Market Basket Analysis with RFM analysis

Market Basket Analysis decodes customer purchasing patterns through transactional data, revealing item associations. When integrated with RFM analysis, businesses gain a holistic understanding to alter strategies and enhance customer satisfaction. Our project merges Market Basket Analysis and RFM (Recency, Frequency, Monetary Value) Analysis to decode customer purchasing patterns. This combined approach uncovers correlations in product purchases and customer behavior, empowering us to tailor strategies for enhanced customer satisfaction. By leveraging these insights, we aim to optimize product placement, marketing strategies, and promotions to meet diverse customer preferences, fostering stronger relationships and loyalty

Importing the Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from sklearn.cluster import KMeans #For customer segmentation
from sklearn.preprocessing import StandardScaler
import warnings
from scipy.stats import pearsonr
from scipy.sparse import csr_matrix
from yellowbrick.cluster import KElbowVisualizer
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
import time
warnings.filterwarnings("ignore")
```

```
#Proprocessing the dataset
df1 = pd.read_excel('/content/online_retail_09_10.xlsx')
df2 = pd.read_excel('/content/online_retail_10_11.xlsx')
df = pd.concat([df1, df2])
df
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Count
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	Unit Kingdc
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	Unit Kingdc
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	Unit Kingdc
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	Unit Kingdc
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	Unit Kingdc
...	...	...	...	...	...	...	...	

Understanding the dataset

In this dataset we have the following rows

1. InvoiceNo: Invoice number.
2. StockCode: Product (item) code. Nominal. A 5-digit integral number uniquely assigned to each distinct product.
3. Description: Product (item) name. Nominal.
4. Quantity: The quantities of each product (item) per transaction. Numeric.

5. InvoiceDate: Invoice date and time. Numeric. The day and time when a transaction was generated.
6. UnitPrice: Unit price. Numeric. Product price per unit in sterling (Â£).
7. CustomerID: Customer number. Nominal. A 5-digit integral number uniquely assigned to each customer.
8. Country: Country name. Nominal. The name of the country where a customer resides.

```
print(f"Shape of the dataframe : {df.shape}")
print(f"Number of transactions : {df.shape[0]}")
print(f"Number of features used : {df.shape[1]}")
```

```
Shape of the dataframe : (1067371, 8)
Number of transactions : 1067371
Number of features used : 8
```

Datatype of each column

```
df.dtypes
```

```
Invoice               object
StockCode             object
Description           object
Quantity               int64
InvoiceDate    datetime64[ns]
Price                 float64
Customer ID           float64
Country                object
dtype: object
```

- The invoice date is in the correct datetime format
- Customer ID can be converted to int/str instead of float as it constitutes of only numbers - for that we'll have to check if it contains NULL values

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1067371 entries, 0 to 541909
Data columns (total 8 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   Invoice      1067371 non-null  object
 1   StockCode    1067371 non-null  object
 2   Description  1062989 non-null  object
 3   Quantity     1067371 non-null  int64
 4   InvoiceDate  1067371 non-null  datetime64[ns]
 5   Price        1067371 non-null  float64
 6   Customer ID  824364 non-null   float64
 7   Country      1067371 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 73.3+ MB
```

Check for NULL values

Here we can see that the 'Description' and 'Customer ID' columns have null values.

```
# Check how many columns have null values in Description and Customer ID
df.isna().sum()
```

```
Invoice             0
StockCode           0
Description      4382
Quantity            0
InvoiceDate         0
Price               0
Customer ID    243007
Country             0
dtype: int64
```

```
# Display the columns where Customer ID is Null
df[df['Customer ID'].isna()]
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Count
263	489464	21733	85123a mixed	-96	2009-12-01 10:52:00	0.00	NaN	Unit Kingdo
283	489463	71477	short	-240	2009-12-01 10:52:00	0.00	NaN	Unit Kingdo
284	489467	85123A	21733 mixed	-192	2009-12-01 10:53:00	0.00	NaN	Unit Kingdo
470	489521	21646	NaN	-50	2009-12-01 11:44:00	0.00	NaN	Unit Kingdo
577	489525	85226C	BLUE PULL BACK RACING CAR	1	2009-12-01 11:49:00	0.55	NaN	Unit Kingdo
...	...	...	...	...	...	...	...	
			JUMBO BAG		2011-12-09			Unit

```
df.dropna(subset = 'Customer ID', axis=0, inplace = True)
print(f'Shape after removing null values from the dataset : {df.shape}')
```

```
Shape after removing null values from the dataset : (824364, 8)
```

```
df.isna().sum()
```

```
Invoice        0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
Price          0
Customer ID    0
Country        0
dtype: int64
```

There are no missing values now

Check for negative values

```
df.describe()
```

	Quantity	Price	Customer ID
count	824364.000000	824364.000000	824364.000000
mean	12.414574	3.676800	15324.638504
std	188.976099	70.241388	1697.464450
min	-80995.000000	0.000000	12346.000000
25%	2.000000	1.250000	13975.000000
50%	5.000000	1.950000	15255.000000
75%	12.000000	3.750000	16797.000000
max	80995.000000	38970.000000	18287.000000

The above table indicated there are negative values in the 'Quantity' and 'Price' column. We'll have to remove those as these indicate cancelled or returned products

```
# These many columns contain negative values
df[(df['Quantity'] <= 0)]
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Countr
178	C489449	22087	PAPER BUNTING WHITE LACE	-12	2009-12-01 10:33:00	2.95	16321.0	Austral
179	C489449	85206A	CREAM FELT EASTER EGG BASKET	-6	2009-12-01 10:33:00	1.65	16321.0	Austral
180	C489449	21895	POTTING SHED SOW 'N' GROW SET	-4	2009-12-01 10:33:00	4.25	16321.0	Austral

