Project Name: SmartHive AI

Project Type - Unsupervised ML

Github

Link: https://github.com/SKitavi/Smarthive-ai

Problem Statement

In Kenya's competitive retail market, small businesses often face challenges in effectively targeting and engaging a diverse customer base. Ineffective marketing strategies, stemming from a lack of detailed customer insights, lead to wasted marketing budgets and missed revenue opportunities. For small businesses, these inefficiencies can result in marketing costs increasing by up to 30% and customer retention rates dropping by 20%. To address these challenges, SmartHive AI offers a data-driven customer segmentation model tailored for small businesses, aiming to improve marketing efficiency, increase revenue by up to 15%, and enhance customer retention by 25%.

Stakeholders: Marketing Team, Sales Team, Product Development Team.

Objectives

- Build a data-driven model to segment customers based on their purchasing behaviour, demographics, preferences, and engagement patterns.
- Measure and Evaluate the Impact of Segmentation Strategies
- Deploy these capabilities through an interactive dashboard,or web application enabling marketing teams to visualise segments, execute targeted campaigns, and monitor their performance in real-time.

1. Data Collection and Preparation

For this project, we will mainly be using the Online Retail.xlsx dataset. The dataset includes features such as customer demographics, purchase history, frequency of purchases, monetary value of purchases, and other relevant variables that can help in segmenting customers effectively.

Data Description

Attribute Information:

InvoiceNo: Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter 'c', it indicates a cancellation.

StockCode: Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.

Description: Product (item) name. Nominal.

Quantity: The quantities of each product (item) per transaction. Numeric.

InvoiceDate: Invoice Date and time. Numeric, the day and time when each transaction was generated.

UnitPrice: Unit price. Numeric, Product price per unit in sterling.

CustomerID: Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.

Country: Country name. Nominal, the name of the country where each customer resides.

Importing Libraries

```
#imorting important libraries.

import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings('ignore')
from numpy import math
```

Data Loading and Preview

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#reading the excel file and preview using head()
retail_df=pd.read_excel('/content/drive/MyDrive/Market Segmentation/Online Retail.xlsx')
retail_df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom

```
#.tail() reads bottom 5 records
retail_df.tail()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France

```
# checking the datatypes and null values in dataset
retail_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

Observations

- Datatype of InvoiceDate is object need to convert it into datatime.
- If InvoiceNo starts with C means it's a cancellation. We need to drop these entries.

```
# shape of dataset
retail_df.shape
```

```
(541909, 8)
```

- There are 541,909 rows/records and 8 columns in this dataset.

```
retail_df.describe()
```

	Quantity	InvoiceDate	UnitPrice	CustomerID
count	541909.000000	541909	541909.000000	406829.000000
mean	9.552250	2011-07-04 13:34:57.156386048	4.611114	15287.690570
min	-80995.000000	2010-12-01 08:26:00	-11062.060000	12346.000000
25%	1.000000	2011-03-28 11:34:00	1.250000	13953.000000
50%	3.000000	2011-07-19 17:17:00	2.080000	15152.000000
75%	10.000000	2011-10-19 11:27:00	4.130000	16791.000000
max	80995.000000	2011-12-09 12:50:00	38970.000000	18287.000000
std	218.081158	NaN	96.759853	1713.600303

```
# Let's check the null values count.
retail_df.isnull().sum().sort_values(ascending=False)
```

	0
CustomerID	135080
Description	1454
InvoiceNo	0
StockCode	0
Quantity	0
InvoiceDate	0
UnitPrice	0
Country	0

dtype: int64

- There are null values in CustomerID and Description.

```
# Count the number of duplicates
duplicate_count = retail_df.duplicated().sum()

print(f"Number of duplicate rows: {duplicate_count}")
```

```
Number of duplicate rows: 5268
```

```
# Count the number of unique rows
unique_rows_count = retail_df.drop_duplicates(keep='first').shape[0]
print("\nNumber of unique rows:", unique_rows_count)

# Check for unique values in specific columns
# For example, to check the uniqueness of 'CustomerID' column:
unique_customer_count = retail_df['CustomerID'].nunique()
print("\nNumber of unique customers:", unique_customer_count)
```

```

Number of unique rows: 536641

Number of unique customers: 4372
```

```
#Summary Statistics for Numerical Features

print("\nSummary Statistics:")
retail_df.describe().T
```

Summary Statistics:

	count	mean	min	25%	50%	75%	max	std
Quantity	541909.0	9.55225	-80995.0	1.0	3.0	10.0	80995.0	218.081158
InvoiceDate	541909	2011-07-04 13:34:57.156386048	2010-12-01 08:26:00	2011-03-28 11:34:00	2011-07-19 17:17:00	2011-10-19 11:27:00	2011-12-09 12:50:00	NaN
UnitPrice	541909.0	4.611114	-11062.06	1.25	2.08	4.13	38970.0	96.759853

Data Cleaning

```
# Visulaizing null values using heatmap.
plt.figure(figsize=(15,5))
sns.heatmap(retail_df.isnull(),cmap='plasma',annot=False,yticklabels=False)
plt.title(" Visualising Missing Values")
```

```
Text(0.5, 1.0, ' Visualising Missing Values')
```

Observations

- Missing values in CustomerID and Description columns.
- CustomerID is our identification feature so if its missing means other wont help us in analysis
- Dropping that all missing datapoints

```
retail_df.dropna(inplace=True)
```

```
retail_df.shape
```

```
(406829, 8)
```

- Now we have 406,829 records after removing null datapoints.

```
retail_df.describe()
```

	Quantity	InvoiceDate	UnitPrice	CustomerID
count	406829.000000	406829	406829.000000	406829.000000
mean	12.061303	2011-07-10 16:30:57.879207424	3.460471	15287.690570
min	-80995.000000	2010-12-01 08:26:00	0.000000	12346.000000
25%	2.000000	2011-04-06 15:02:00	1.250000	13953.000000
50%	5.000000	2011-07-31 11:48:00	1.950000	15152.000000
75%	12.000000	2011-10-20 13:06:00	3.750000	16791.000000
max	80995.000000	2011-12-09 12:50:00	38970.000000	18287.000000
std	248.693370	NaN	69.315162	1713.600303

- Here we can see that min value for Quantity column is negative.
- UnitPrice has 0 as min value
- Need to Explore these columns

```
# dataframe have negative valiues in quantity.
retail_df[retail_df['Quantity']<0]
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
141	C536379	D	Discount	-1	2010-12-01 09:41:00	27.50	14527.0	United Kingdom
154	C536383	35004C	SET OF 3 COLOURED FLYING DUCKS	-1	2010-12-01 09:49:00	4.65	15311.0	United Kingdom
235	C536391	22556	PLASTERS IN TIN CIRCUS PARADE	-12	2010-12-01 10:24:00	1.65	17548.0	United Kingdom
236	C536391	21984	PACK OF 12 PINK PAISLEY TISSUES	-24	2010-12-01 10:24:00	0.29	17548.0	United Kingdom
237	C536391	21983	PACK OF 12 BLUE PAISLEY TISSUES	-24	2010-12-01 10:24:00	0.29	17548.0	United Kingdom
...	...	...	...	...	...	...	...	...
540449	C581490	23144	ZINC T-LIGHT HOLDER STARS SMALL	-11	2011-12-09 09:57:00	0.83	14397.0	United Kingdom
541541	C581499	M	Manual	-1	2011-12-09 10:28:00	224.69	15498.0	United Kingdom

- Here we observed that Invoice number starting with C has negative values and as per description of the data those are cancelations. So we need to drop these entries.

```
# changing the datatype to str
retail_df['InvoiceNo'] = retail_df['InvoiceNo'].astype('str')
```

```
# also If InvoiceNo starts with C means it's a cancellation. We need to drop this entries.
retail_df=retail_df[~retail_df['InvoiceNo'].str.contains('C')]
```

```
# Checking how many values are present for unitprice==0
# almost 40 values are present so will drop this values
len(retail_df[retail_df['UnitPrice']==0])
```

```
40
```

```
# taking unitprice values greater than 0.
retail_df=retail_df[retail_df['UnitPrice']>0]
retail_df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom

- Now our values are okay and we have eliminated the negative and zero minimums

```
retail_df.shape
```

```
(397884, 8)
```

- We have 397,884 datapoints left after cleaning.

Feature Engineering

```
# Converting InvoiceDate to datetime. InvoiceDate is in format of 01-12-2010 08:26.
retail_df["InvoiceDate"] = pd.to_datetime(retail_df["InvoiceDate"], format="%d-%m-%Y %H:%M")
```

```
retail_df["year"] = retail_df["InvoiceDate"].apply(lambda x: x.year)
retail_df["month_num"] = retail_df["InvoiceDate"].apply(lambda x: x.month)
retail_df["day_num"] = retail_df["InvoiceDate"].apply(lambda x: x.day)
retail_df["hour"] = retail_df["InvoiceDate"].apply(lambda x: x.hour)
retail_df["minute"] = retail_df["InvoiceDate"].apply(lambda x: x.minute)
```

```
# extracting month from the Invoice date
retail_df['Month']=retail_df['InvoiceDate'].dt.month_name()
```

```
# extracting day from the Invoice date
retail_df['Day']=retail_df['InvoiceDate'].dt.day_name()
```

```
retail_df['TotalAmount']=retail_df['Quantity']*retail_df['UnitPrice']
```

```
retail_df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	year	month_num	day_num	hour	minute
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010	12	1	8	26
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010	12	1	8	26
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010	12	1	8	26
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010	12	1	8	26
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010	12	1	8	26

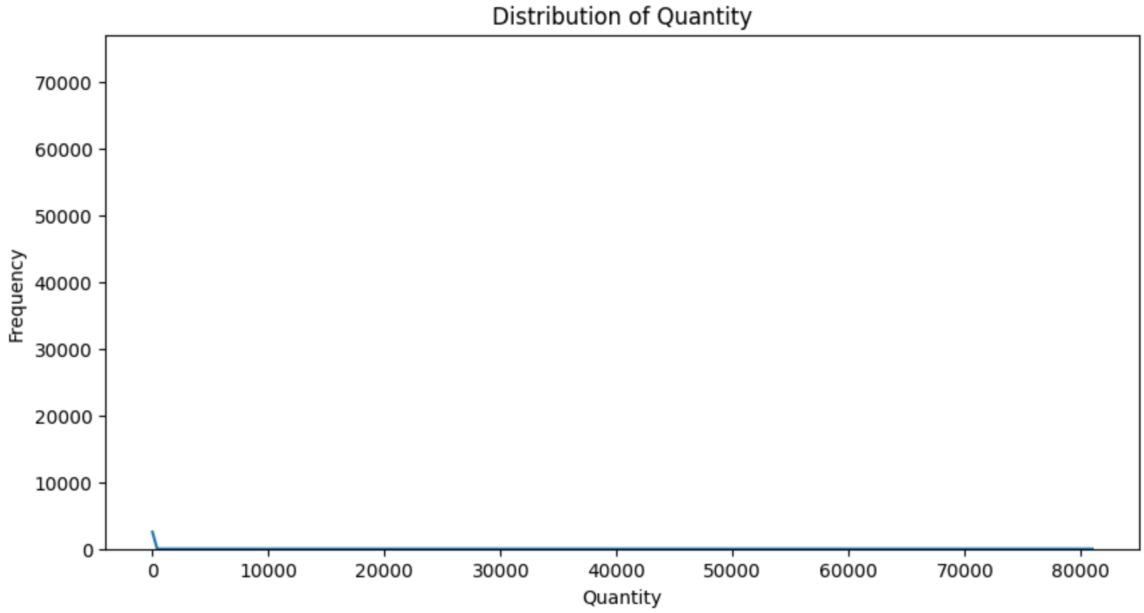
EDA(Exploratory Data Analysis)

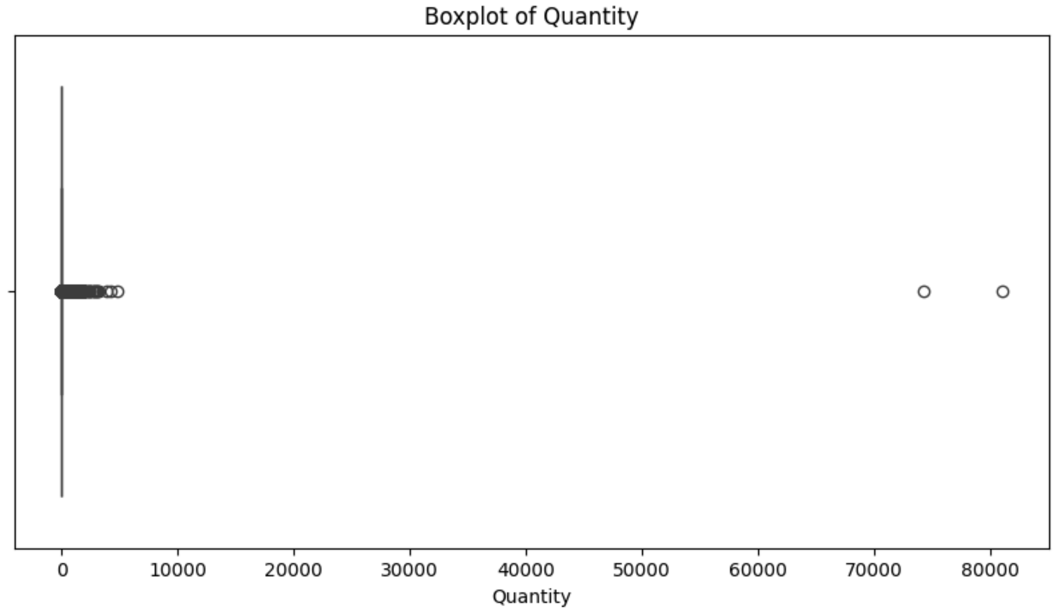
```
retail_df.columns
```

