase)

In [155]:
```
customer_activity= df.groupby('CustomerID')['InvoiceDate'].agg(['min', 'max'])
customer_activity['Customer_activity'] = (customer_activity['max'] - customer_activity['min']).dt.days

# Calculate the average customer lifespan
average_customer_activity = customer_activity['Customer_activity'] .mean()

print(f'On average customers remain active (between their first and last purchase): {average_customer_activity:.2f}
```

```
On average customers remain active (between their first and last purchase): 130.46 days
```

The customer segments based on their purchase behavior was already discussed in the RFM section

-> *Analysing returns and refunds:*

We want to analyze how many purchases were returned so we will consider the ones that have 'Quantity'< 0 and the word refund or return in the Description.

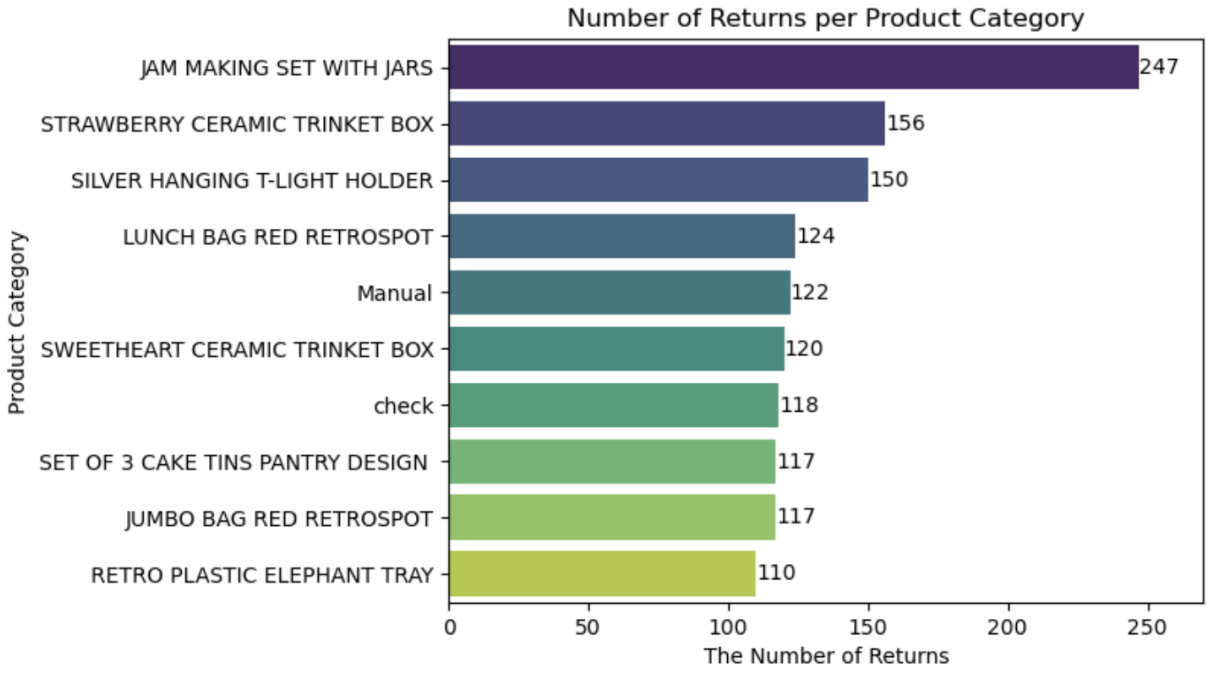
In [228]:
```
negative_values_count = (df['Quantity'] < 0).sum()
negative_values_count
```

Out[228]: 7149

Based on that we can calculate the Percentage of orders with returns or refunds

In [187]:
```
percent_re = (negative_values_count + len(filtered_rows)) / len(df)
print(f"Percentage of orders with returns or refunds: {percent_re:.3f}%")
```

```
Percentage of orders with returns or refunds: 0.016%
```



In [204]:
```
product_name=return_product_count.index[0]
highest_product =return_product_count.iloc[0]
print(f"{product_name} is the product with the highest number of returns with {highest_product:.2f}")
```

```
JAM MAKING SET WITH JARS is the product with the highest number of returns with 247.00
```

->*Analysing methods of monetary transactions:*

We do not have the information about the payment method but we know that 'Credit Cards', 'Paypal', 'Gift Card', 'Mobile Payment', 'E-check' are the most famous so we will use those to randomly assign values for each purchase.