```
df['Invoice'].str.contains('C').sum()
```

```
18744
```

PAPER

```
df = df[~df['Invoice'].str.contains('C', na=False)]
df
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Count
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	Unit Kingdc
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	Unit Kingdc
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	Unit Kingdc
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	Unit Kingdc
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	Unit Kingdc
...	...	...	...	...	...	...	...	...

```
df['Description'] = df['Description'].str.strip()
```

We have taken care of all the negative values in the dataframe

Remove duplicate values

```
df.duplicated().sum()
```

```
26125
```

```
df = df.drop_duplicates()
df.duplicated().sum()
```

```
0
```

```
df
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Count
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	Unit Kingdc
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	Unit Kingdc
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	Unit Kingdc

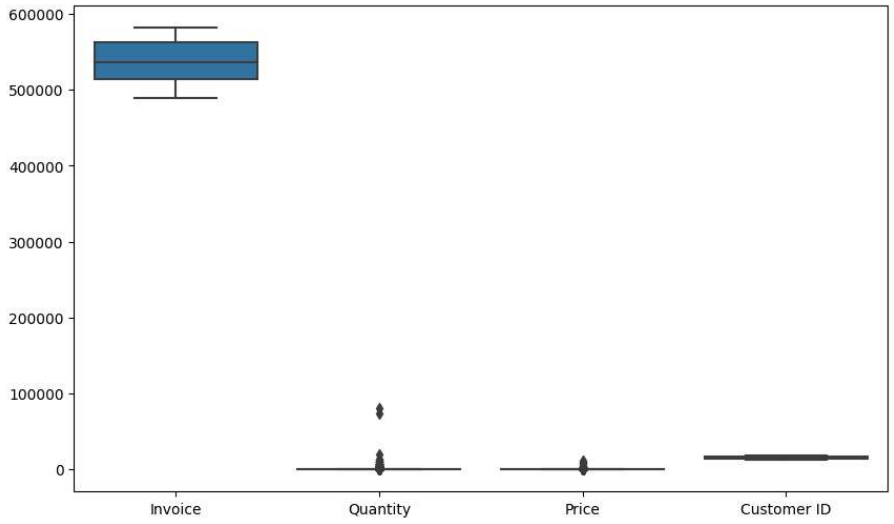
Now, our data is free of all missing values, duplicate values, and negative values

SINGLE SIZE

Check for Outliers

TRINKET BOX

```
plt.figure(figsize=(10, 6))
sns.boxplot(data = df)
plt.show()
```



From the above boxplot and based on df.describe, there exists outliers in the Quantity and Price column

```
# We'll remove only the extreme outliers as we have lost enough data already
Q1 = df[['Quantity', 'Price']].quantile(0.01)
Q3 = df[['Quantity', 'Price']].quantile(0.99)
IQR = Q3 - Q1

df_outliers = df[((df[['Quantity', 'Price']] < (Q1 - 1.5 * IQR)) | (df[['Quantity', 'Price']] > (Q3 + 1.5 * IQR))).any(axis=1)]


df_outliers
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Countr
126	489444	POST	POSTAGE	1	2009-12-01 09:55:00	141.00	12636.0	US/
173	489447	POST	POSTAGE	1	2009-12-01 10:10:00	130.00	12362.0	Belgiun
217	489460	84598	BOYS ALPHABET	576	2009-12-01	0.21	16167.0	Unite

```
df = df.drop(df_outliers.index)
df
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Count
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	Unit Kingdc
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	Unit Kingdc
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	Unit Kingdc
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	Unit Kingdc
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	Unit Kingdc
...	...	...	...	...	...	...	...	...

```
df.describe()
```

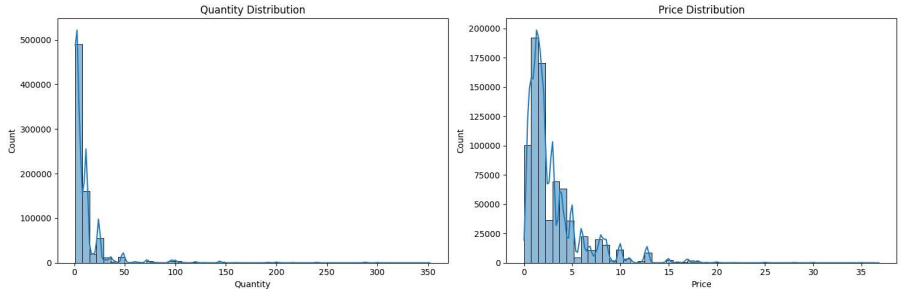
	Quantity	Price	Customer ID
count	774156.000000	774156.000000	774156.000000
mean	11.343552	2.899091	15320.665366
std	22.896442	2.926629	1694.962047
min	1.000000	0.000000	12346.000000
25%	2.000000	1.250000	13975.000000
50%	6.000000	1.950000	15249.000000
75%	12.000000	3.750000	16794.000000
max	352.000000	36.800000	18287.000000

```
#Final shape of the dataset
print(f' Final shape of the dataset : {df.shape}')
```

```
 Final shape of the dataset : (774156, 8)
```

EDA to answer some Key questions