```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country', 'year', 'month_num', 'day_num',
       'hour', 'minute', 'Month', 'Day', 'TotalAmount'],
      dtype='object')
```

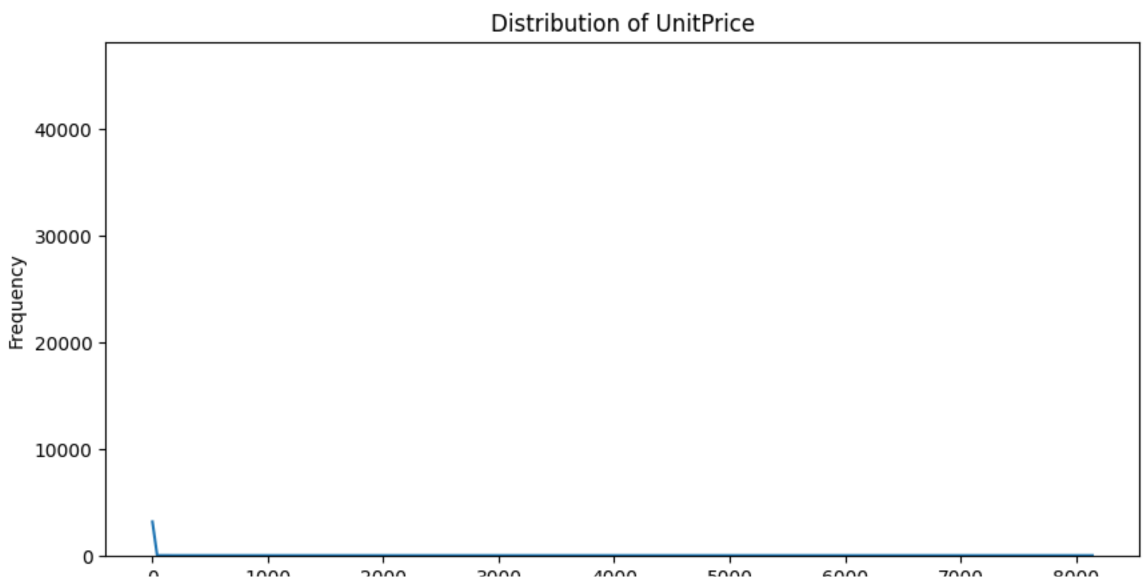
1. Univariate Analysis

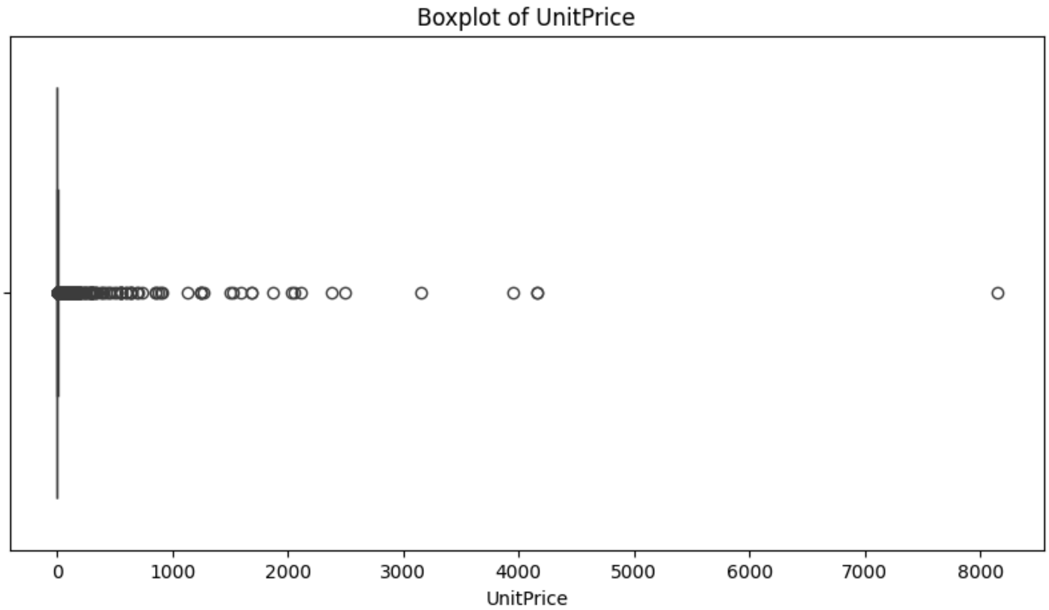
```
# Univariate Analysis for Numerical Features
numerical_features = ['Quantity', 'UnitPrice', 'TotalAmount']

for feature in numerical_features:
  plt.figure(figsize=(10, 5))
  sns.histplot(retail_df[feature], kde=True)
  plt.title(f'Distribution of {feature}')
  plt.xlabel(feature)
  plt.ylabel('Frequency')
  plt.show()

  plt.figure(figsize=(10, 5))
  sns.boxplot(x=retail_df[feature])
  plt.title(f'Boxplot of {feature}')
  plt.show()

  print(f"\nSummary Statistics for {feature}:")
  print(retail_df[feature].describe())

# Univariate Analysis for Categorical Features
categorical_features = ['Country', 'Month', 'Day']

for feature in categorical_features:
  plt.figure(figsize=(15, 5))
  sns.countplot(x=retail_df[feature], order=retail_df[feature].value_counts().index)
  plt.title(f'Frequency of {feature}')
  plt.xticks(rotation=90)
  plt.show()

  print(f"\nValue Counts for {feature}:")
  print(retail_df[feature].value_counts())
```



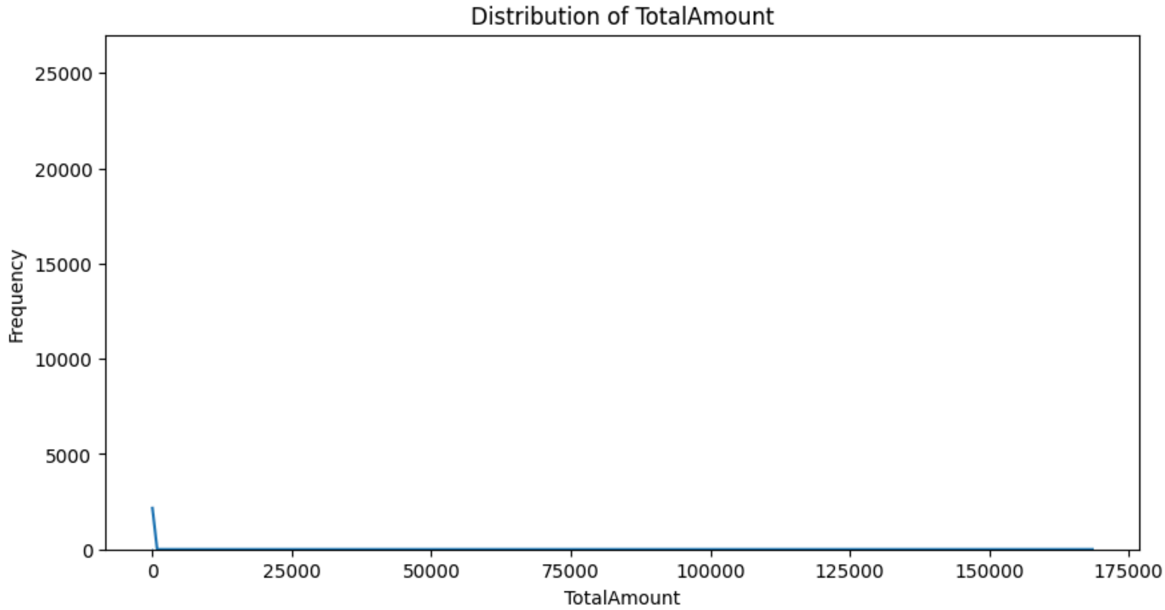


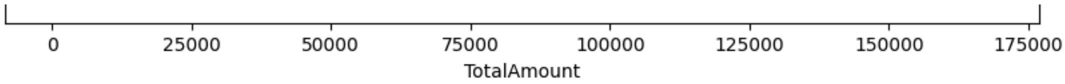