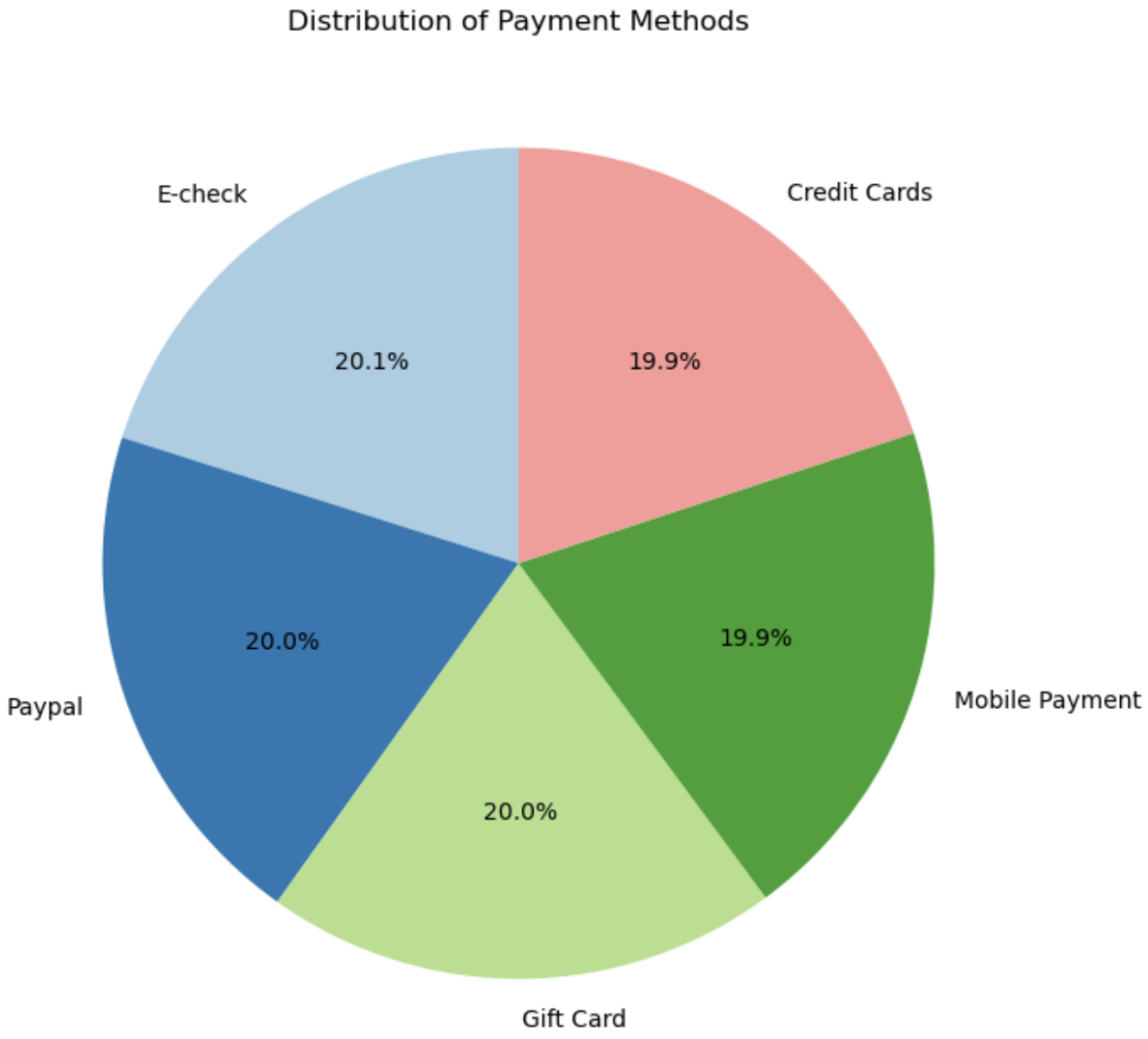
In [226]:
```
payment_method = ['Credit Cards', 'Paypal', 'Gift Card', 'Mobile Payment', 'E-check']

np.random.seed(42)

df['PaymentMethod'] = np.random.choice(payment_method, size=len(df))
df.head()
```

Out[226]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total	DayOfWeek	Hour	Month_Name	PaymentMethod
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	15.30	Wednesday	8	December	Mobile Payment
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	Wednesday	8	December	E-check
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	22.00	Wednesday	8	December	Gift Card
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	Wednesday	8	December	E-check
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	Wednesday	8	December	E-check

In [227]:
```
most_common = df['PaymentMethod'].value_counts().index[0]

print("Most common payment methods:", most_common)
```

```
Most common payment methods: E-check
```


Distribution of Payment Methods
E-check
Credit Cards
20.1%
19.9%
20.0%
19.9%
Paypal
Mobile Payment
20.0%
Gift Card