```
figs, axes = plt.subplots(1, 2, figsize=(15, 5))
sns.histplot(df['Quantity'], bins = 50, kde = True, ax = axes[0])
axes[0].set_title('Quantity Distribution')

sns.histplot(df['Price'], bins = 50, kde = True, ax = axes[1])
axes[1].set_title('Price Distribution')

plt.tight_layout()
plt.show()
```

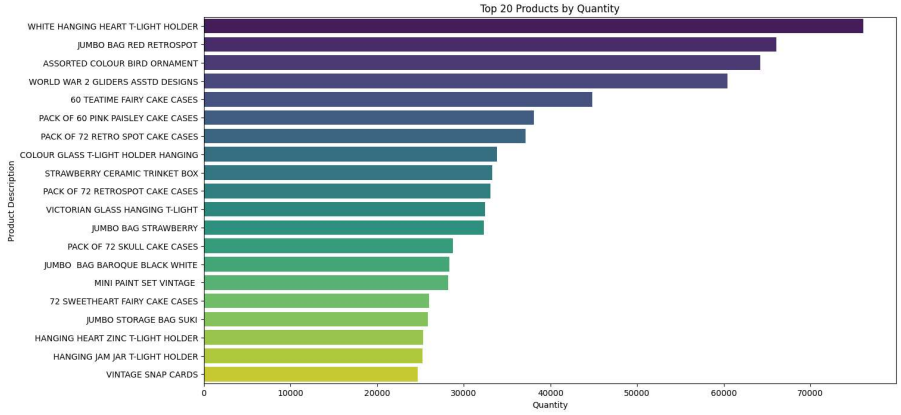


- We can see that the consumers buy things in lower quantity more (especially in the range of 1 - 10)
- Also the consumers tend to buy more things in the lower price range

1. Top 20 products based on Quantity sold

```
product_quantity = df.groupby(df['Description'])['Quantity'].sum().reset_index()
sorted_product_quantity = product_quantity.sort_values(by = 'Quantity', ascending = False)
# Let's get the top 20 products that are brought by consumers
top_20_products = sorted_product_quantity.head(20)
```

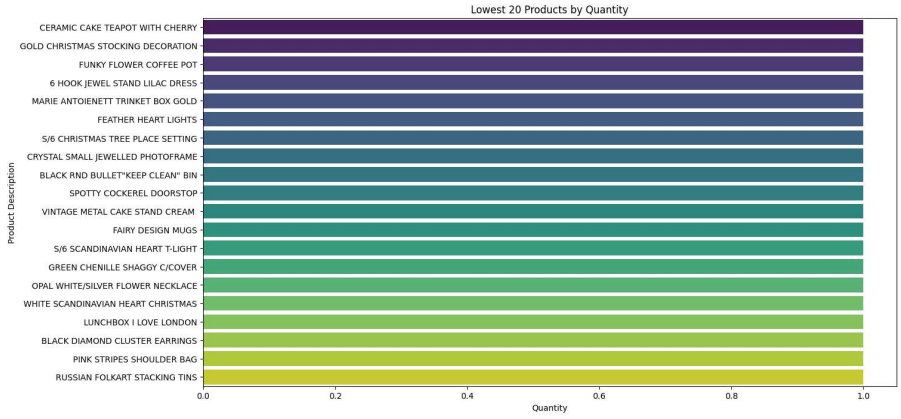
```
# Plot the bar plot
plt.figure(figsize=(15, 8))
sns.barplot(x='Quantity', y='Description', data=top_20_products, palette='viridis')
plt.title(f'Top 20 Products by Quantity')
plt.xlabel('Quantity')
plt.ylabel('Product Description')
plt.show()
```



2. Least sold products

```
product_quantity = df.groupby(df['Description'])['Quantity'].sum().reset_index()
sorted_product_quantity = product_quantity.sort_values(by = 'Quantity', ascending = True)
# Let's get the top 20 products that are brought by consumers
low_20_products = sorted_product_quantity.head(20)
```

```
# Plot the bar plot
plt.figure(figsize=(15, 8))
sns.barplot(x='Quantity', y='Description', data=low_20_products, palette='viridis')
plt.title(f'Lowest 20 Products by Quantity')
plt.xlabel('Quantity')
plt.ylabel('Product Description')
plt.show()
```




```
low_20_products.Description
```

```
910        CERAMIC CAKE TEAPOT WITH CHERRY
1991    GOLD CHRISTMAS STOCKING DECORATION
1861                FUNKY FLOWER COFFEE POT
145          6 HOOK JEWEL STAND LILAC DRESS
2670     MARIE ANTOIENETT TRINKET BOX GOLD
1614                  FEATHER HEART LIGHTS
3981       S/6 CHRISTMAS TREE PLACE SETTING
1191     CRYSTAL SMALL JEWELLED PHOTOFRAME
510       BLACK RND BULLET"KEEP CLEAN" BIN
4553              SPOTTY COCKEREL DOORSTOP
4895       VINTAGE METAL CAKE STAND CREAM
1597                      FAIRY DESIGN MUGS
3988        S/6 SCANDINAVIAN HEART T-LIGHT
2029         GREEN CHENILLE SHAGGY C/COVER
2950     OPAL WHITE/SILVER FLOWER NECKLACE
5051    WHITE SCANDINAVIAN HEART CHRISTMAS
2623                LUNCHBOX I LOVE LONDON
477         BLACK DIAMOND CLUSTER EARRINGS
3437              PINK STRIPES SHOULDER BAG
3922         RUSSIAN FOLKART STACKING TINS
Name: Description, dtype: object
```

```
df[df['Description'] == 'WHITE SCANDINAVIAN HEART CHRISTMAS']
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Count