```
Summary Statistics for Quantity:
count    397884.000000
mean         12.988238
std         179.331775
min           1.000000
25%           2.000000
50%           6.000000
75%          12.000000
max       80995.000000
Name: Quantity, dtype: float64
```



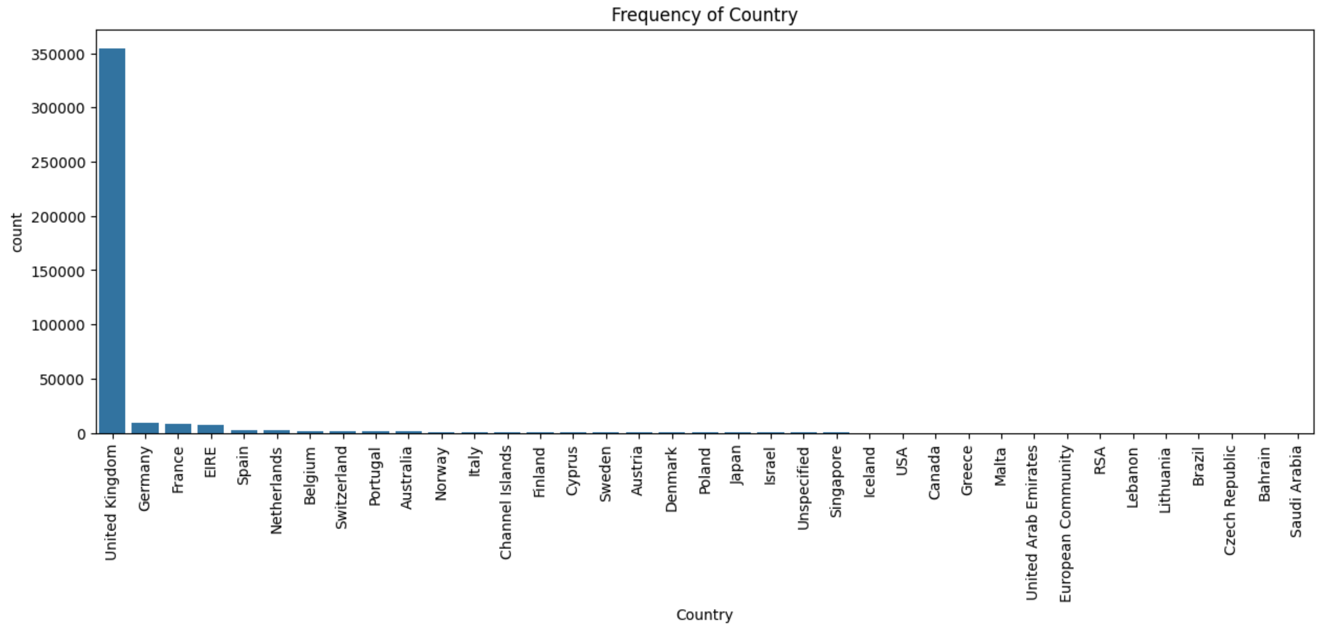



```
Summary Statistics for UnitPrice:
count    397884.000000
mean          3.116488
std          22.097877
min           0.001000
25%           1.250000
50%           1.950000
75%           3.750000
max        8142.750000
Name: UnitPrice, dtype: float64
```

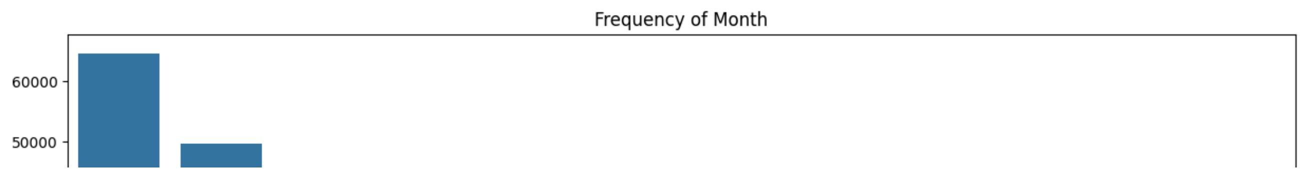


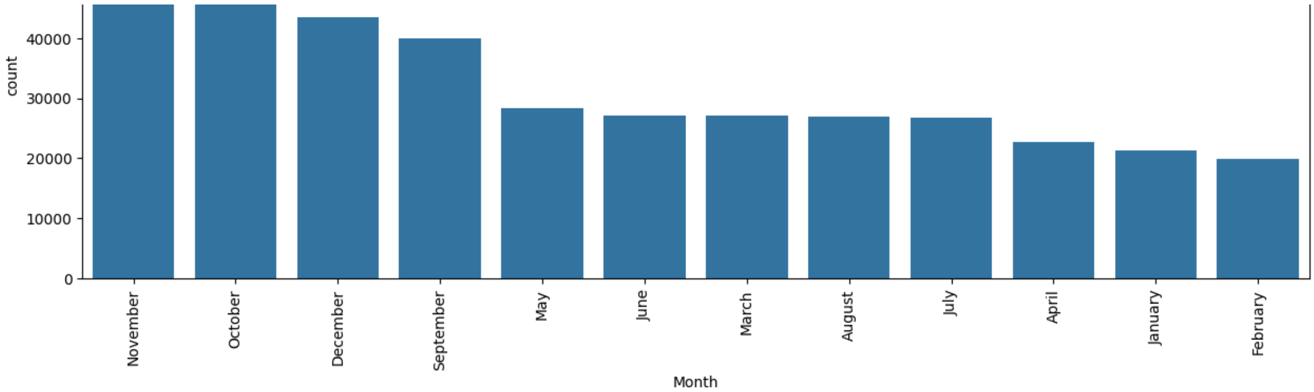



```
Summary Statistics for TotalAmount:
count    397884.000000
mean         22.397000
std         309.071041
min           0.001000
25%           4.680000
50%          11.800000
75%          19.800000
max      168469.600000
Name: TotalAmount, dtype: float64
```

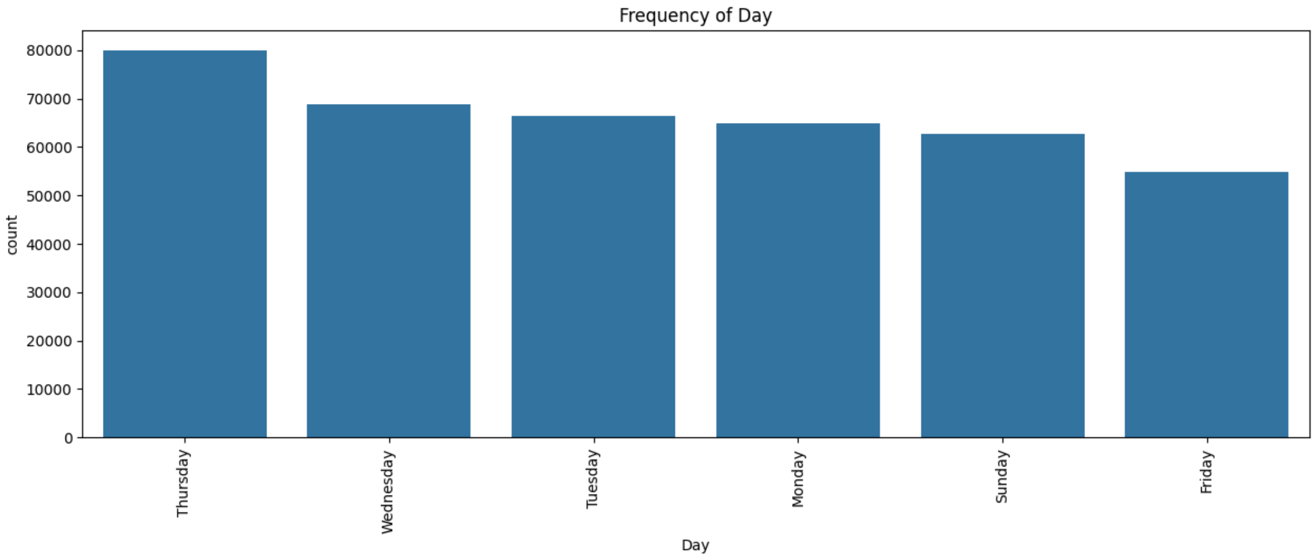



```
Value Counts for Country:
Country
United Kingdom          354321
Germany                   9040
France                    8341
EIRE                      7236
Spain                     2484
Netherlands               2359
Belgium                   2031
Switzerland               1841
Portugal                  1462
Australia                 1182
Norway                    1071
Italy                      758
Channel Islands            748
Finland                    685
Cyprus                     614
Sweden                     451
Austria                    398
Denmark                    380
Poland                     330
Japan                      321
Israel                     248
Unspecified                244
Singapore                  222
Iceland                    182
USA                        179
Canada                     151
Greece                     145
Malta                      112
United Arab Emirates        68
European Community          60
RSA                         57
Lebanon                     45
Lithuania                   35
Brazil                      32
Czech Republic              25
Bahrain                     17
Saudi Arabia                 9
Name: count, dtype: int64
```





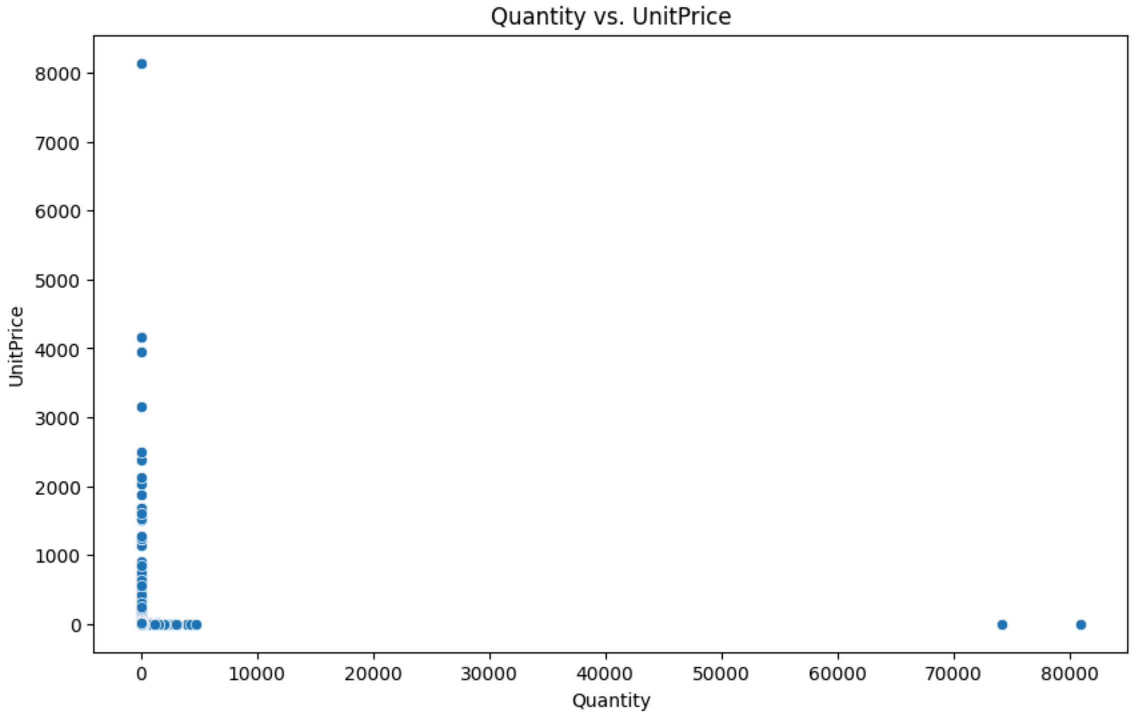

```
Value Counts for Month:
Month
November     64531
October      49554
December     43461
September    40028
May          28320
June         27185
March        27175
August       27007
July         26825
April        22642
January      21229
February     19927
Name: count, dtype: int64
```



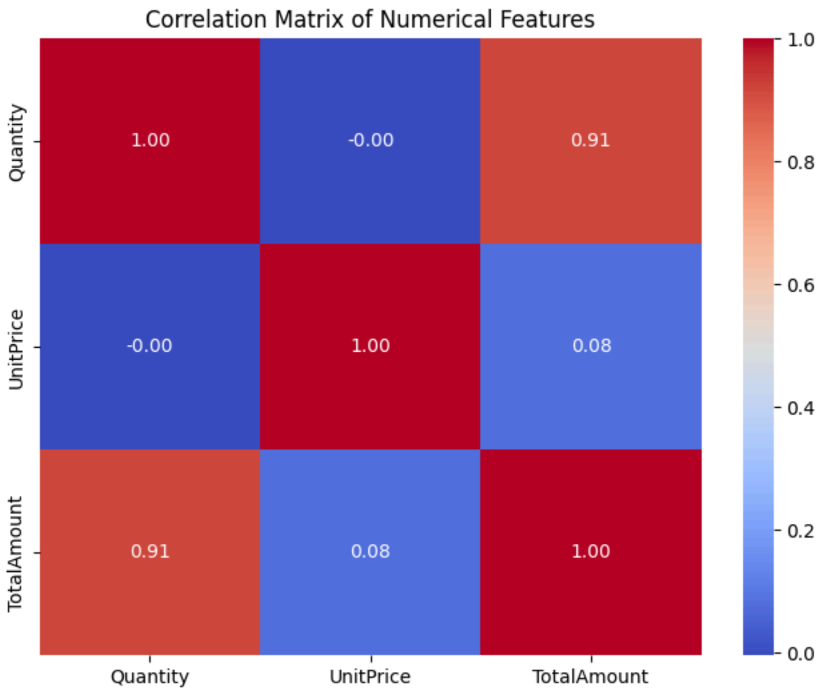