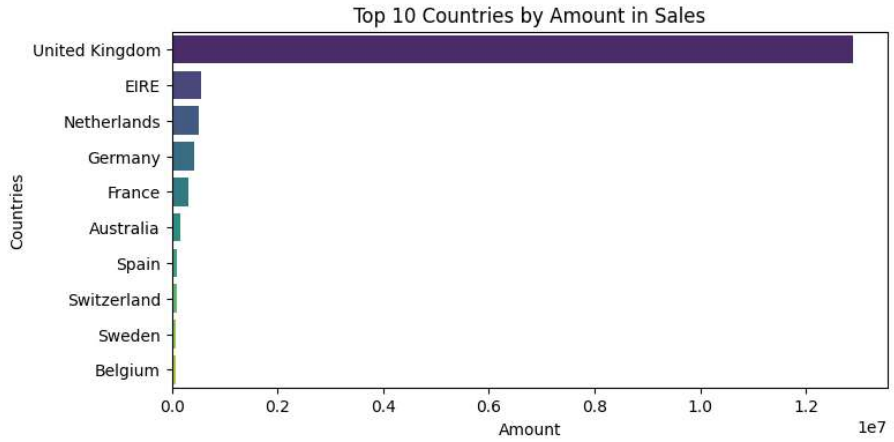
3. Top 10 countries based on total transactions and amount spent in sales

```
df['Amount'] = df['Quantity'] * df['Price']
```

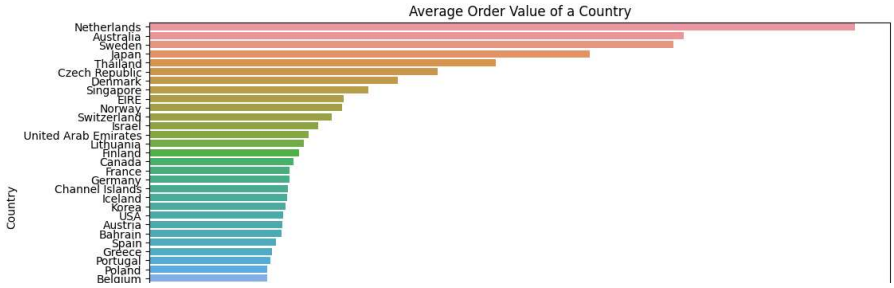
```
country_sales = df.groupby('Country')['Amount'].sum().reset_index()
country_sales = country_sales.sort_values(by = 'Amount', ascending=False)
top_10_country_sales = country_sales.head(10)
top_10_country_sales
```

	Country	Amount
38	United Kingdom	1.289280e+07
10	EIRE	5.600062e+05
24	Netherlands	5.037647e+05
14	Germany	4.173126e+05
13	France	3.176805e+05
0	Australia	1.581344e+05
32	Spain	9.362368e+04
34	Switzerland	9.326504e+04
33	Sweden	7.423948e+04
3	Belgium	6.320734e+04

```
# Plot the bar plot
plt.figure(figsize=(8, 4))
sns.barplot(x='Amount', y='Country', data=top_10_country_sales, palette='viridis')
plt.title(f'Top 10 Countries by Amount in Sales')
plt.xlabel('Amount')
plt.ylabel('Countries')
plt.show()
```

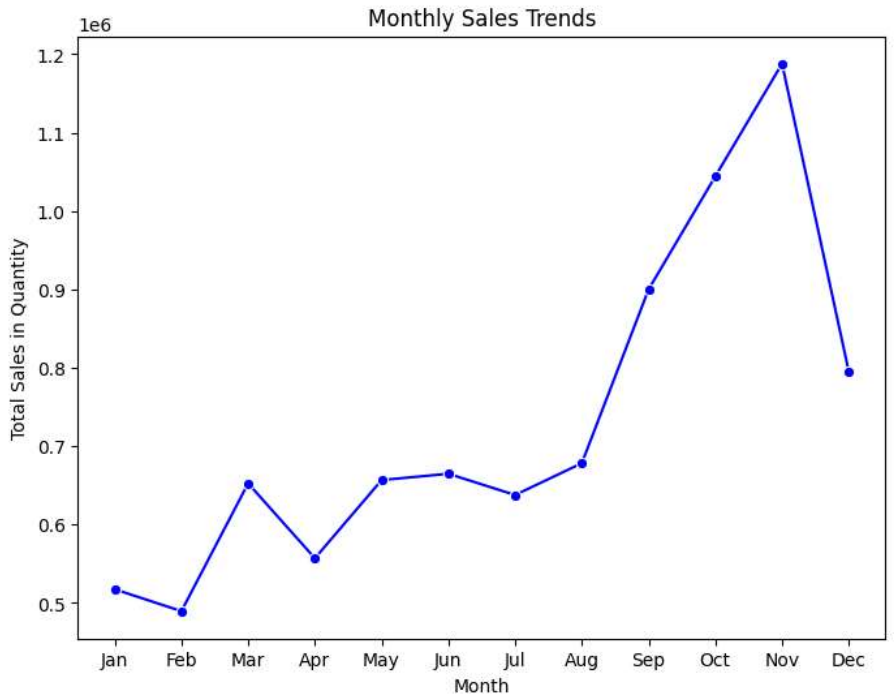



```
country_mean = df.groupby('Country')['Quantity'].mean().reset_index()
sorted_country_mean = country_mean.sort_values(by = 'Quantity', ascending = False)
plt.figure(figsize=(12, 6))
sns.barplot(x='Quantity', y='Country', data=sorted_country_mean)
plt.title('Average Order Value of a Country')
plt.xlabel('Average Order Value')
plt.ylabel('Country')
plt.show()
```



4. Top 10 selling products based on the month

```
df['Month'] = df['InvoiceDate'].dt.month
df['Day'] = df['InvoiceDate'].dt.day
df['Hour'] = df['InvoiceDate'].dt.hour


sales_by_month = df.groupby('Month')['Quantity'].sum().reset_index()


# Plot the monthly sales trends
plt.figure(figsize=(8, 6))
sns.lineplot(x='Month', y='Quantity', data=sales_by_month, marker='o', color='blue')
plt.title('Monthly Sales Trends')
plt.xlabel('Month')
plt.ylabel('Total Sales in Quantity')
plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.show()
```



We can see that there is huge rise in sale of products around second half of the year, especially during October and November, which is also holiday season. Let's further analyze which products are sold more in each month as this would help in planning the inventory

5. Top products on monthly-basis as this would help us maintain the inventory

```
# Group by month and product description, calculate total sales, and rank products
monthly_top_products = (
    df.groupby(['Month', 'Description'])
    .agg({'Quantity': 'sum'})
    .reset_index())
```

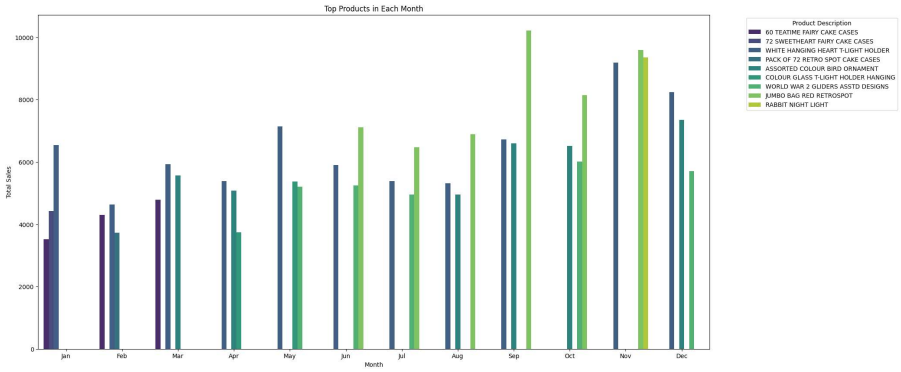
```
# Rank products within each month based on total sales
monthly_top_products['Rank'] = monthly_top_products.groupby('Month')['Quantity'].rank(ascending=False, method='dense')


# Filter to get the top N products in each month (adjust N as needed)
top_n = 3
monthly_top_n_products = monthly_top_products[monthly_top_products['Rank'] <= top_n]

# Display the top products in each month
print(monthly_top_n_products)
```

```
        Month                         Description  Quantity  Rank
109         1          60 TEATIME FAIRY CAKE CASES      3523   3.0
111         1       72 SWEETHEART FAIRY CAKE CASES      4424   2.0
3132        1   WHITE HANGING HEART T-LIGHT HOLDER      6545   1.0
3373        2          60 TEATIME FAIRY CAKE CASES      4302   2.0
5174        2     PACK OF 72 RETRO SPOT CAKE CASES      3734   3.0
6417        2   WHITE HANGING HEART T-LIGHT HOLDER      4641   1.0
6655        3          60 TEATIME FAIRY CAKE CASES      4790   3.0
6739        3       ASSORTED COLOUR BIRD ORNAMENT      5571   2.0
9854        3   WHITE HANGING HEART T-LIGHT HOLDER      5934   1.0
10164       4       ASSORTED COLOUR BIRD ORNAMENT      5073   2.0
10690       4  COLOUR GLASS T-LIGHT HOLDER HANGING      3748   3.0
13084       4   WHITE HANGING HEART T-LIGHT HOLDER      5393   1.0
13944       5  COLOUR GLASS T-LIGHT HOLDER HANGING      5370   2.0
16370       5   WHITE HANGING HEART T-LIGHT HOLDER      7143   1.0
16444       5    WORLD WAR 2 GLIDERS ASSTD DESIGNS      5200   3.0
18078       6              JUMBO BAG RED RETROSPOT      7110   1.0
19851       6   WHITE HANGING HEART T-LIGHT HOLDER      5908   2.0
19927       6    WORLD WAR 2 GLIDERS ASSTD DESIGNS      5244   3.0
21509       7              JUMBO BAG RED RETROSPOT      6471   1.0
23229       7   WHITE HANGING HEART T-LIGHT HOLDER      5390   2.0
23307       7    WORLD WAR 2 GLIDERS ASSTD DESIGNS      4954   3.0
23564       8       ASSORTED COLOUR BIRD ORNAMENT      4956   3.0
24940       8              JUMBO BAG RED RETROSPOT      6890   1.0
26667       8   WHITE HANGING HEART T-LIGHT HOLDER      5314   2.0
27024       9       ASSORTED COLOUR BIRD ORNAMENT      6593   3.0
28428       9              JUMBO BAG RED RETROSPOT     10214   1.0
30198       9   WHITE HANGING HEART T-LIGHT HOLDER      6725   2.0
30562      10       ASSORTED COLOUR BIRD ORNAMENT      6521   2.0
31992      10              JUMBO BAG RED RETROSPOT      8141   1.0
33948      10    WORLD WAR 2 GLIDERS ASSTD DESIGNS      6014   3.0
35675      11              JUMBO BAG RED RETROSPOT      9599   1.0
36515      11                   RABBIT NIGHT LIGHT      9352   2.0
37536      11   WHITE HANGING HEART T-LIGHT HOLDER      9183   3.0
37948      12       ASSORTED COLOUR BIRD ORNAMENT      7354   2.0
41844      12   WHITE HANGING HEART T-LIGHT HOLDER      8244   1.0
41946      12    WORLD WAR 2 GLIDERS ASSTD DESIGNS      5701   3.0
```

```
monthly_top_n_products['Description'].value_counts()
```

```
WHITE HANGING HEART T-LIGHT HOLDER     11
ASSORTED COLOUR BIRD ORNAMENT           6
JUMBO BAG RED RETROSPOT                 6
WORLD WAR 2 GLIDERS ASSTD DESIGNS       5
60 TEATIME FAIRY CAKE CASES             3
COLOUR GLASS T-LIGHT HOLDER HANGING     2
72 SWEETHEART FAIRY CAKE CASES          1
PACK OF 72 RETRO SPOT CAKE CASES        1
RABBIT NIGHT LIGHT                      1
Name: Description, dtype: int64
```

```
plt.figure(figsize=(20,10))

# Create a grouped bar plot
sns.barplot(x='Month', y='Quantity', hue='Description', data=monthly_top_n_products, palette='viridis')

# Add labels and title
plt.title('Top Products in Each Month')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.xticks(range(0, 12), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.legend(title='Product Description', bbox_to_anchor=(1.05, 1), loc='best')
plt.show()
```