```
Value Counts for Day:
Day
Thursday     80035
Wednesday    68885
Tuesday      66473
Monday       64893
Sunday       62773
Friday       54825
Name: count, dtype: int64
```

2. Bivariate Analysis

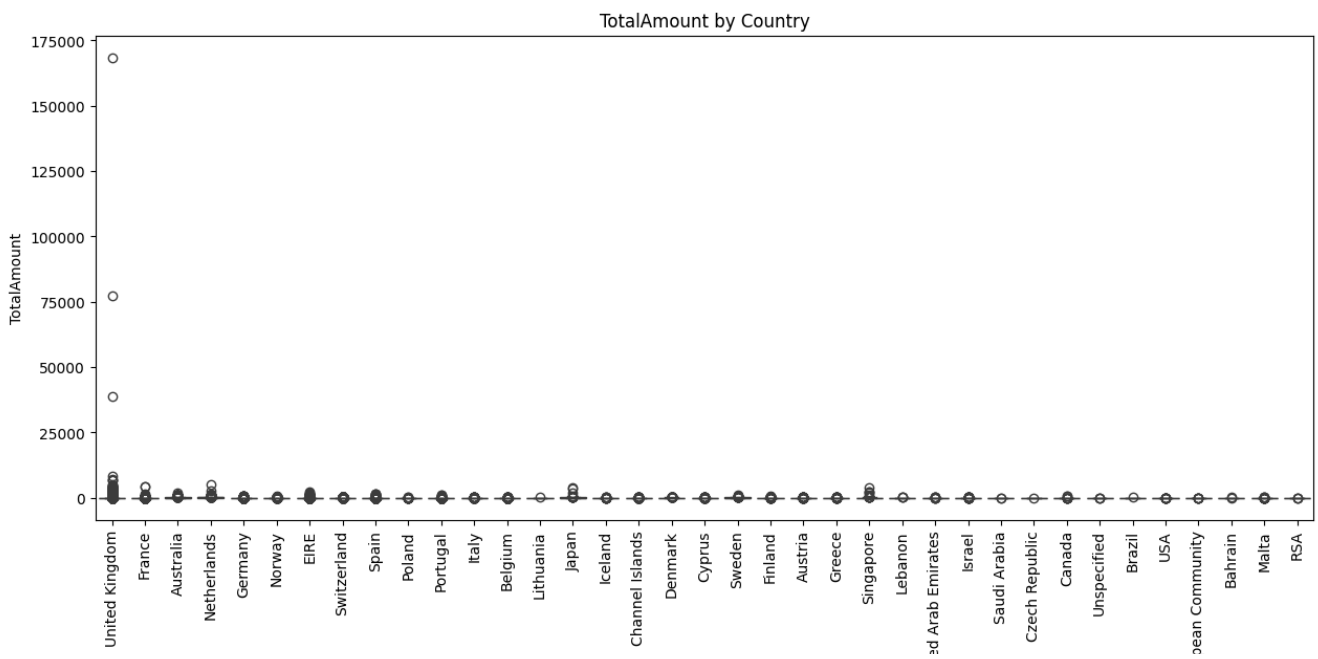
```
# Bivariate Analysis: Numerical vs. Numerical
# 1. Scatter plot: Quantity vs. UnitPrice
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Quantity', y='UnitPrice', data=retail_df)
plt.title('Quantity vs. UnitPrice')
plt.show()

# 2. Correlation matrix: Correlation between numerical features
correlation_matrix = retail_df[['Quantity', 'UnitPrice', 'TotalAmount']].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()

# Bivariate Analysis: Numerical vs. Categorical
# 1. Boxplot: TotalAmount vs. Country
plt.figure(figsize=(15, 6))
sns.boxplot(x='Country', y='TotalAmount', data=retail_df)
plt.title('TotalAmount by Country')
plt.xticks(rotation=90)
plt.show()

# 2. Bar plot: Average TotalAmount per Month
plt.figure(figsize=(12, 6))
average_amount_per_month = retail_df.groupby('Month')['TotalAmount'].mean()
sns.barplot(x=average_amount_per_month.index, y=average_amount_per_month.values)
plt.title('Average TotalAmount per Month')
plt.xticks(rotation=90)
plt.show()

# Bivariate Analysis: Categorical vs. Categorical
# 1. Contingency table and heatmap: Country vs. Month
contingency_table = pd.crosstab(retail_df['Country'], retail_df['Month'])
plt.figure(figsize=(15, 10))
sns.heatmap(contingency_table, annot=True, cmap='viridis', fmt='d')
plt.title('Country vs. Month')
plt.show()
```



Quantity vs. UnitPrice
UnitPrice
Quantity
0
1000
2000
3000
4000
5000
6000
7000
8000
0
10000
20000
30000
40000
50000
60000
70000
80000

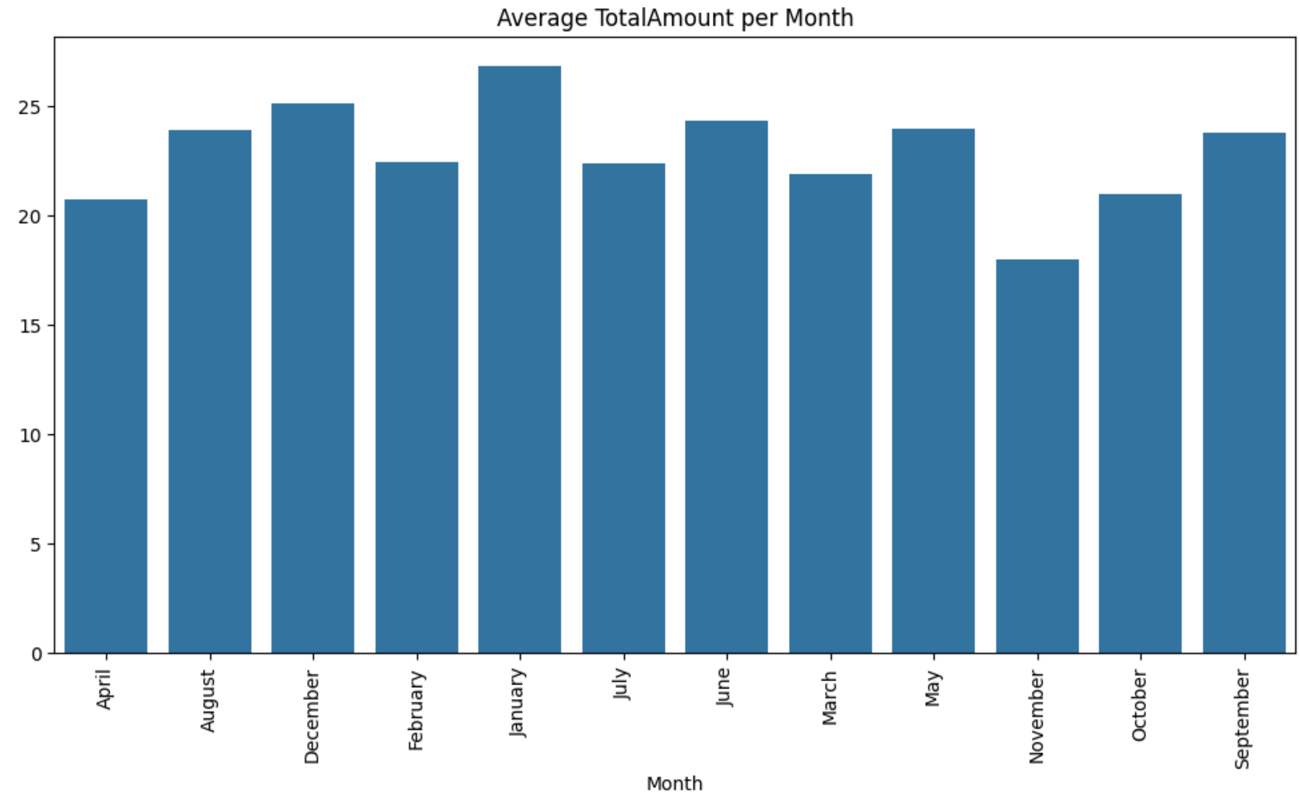


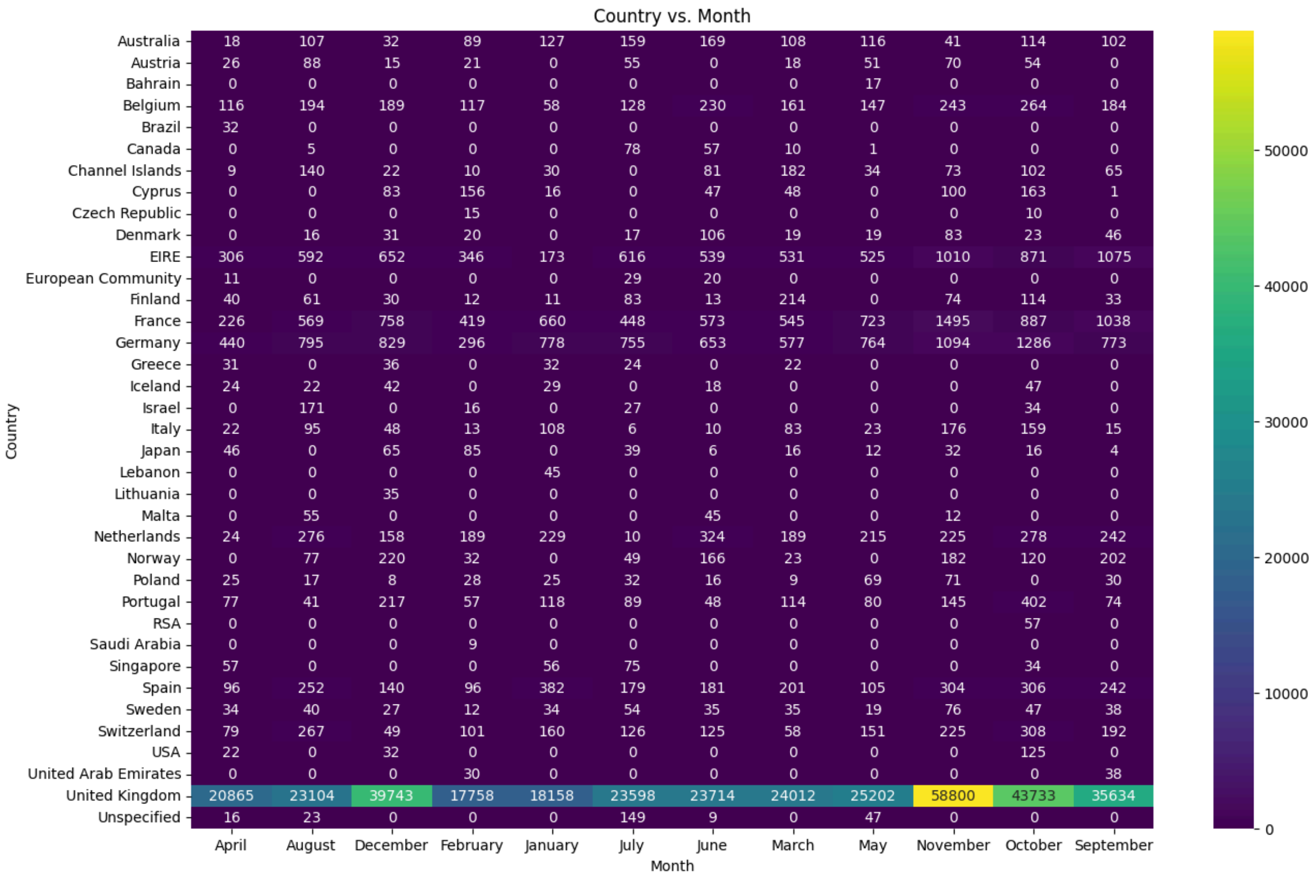
Correlation Matrix of Numerical Features
Quantity
UnitPrice
TotalAmount
1.00
-0.00
0.91
-0.00
1.00
0.08
0.91
0.08
1.00
Quantity
UnitPrice
TotalAmount
1.0
0.8
0.6
0.4
0.2
0.0



TotalAmount by Country
TotalAmount
175000
150000
125000
100000
75000
50000
25000
0
United Kingdom
France
Australia
Netherlands
Germany
Norway
EIRE
Switzerland
Spain
Poland
Portugal
Italy
Belgium
Lithuania
Japan
Iceland
Channel Islands
Denmark
Cyprus
Sweden
Finland
Austria
Greece
Singapore
Lebanon
ed Arab Emirates
Israel
Saudi Arabia
Czech Republic
Canada
Unspecified
Brazil
USA
ean Community
Bahrain
Malta
RSA


Country


Average TotalAmount per Month
0
5
10
15
20
25
April
August
December
February
January
July
June
March
May
November
October
September
Month

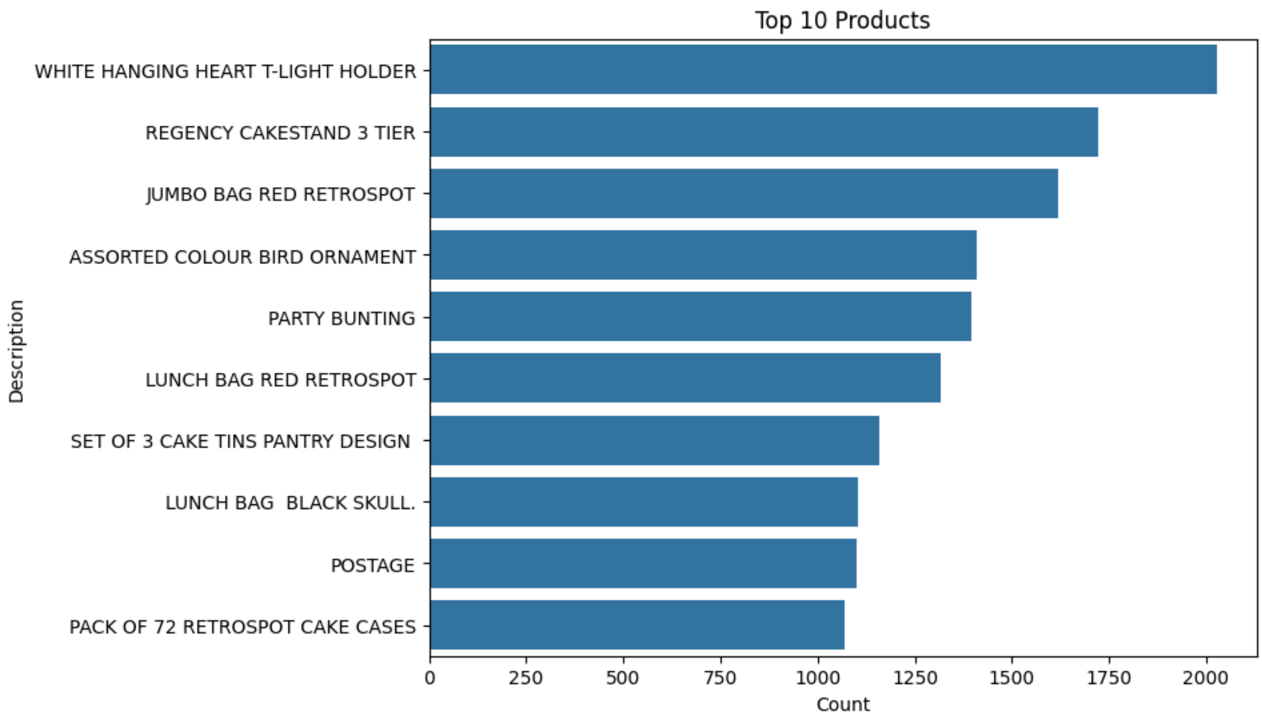

Country vs. Month
Australia 18 107 32 89 127 159 169 108 116 41 114 102
Austria 26 88 15 21 0 55 0 18 51 70 54 0
Bahrain 0 0 0 0 0 0 0 0 17 0 0 0
Belgium 116 194 189 117 58 128 230 161 147 243 264 184
Brazil 32 0 0 0 0 0 0 0 0 0 0 0
Canada 0 5 0 0 0 78 57 10 1 0 0 0
Channel Islands 9 140 22 10 30 0 81 182 34 73 102 65
Cyprus 0 0 83 156 16 0 47 48 0 100 163 1
Czech Republic 0 0 0 15 0 0 0 0 0 0 10 0
Denmark 0 16 31 20 0 17 106 19 19 83 23 46
EIRE 306 592 652 346 173 616 539 531 525 1010 871 1075
European Community 11 0 0 0 0 29 20 0 0 0 0 0
Finland 40 61 30 12 11 83 13 214 0 74 114 33
France 226 569 758 419 660 448 573 545 723 1495 887 1038
Germany 440 795 829 296 778 755 653 577 764 1094 1286 773
Greece 31 0 36 0 32 24 0 22 0 0 0 0
Iceland 24 22 42 0 29 0 18 0 0 0 47 0
Israel 0 171 0 16 0 27 0 0 0 0 34 0
Italy 22 95 48 13 108 6 10 83 23 176 159 15
Japan 46 0 65 85 0 39 6 16 12 32 16 4
Lebanon 0 0 0 0 45 0 0 0 0 0 0 0
Lithuania 0 0 35 0 0 0 0 0 0 0 0 0
Malta 0 55 0 0 0 0 45 0 0 12 0 0
Netherlands 24 276 158 189 229 10 324 189 215 225 278 242
Norway 0 77 220 32 0 49 166 23 0 182 120 202
Poland 25 17 8 28 25 32 16 9 69 71 0 30
Portugal 77 41 217 57 118 89 48 114 80 145 402 74
RSA 0 0 0 0 0 0 0 0 0 0 57 0
Saudi Arabia 0 0 0 9 0 0 0 0 0 0 0 0
Singapore 57 0 0 0 56 75 0 0 0 0 34 0
Spain 96 252 140 96 382 179 181 201 105 304 306 242
Sweden 34 40 27 12 34 54 35 35 19 76 47 38
Switzerland 79 267 49 101 160 126 125 58 151 225 308 192
USA 22 0 32 0 0 0 0 0 0 0 125 0
United Arab Emirates 0 0 0 30 0 0 0 0 0 0 0 38
United Kingdom 20865 23104 39743 17758 18158 23598 23714 24012 25202 58800 43733 35634
Unspecified 16 23 0 0 0 149 9 0 47 0 0 0
April August December February January July June March May November October September
Month
Country
0
10000
20000
30000
40000
50000

- **Top 10 items in terms of description(Name)**

```
top_10_product=retail_df['Description'].value_counts().reset_index().rename(columns={'index':'Product_name','Description':'Count'}).head
top_10_product
```

	Count	count
0	WHITE HANGING HEART T-LIGHT HOLDER	2028
1	REGENCY CAKESTAND 3 TIER	1723
2	JUMBO BAG RED RETROSPOT	1618
3	ASSORTED COLOUR BIRD ORNAMENT	1408
4	PARTY BUNTING	1396
5	LUNCH BAG RED RETROSPOT	1316
6	SET OF 3 CAKE TINS PANTRY DESIGN	1159
7	LUNCH BAG BLACK SKULL.	1105
8	POSTAGE	1099
9	PACK OF 72 RETROSPOT CAKE CASES	1068

```
top_10_product = retail_df.groupby('Description').size().reset_index(name='Count').sort_values(by='Count', ascending=False).head(10)
# Plotting the top 10 products
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,6))
sns.barplot(x='Count', y='Description', data=top_10_product)
plt.title('Top 10 Products')
plt.show()
```



Observations

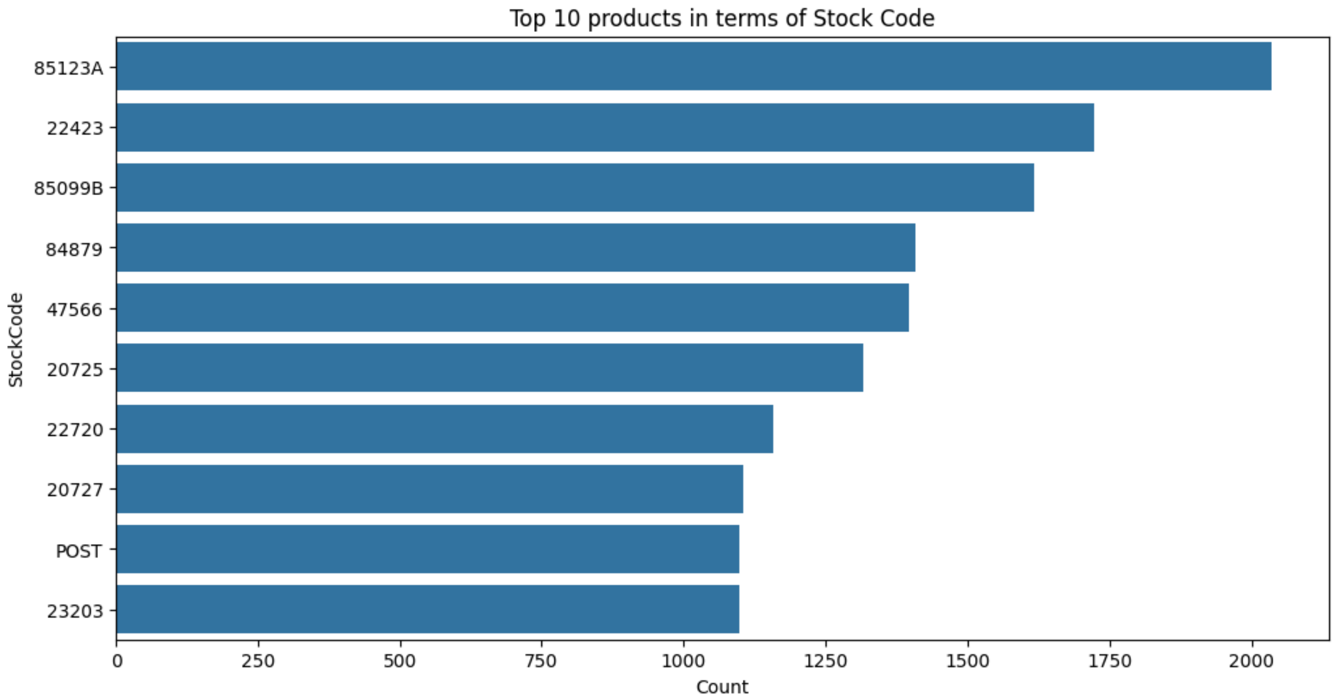
- WHITE HANGING HEART T-LIGHT HOLDER is the highest selling product almost 2018 units were sold
- REGENCY CAKESTAND 3 TIER is the 2nd highest selling product almost 1723 units were sold

Top 10 items in terms of StockCode.