Insights -

- 60 TEATIME FAIRY CAKE CASES - is being sold more mainly in the first 3 months of the year (Spring)
- WHITE HANGING HEART T-LIGHT HOLDER - is one of their best selling products, as it is bought in huge volumes all year around
- ASSORTED COLOUR BIRD ORNAMENT - is sold mainly in the second half of the year
- JUMBO BAG RED RETROSPOT - is sold abundantly in the months from June to November, peaking in the month of September

6. Top products sold based on the Country the consumers buy from

```
# Group by country and product description, calculate total sales, and rank products
countrywise_top_products = (
    df.groupby(['Country', 'Description'])
    .agg({'Quantity': 'sum'})
    .reset_index())
```

```
countrywise_top_products
```

	Country	Description	Quantity
0	Australia	DOLLY GIRL BEAKER	200
1	Australia	I LOVE LONDON MINI BACKPACK	4
2	Australia	10 COLOUR SPACEBOY PEN	48
3	Australia	12 PENCILS SMALL TUBE RED SPOTTY	24
4	Australia	12 PENCILS TALL TUBE POSY	252
...	...	...	...
29210	West Indies	VINTAGE BEAD PINK SCARF	3
29211	West Indies	WHITE AND BLUE CERAMIC OIL BURNER	6
29212	West Indies	WOODLAND PARTY BAG + STICKER SET	1
29213	West Indies	WOVEN BERRIES CUSHION COVER	2
29214	West Indies	WOVEN FROST CUSHION COVER	2

29215 rows × 3 columns

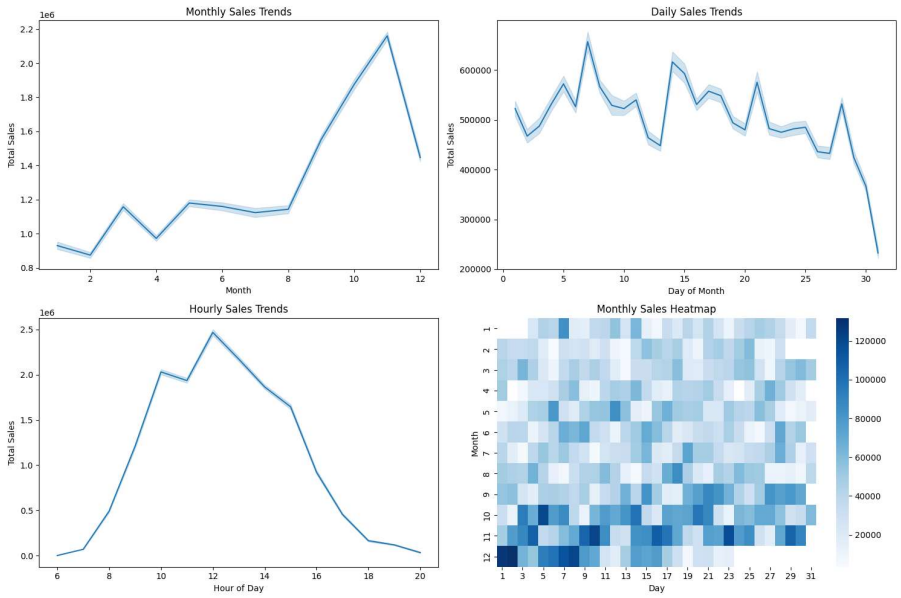
```
# Rank products within each country based on total sales
countrywise_top_products['Rank'] = countrywise_top_products.groupby('Country')['Quantity'].rank(ascending=False, method='dense')
```

```
# Filter to get the top N products in each country (adjust N as needed)
top_n = 1
countrywise_top_products = countrywise_top_products[countrywise_top_products['Rank'] <= top_n]

# Display the top products in each month
countrywise_top_products
```

	Country	Description	Quantity	Rank
176	Australia	DOLLY GIRL LUNCH BOX	1316	1.0
1293	Austria	SET 12 KIDS COLOUR CHALK STICKS	288	1.0
1428	Bahrain	ICE CREAM SUNDAE LIP GLOSS	96	1.0
1726	Belgium	DOLLY GIRL LUNCH BOX	572	1.0
2554	Brazil	DOLLY GIRL LUNCH BOX	25	1.0
2557	Brazil	DRAGONS BLOOD INCENSE	25	1.0
2832	Canada	WORLD WAR 2 GLIDERS ASSTD DESIGNS	288	1.0
2868	Channel Islands	AFGHAN SLIPPER SOCK PAIR	600	1.0
3986	Cyprus	HEART DECORATION PAINTED ZINC	384	1.0
4510	Czech Republic	WOODEN STAR CHRISTMAS SCANDINAVIAN	72	1.0
4511	Czech Republic	WOODEN TREE CHRISTMAS SCANDINAVIAN	72	1.0
4666	Denmark	GUMBALL COAT RACK	354	1.0
5030	EIRE	60 TEATIME FAIRY CAKE CASES	4020	1.0
7991	European Community	RED ROCKING HORSE HAND PAINTED	24	1.0
7992	European Community	ROCKING HORSE GREEN CHRISTMAS	24	1.0
7993	European Community	ROCKING HORSE RED CHRISTMAS	24	1.0
8005	European Community	SET OF 60 PANTRY DESIGN CAKE CASES	24	1.0
8006	European Community	SET OF 60 VINTAGE LEAF CAKE CASES	24	1.0
8011	European Community	WHITE ROCKING HORSE HAND PAINTED	24	1.0
8141	Finland	CHILDRENS CUTLERY POLKADOT PINK	480	1.0
9683	France	MINI PAINT SET VINTAGE	3168	1.0
12542	Germany	ROUND SNACK BOXES SET OF4 WOODLAND	2668	1.0
13704	Greece	WHITE HANGING HEART T-LIGHT HOLDER	160	1.0
13773	Iceland	ICE CREAM SUNDAE LIP GLOSS	240	1.0
14123	Israel	WOODLAND CHARLOTTE BAG	120	1.0
14384	Italy	FEATHER PEN,HOT PINK	432	1.0
15106	Japan	PACK OF 12 TRADITIONAL CRAYONS	481	1.0
15266	Korea	CACTI T-LIGHT CANDLES	48	1.0
15278	Korea	HEART T-LIGHT HOLDER	48	1.0