```
top_10_StockCodes=retail_df.groupby('StockCode').size().reset_index(name='Count').sort_values(by='Count',ascending=False).head(10)
```

```
# top 10 product in terms of StcokCode
plt.figure(figsize=(12,6))
sns.barplot(x=top_10_StockCodes['Count'],y=top_10_StockCodes['StockCode'])
plt.title('Top 10 products in terms of Stock Code')
```

```
Text(0.5, 1.0, 'Top 10 products in terms of Stock Code')
```



Observations

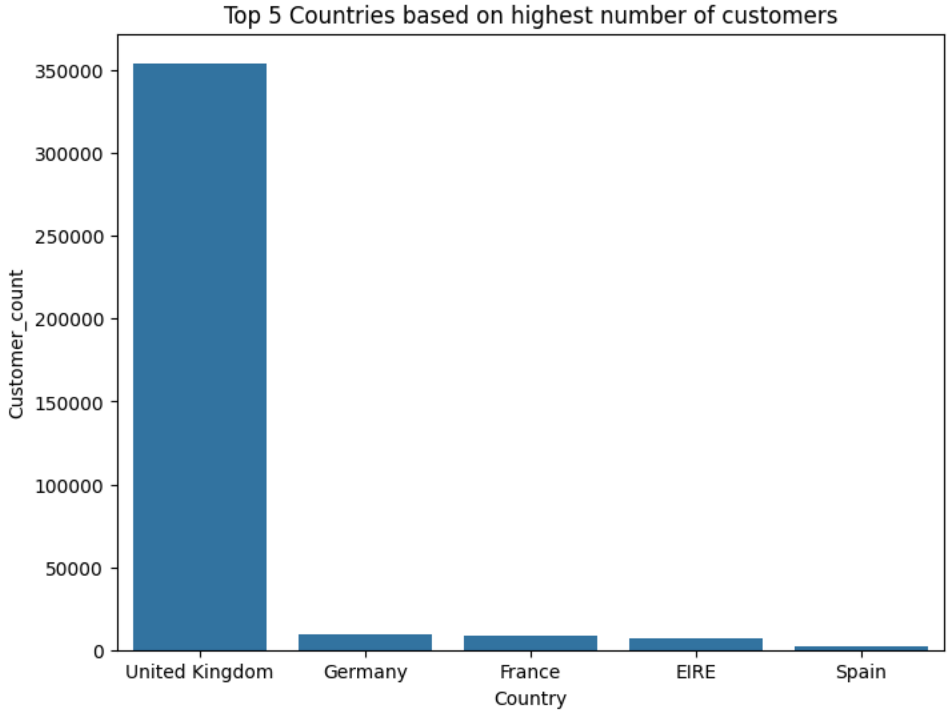
- StockCode-85123Ais the first highest selling product.
- StockCode-22423 is the 2nd highest selling product.

Top 5 countries with highest number of customers

```
top_5_countries=retail_df.groupby('Country').size().reset_index(name='Customer_count').sort_values(by='Customer_count',ascending=False).
```

```
# top 5 countries where max sell happens.
plt.figure(figsize=(8,6))
sns.barplot(x=top_5_countries['Country'].head(5),y=top_5_countries['Customer_count'].head(5))
plt.title('Top 5 Countries based on highest number of customers')
```

```
Text(0.5, 1.0, 'Top 5 Countries based on highest number of customers')
```



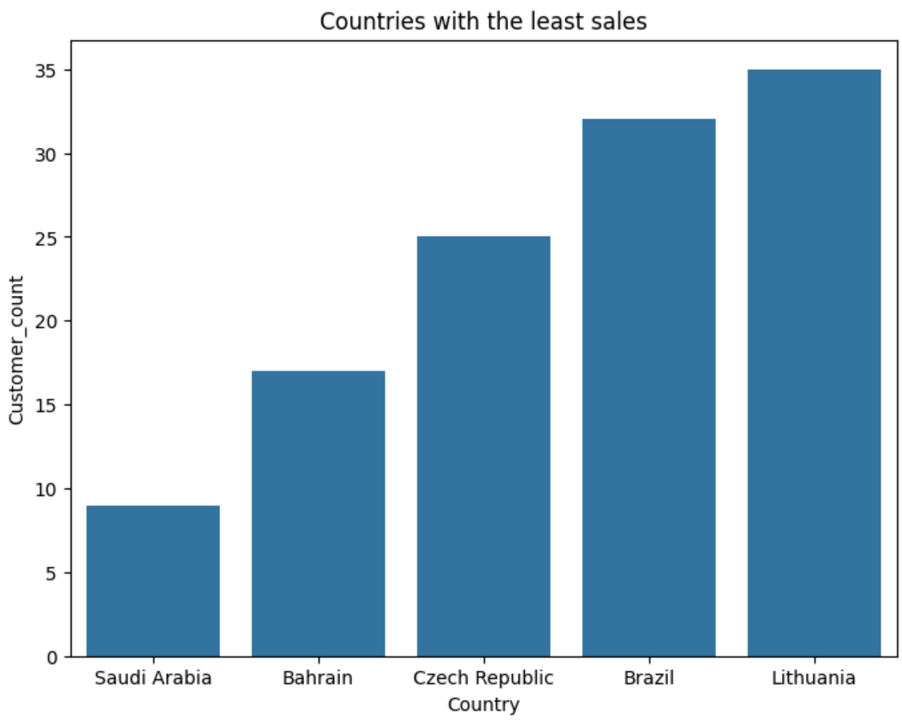
Observation

- UK has highest number of customers
- Germany,France and IreLand has almost equal number of customers

```
# top 5 countries where max sell happens.
bottom_5_countries=retail_df.groupby('Country').size().reset_index(name='Customer_count').sort_values(by='Customer_count',ascending=True
bottom_5_countries
```

	Country	Customer_count
28	Saudi Arabia	9
2	Bahrain	17
8	Czech Republic	25
4	Brazil	32
21	Lithuania	35

```
# barplot of countries with the least cutomers
plt.figure(figsize=(8,6))
sns.barplot(x=bottom_5_countries['Country'].head(5),y=bottom_5_countries['Customer_count'].head(5))
plt.title('Countries with the least sales');
```



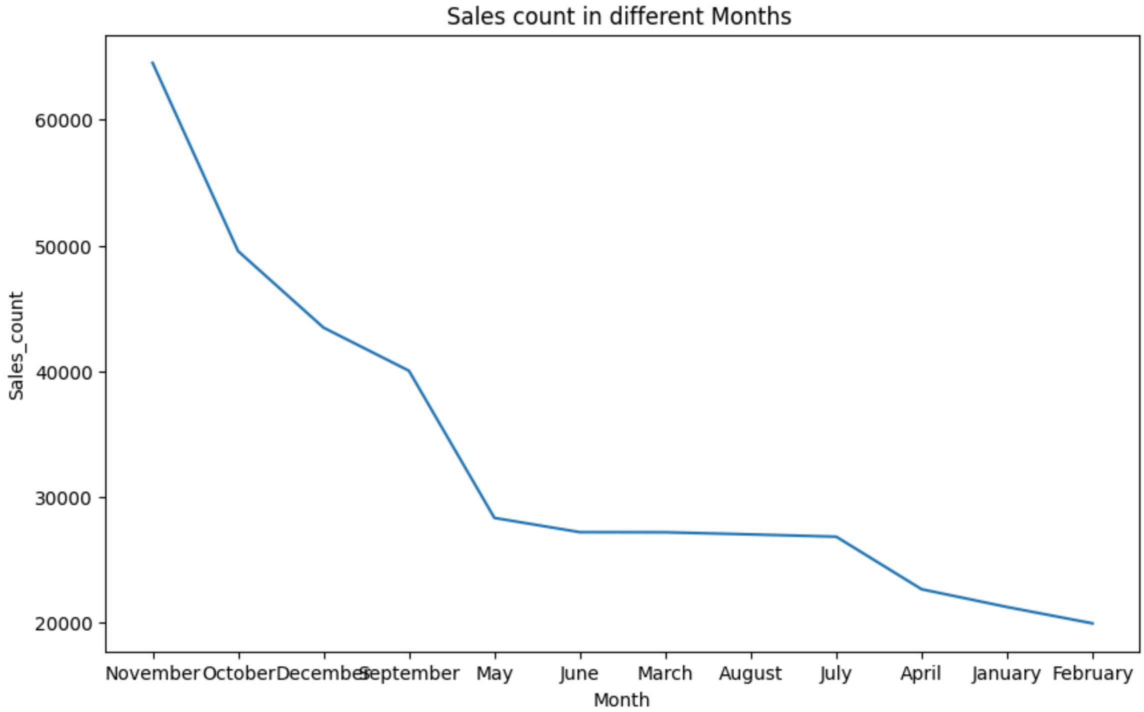
Observations

- There are very less customers from Saudi Arabia
- Bahrain is the 2nd Country having least number of customers

```
sales_in_month=retail_df.groupby('Month').size().reset_index(name='Sales_count').sort_values(by='Sales_count',ascending=False)
sales_in_month
```

	Month	Sales_count
9	November	64531
10	October	49554
2	December	43461
11	September	40028
8	May	28320
6	June	27185
7	March	27175
1	August	27007
5	July	26825
0	April	22642
4	January	21229
3	February	19927

```
# Sales count in different months.
plt.figure(figsize=(10,6))
sns.lineplot(x=sales_in_month['Month'],y=sales_in_month['Sales_count'])
plt.title('Sales count in different Months ');
```



Observations

- Most of the sale happened in Novmenber month.
- February Month had least sales.

Data Preprocessing

```
retail_df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	year	month_num	day_num	hour	minute
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010	12	1	8	26
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010	12	1	8	26
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010	12	1	8	26
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010	12	1	8	26
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010	12	1	8	26

```
retail_df.drop_duplicates(inplace=True)
```

Model Building

RFM Model Analysis:

What is RFM?

RFM (Recency, Frequency, Monetary) analysis is a widely used customer segmentation technique in marketing and analytics. It helps businesses understand and categorize their customers based on three key factors:

- How recently they made a purchase **(Recency)**,
- How frequently they make purchases **(Frequency)**,
- How much they spend **(Monetary value)**.

RFM analysis enables businesses to identify and target different customer segments with customized marketing approaches.

Why it is Needed?

RFM Analysis is a marketing framework that is used to understand and analyze customer behaviour based on the above three factors RECENCY, Frequency, and Monetary.

The RFM Analysis will help the businesses to segment their customer base into different homogenous groups so that they can engage with each group with different targeted marketing strategies.

```
df=retail_df.copy()
```

```
df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	year	month_num	day_num	hour	minute
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010	12	1	8	26
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010	12	1	8	26
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010	12	1	8	26
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010	12	1	8	26
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010	12	1	8	26

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 392692 entries, 0 to 541908
Data columns (total 16 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    392692 non-null  object
 1   StockCode    392692 non-null  object
 2   Description  392692 non-null  object
 3   Quantity     392692 non-null  int64
 4   InvoiceDate  392692 non-null  datetime64[ns]
 5   UnitPrice    392692 non-null  float64
 6   CustomerID   392692 non-null  float64
 7   Country      392692 non-null  object
 8   year         392692 non-null  int64
 9   month_num    392692 non-null  int64
 10  day_num      392692 non-null  int64
 11  hour         392692 non-null  int64
 12  minute       392692 non-null  int64
 13  Month        392692 non-null  object
 14  Day          392692 non-null  object
 15  TotalAmount  392692 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(6), object(6)
memory usage: 50.9+ MB
```

- RFM (Recency, Frequency, Monetary) analysis is a popular marketing technique to segment customers based on their purchasing behavior. It focuses on three factors:
- Recency (R): How recently a customer made a purchase.
- Frequency (F): How often a customer makes a purchase.
- Monetary (M): How much money a customer has spent in total.

```
# Set a reference date for Recency calculation
# You need to decide on a reference date (usually the most recent transaction date in your dataset) to compute the Recency metric
# we'll use the latest InvoiceDate in the dataset as the reference.
reference_date = df['InvoiceDate'].max()
print("Reference date:", reference_date)
##calculating recency
# Recency refers to the number of days since the customer's last purchase.
# Group by CustomerID and calculate Recency as the difference in days from the reference date
# Group by CustomerID and calculate Recency as the difference in days from the reference date
recency_df = df.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (reference_date - x.max()).days
}).reset_index()

# Rename the column for clarity
recency_df.columns = ['CustomerID', 'Recency']
```

```
Reference date: 2011-12-09 12:50:00
```

- **Calculate Frequency**

Frequency is the number of unique purchases made by each customer. We count the number of unique InvoiceNo entries per customer.

```
# Group by CustomerID and count unique InvoiceNo
frequency_df = df.groupby('CustomerID').agg({
    'InvoiceNo': 'nunique'
}).reset_index()

# Rename the column
frequency_df.columns = ['CustomerID', 'Frequency']
```

- **Calculate Monetary Value**

Monetary value is the total amount of money each customer has spent. We'll sum the TotalAmount per customer.

```
# Group by CustomerID and sum TotalAmount to calculate Monetary value
monetary_df = df.groupby('CustomerID').agg({
    'TotalAmount': 'sum'
}).reset_index()

# Rename the column
monetary_df.columns = ['CustomerID', 'Monetary']
```

- **Merge R, F, M Metrics**

Now that we have calculated Recency, Frequency, and Monetary values, let's combine them into a single DataFrame.

```
# Merge Recency, Frequency, and Monetary dataframes
rfm_df = recency_df.merge(frequency_df, on='CustomerID').merge(monetary_df, on='CustomerID')

# Inspect the combined RFM data
rfm_df.head()
```

	CustomerID	Recency	Frequency	Monetary
0	12346.0	325	1	77183.60
1	12347.0	1	7	4310.00
2	12348.0	74	4	1797.24
3	12349.0	18	1	1757.55
4	12350.0	309	1	334.40

- **Scoring the RFM Metrics**

To standardize the RFM values, we'll assign scores between 1 and 5 using quantiles.

```
# Score Recency
# Lower Recency values are better, so we assign higher scores for lower Recency.
rfm_df['R_Score'] = pd.qcut(rfm_df['Recency'], 5, labels=[5, 4, 3, 2, 1])

# : Score Frequency
# Higher Frequency values are better, so we assign higher scores for higher Frequency.
rfm_df['F_Score'] = pd.qcut(rfm_df['Frequency'].rank(method='first'), 5, labels=[1, 2, 3, 4, 5])

# .3: Score Monetary
# Higher Monetary values are better, so we assign higher scores for higher Monetary values.
rfm_df['M_Score'] = pd.qcut(rfm_df['Monetary'], 5, labels=[1, 2, 3, 4, 5])

# We can now combine the R_Score, F_Score, and M_Score to create a unified RFM score. This score can be used to segment customers
# Concatenate R, F, M scores
rfm_df['RFM_Score'] = rfm_df['R_Score'].astype(str) + rfm_df['F_Score'].astype(str) + rfm_df['M_Score'].astype(str)

# Display the first few rows of the final RFM data
rfm_df.head()
```

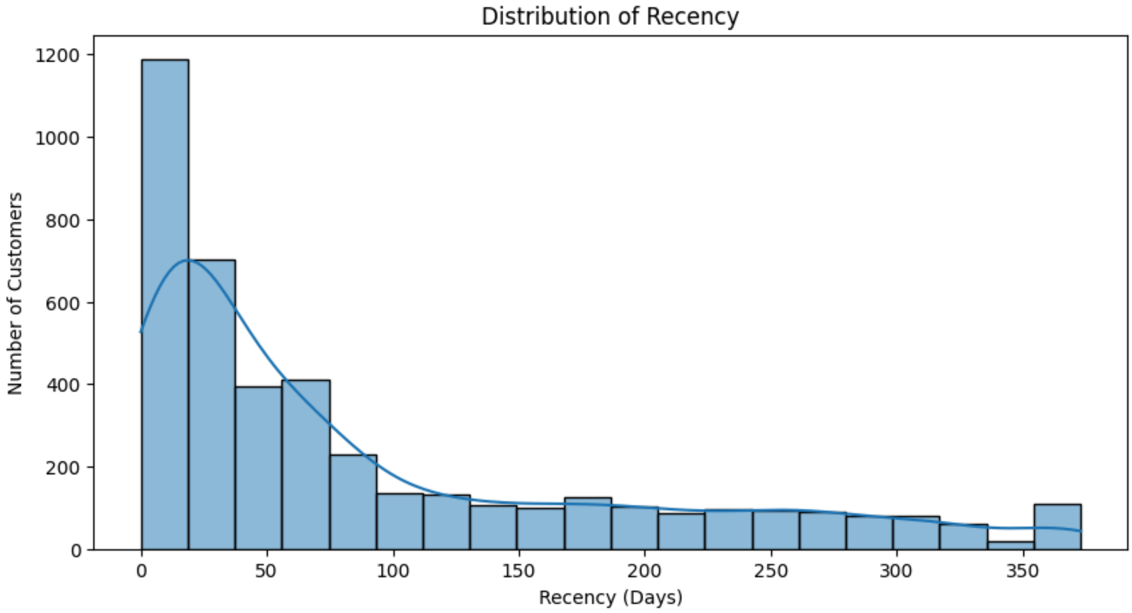
	CustomerID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score
0	12346.0	325	1	77183.60	1	1	5	115
1	12347.0	1	7	4310.00	5	5	5	555
2	12348.0	74	4	1797.24	2	4	4	244
3	12349.0	18	1	1757.55	4	1	4	414
4	12350.0	309	1	334.40	1	1	2	112

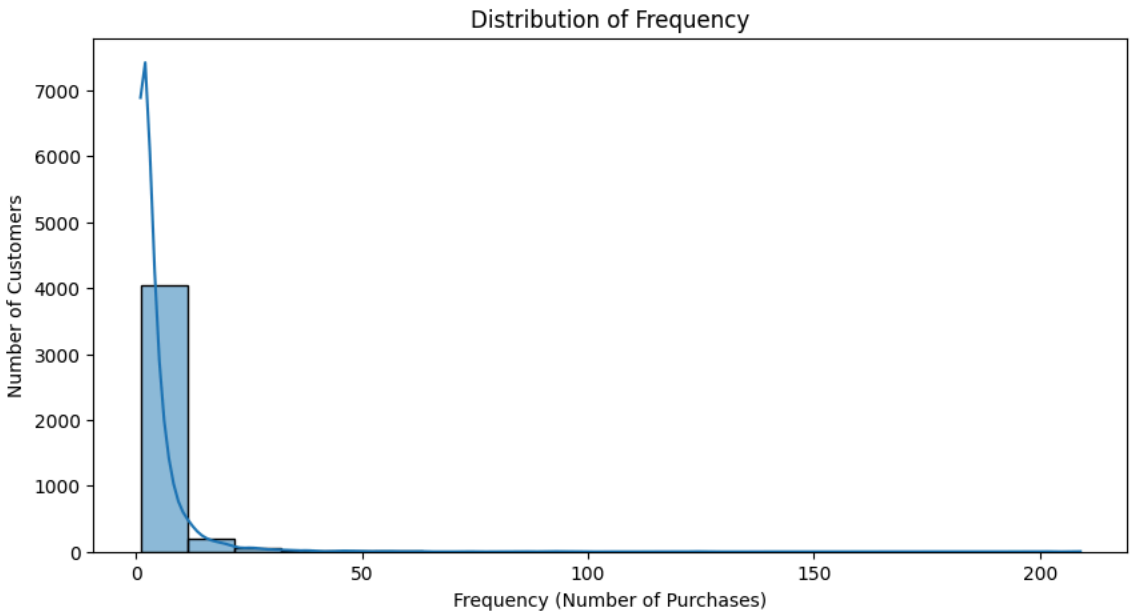
```
# Sum up R_Score, F_Score, and M_Score
rfm_df['RFM_Sum'] = rfm_df[['R_Score', 'F_Score', 'M_Score']].sum(axis=1)
# Categorize customers based on quartiles of the RFM sum
rfm_df['Customer_Category'] = pd.cut(
    rfm_df['RFM_Sum'],
    bins=[0, 5, 10, 15, 20],  # Adjust based on your data distribution
    labels=['Low', 'Medium', 'High', 'Very High']
)
rfm_df.head()
```

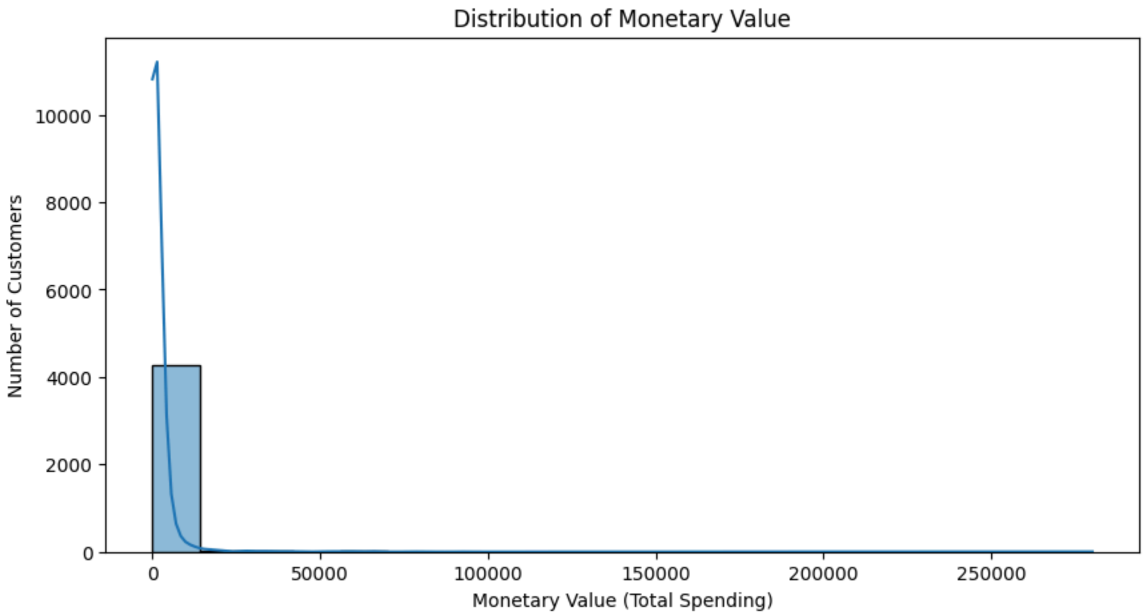
	CustomerID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	RFM_Sum	Customer_Category
0	12346.0	325	1	77183.60	1	1	5	115	7	Medium
1	12347.0	1	7	4310.00	5	5	5	555	15	High
2	12348.0	74	4	1797.24	2	4	4	244	10	Medium
3	12349.0	18	1	1757.55	4	1	4	414	9	Medium
4	12350.0	309	1	334.40	1	1	2	112	4	Low