7. Time-Series Analysis - Visualize monthly, daily and hourly sales trend

```
# Set up subplots for time-based analysis
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))

# Monthly sales trends
sns.lineplot(x='Month', y='Amount', data=df, estimator='sum', ax=axes[0, 0])
axes[0, 0].set_title('Monthly Sales Trends')
axes[0, 0].set_xlabel('Month')
axes[0, 0].set_ylabel('Total Sales')

# Daily sales trends
sns.lineplot(x='Day', y='Amount', data=df, estimator='sum', ax=axes[0, 1])
axes[0, 1].set_title('Daily Sales Trends')
axes[0, 1].set_xlabel('Day of Month')
axes[0, 1].set_ylabel('Total Sales')

# Hourly sales trends
sns.lineplot(x='Hour', y='Amount', data=df, estimator='sum', ax=axes[1, 0])
axes[1, 0].set_title('Hourly Sales Trends')
axes[1, 0].set_xlabel('Hour of Day')
axes[1, 0].set_ylabel('Total Sales')

# Monthly sales heatmap
monthly_sales_heatmap = df.groupby(['Month', 'Day'])['Amount'].sum().unstack()
sns.heatmap(monthly_sales_heatmap, cmap='Blues', ax=axes[1, 1])
axes[1, 1].set_title('Monthly Sales Heatmap')

plt.tight_layout()
plt.show()
```



- We can observe that there is a increased sale of goods in the last 5 months (more of the later half of the year)
- The daily sales trend kind of stays constant, but declines towards the end of a month
- The hourly sale sees a spike in the middle of the day, and decreases in the very early and late hours
- The heatmap shows that the sale of goods is more during the last 3 months in contrast to the initial months of the year

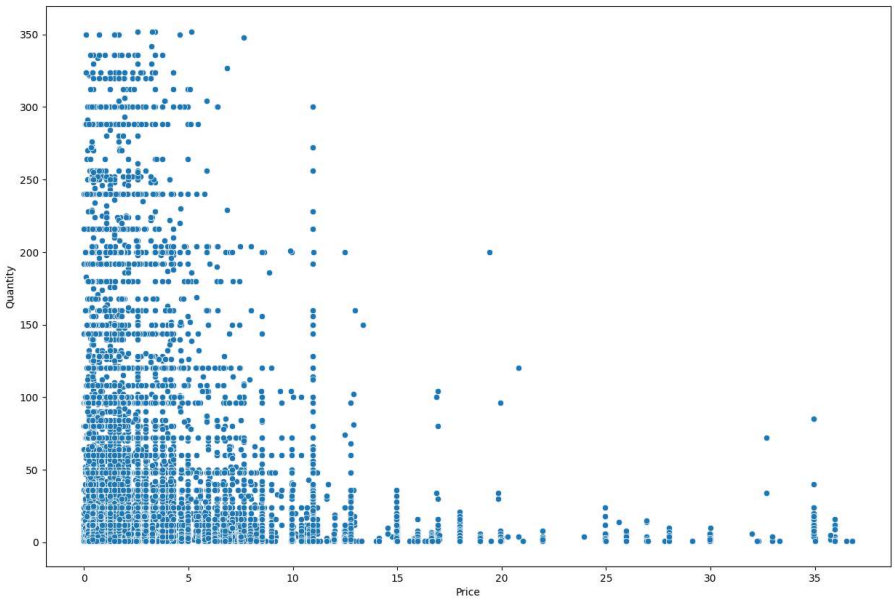
8. Correlation Analysis

```
# Calculate the Pearson correlation coefficient
correlation_coefficient, _ = pearsonr(df['Quantity'], df['Price'])

print(f"Pearson Correlation Coefficient: {correlation_coefficient}")
# Interpretation
if correlation_coefficient > 0:
    print("There is a positive correlation between Quantity and Price.")
elif correlation_coefficient < 0:
    print("There is a negative correlation between Quantity and Price.")
else:
    print("There is no correlation between Quantity and Price.")
```

```
Pearson Correlation Coefficient: -0.19047973245926025
There is a negative correlation between Quantity and Price.
```

This makes sense as well, because as the price of a product increases consumers tend to buy lower quantities of the product.

```
plt.figure(figsize = (15,10))
sns.scatterplot(data = df, x = df['Price'], y = df['Quantity'])
plt.show()
```