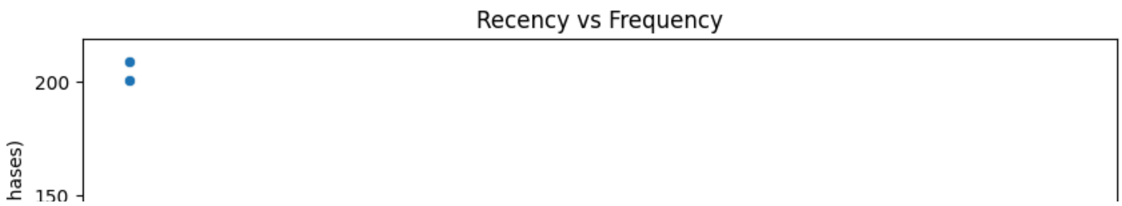
```
# visual representation of the distribution of recency, frequency and monetary

import matplotlib.pyplot as plt
import seaborn as sns

def visualize_rfm_distribution(rfm_df):
  """
  Creates visual representations of the distribution of Recency, Frequency, and Monetary value.

  Args:
    rfm_df: A pandas DataFrame containing RFM metrics (Recency, Frequency, Monetary).
  """

  # Histogram of Recency
  plt.figure(figsize=(10, 5))
  sns.histplot(rfm_df['Recency'], bins=20, kde=True)
  plt.title('Distribution of Recency')
  plt.xlabel('Recency (Days)')
  plt.ylabel('Number of Customers')
  plt.show()

  # Histogram of Frequency
  plt.figure(figsize=(10, 5))
  sns.histplot(rfm_df['Frequency'], bins=20, kde=True)
```

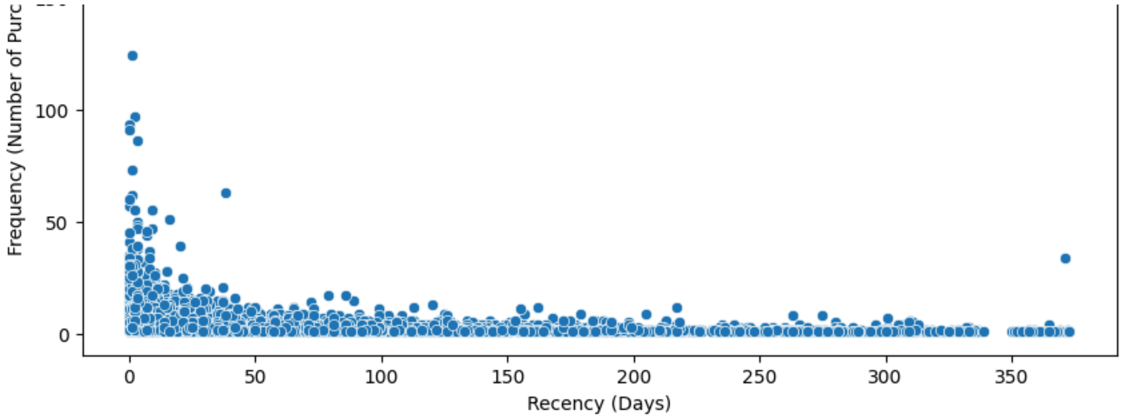
```
    plt.title('Distribution of Frequency')
    plt.xlabel('Frequency (Number of Purchases)')
    plt.ylabel('Number of Customers')
    plt.show()

    # Histogram of Monetary Value
    plt.figure(figsize=(10, 5))
    sns.histplot(rfm_df['Monetary'], bins=20, kde=True)
    plt.title('Distribution of Monetary Value')
    plt.xlabel('Monetary Value (Total Spending)')
    plt.ylabel('Number of Customers')
    plt.show()

    # Scatter plot of Recency vs Frequency
    plt.figure(figsize=(10, 5))
    sns.scatterplot(x='Recency', y='Frequency', data=rfm_df)
    plt.title('Recency vs Frequency')
    plt.xlabel('Recency (Days)')
    plt.ylabel('Frequency (Number of Purchases)')
    plt.show()

    # Scatter plot of Frequency vs Monetary Value
    plt.figure(figsize=(10, 5))
    sns.scatterplot(x='Frequency', y='Monetary', data=rfm_df)
    plt.title('Frequency vs Monetary Value')
    plt.xlabel('Frequency (Number of Purchases)')
    plt.ylabel('Monetary Value (Total Spending)')
    plt.show()

    # Scatter plot of Recency vs Monetary Value
    plt.figure(figsize=(10, 5))
    sns.scatterplot(x='Recency', y='Monetary', data=rfm_df)
    plt.title('Recency vs Monetary Value')
    plt.xlabel('Recency (Days)')
    plt.ylabel('Monetary Value (Total Spending)')
    plt.show()

# Assuming your RFM dataframe is named 'rfm_df'
visualize_rfm_distribution(rfm_df)
```

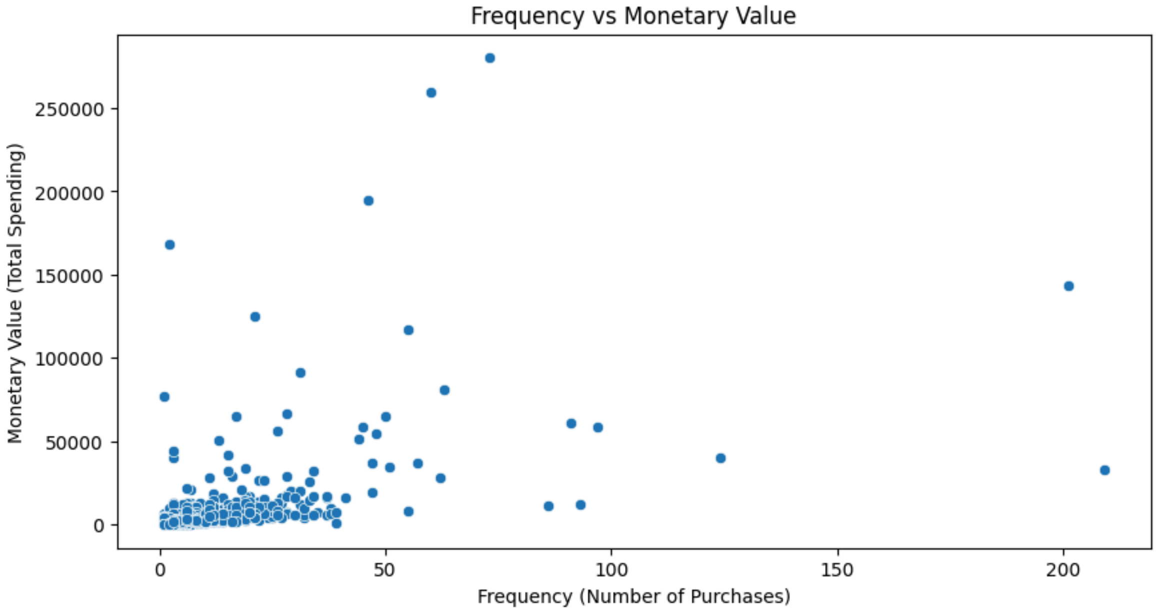

Distribution of Recency
Number of Customers
Recency (Days)

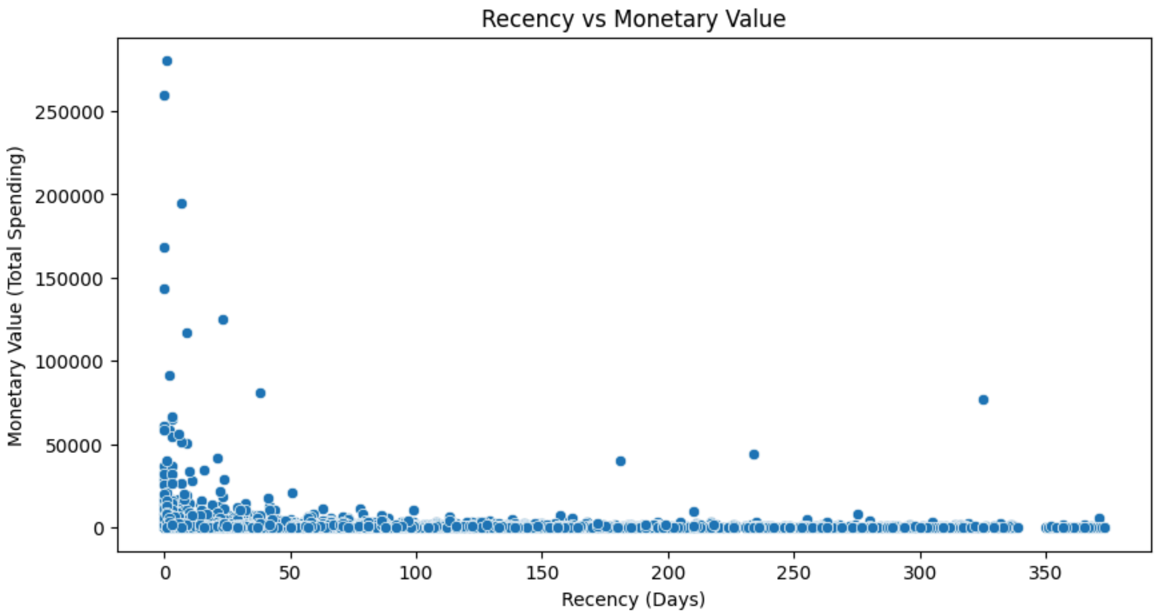

Distribution of Frequency
Number of Customers
Frequency (Number of Purchases)


Distribution of Monetary Value
Number of Customers
Monetary Value (Total Spending)


Recency vs Frequency


Frequency (Number of Purc
Recency (Days)


Frequency vs Monetary Value
Monetary Value (Total Spending)
Frequency (Number of Purchases)


Recency vs Monetary Value
Monetary Value (Total Spending)
Recency (Days)

```
#perform log transfformation to reduce the skewness of the columns above
rfm_df['Recency_log']= np.log(rfm_df.Recency)
rfm_df['Frequency_log'] = np.log(rfm_df.Frequency)
rfm_df['Monetary_log']= np.log(rfm_df.Monetary)
```

```
rfm_df.head()
```

	CustomerID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	RFM_Sum	Customer_Category	Recency_log	Frequency
0	12346.0	325	1	77183.60	1	1	5	115	7	Medium	5.783825	0.00
1	12347.0	1	7	4310.00	5	5	5	555	15	High	0.000000	1.94
2	12348.0	74	4	1797.24	2	4	4	244	10	Medium	4.304065	1.38
3	12349.0	18	1	1757.55	4	1	4	414	9	Medium	2.890372	0.00
4	12350.0	309	1	334.40	1	1	2	112	4	Low	5.733341	0.00

```
rfm_df['Recency_log'] = rfm_df['Recency'].round(2)
rfm_df['Frequency_log'] = rfm_df['Frequency'].round(2)
rfm_df['Monetary_log'] = rfm_df['Monetary'].round(2)
```

```
from sklearn.preprocessing import StandardScaler

# Select relevant columns for clustering
X = rfm_df[['Recency_log', 'Frequency_log', 'Monetary_log']]

# Standardize the data (mean = 0, variance = 1)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# import numpy as np

# Check for NaN values
print("NaN values in dataset: \n", np.isnan(X_scaled).sum())

# Check for infinite values
print("Infinite values in dataset: \n", np.isinf(X_scaled).sum())
# Replace infinite values with NaN
X_scaled = np.where(np.isinf(X_scaled), np.nan, X_scaled)
# Optionally, you can fill NaN values with the column mean
X_scaled = np.nan_to_num(X_scaled, nan=np.nanmean(X_scaled))
```

```
NaN values in dataset: 
 0
Infinite values in dataset: 
 0
```