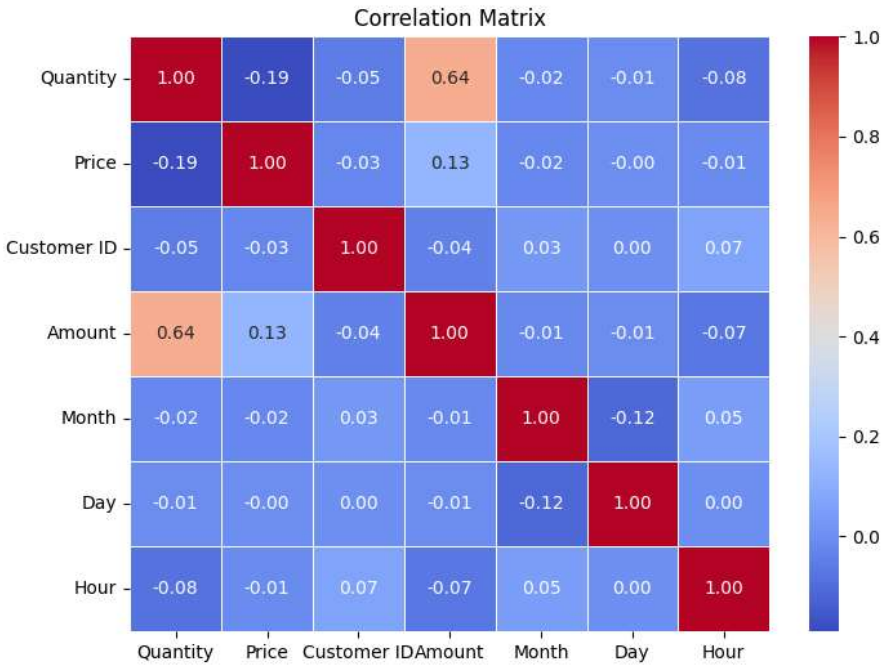



```
print('Correlation matrix :')
df.corr()
```

```
Correlation matrix :
```

	Quantity	Price	Customer ID	Amount	Month	Day	Hour
Quantity	1.000000	-0.190480	-0.054536	0.640127	-0.019210	-0.005967	-0.078898
Price	-0.190480	1.000000	-0.026243	0.133775	-0.020899	-0.001568	-0.011526
Customer ID	-0.054536	-0.026243	1.000000	-0.037775	0.028848	0.000497	0.065437

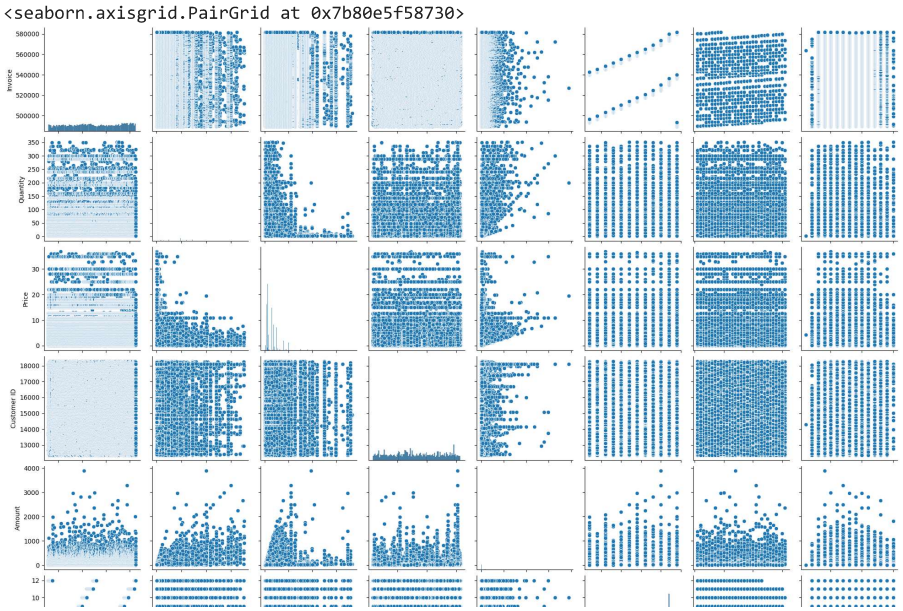
```
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



There doesn't seem to be any correlation majorly between any variables. There is significant correaltion between Quantity and Amount because Amount was derived using Quantity and Price.

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7b80e5f58730>
```



9. Market Basket Analysis

9a. Start with Apriori Algorithm to get the association mining rules

The Apriori algorithm is commonly used for association rule mining. Before applying the Apriori algorithm, we need to format your data in a way that each row represents a transaction and each column represents an item

```
# Drop duplicate rows to ensure that the same product is not counted multiple times in a single transaction
df_unique = df[['Customer ID', 'Description']].drop_duplicates()
# Create a one-hot encoded matrix
basket = df_unique.pivot_table(index='Customer ID', columns='Description', aggfunc='size', fill_value=0)
basket
```

Description	10 COLOUR SPACEBOY PEN	11 PC CERAMIC TEA SET POLKADOT	12 ASS ZINC CHRISTMAS DECORATIONS	12 COLOURED PARTY BALLOONS	12 DAISY PEGS IN WOOD BOX	12 EGG HOUSE PAINTED WOOD	12 HANGING EGGS HAND PAINTED	12 IVOR ROSE PE PLAC SETTING
Customer ID								
12346.0	0	0	0	0	0	0	0	
12347.0	1	0	0	0	0	0	0	
12348.0	0	0	0	0	0	0	0	
12349.0	0	0	0	0	0	0	0	
12350.0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	.
18283.0	1	0	0	0	0	0	0	
18284.0	1	0	0	0	0	0	0	
18285.0	0	0	0	0	0	0	0	
18286.0	1	0	0	0	0	0	0	
18287.0	0	0	0	0	0	0	0	

5833 rows × 5220 columns

```
# Convert counts to binary values (0 or 1)
basket[basket > 0] = 1
basket.head()
```

Description	10 COLOUR SPACEBOY PEN	11 PC CERAMIC TEA SET POLKADOT	12 ASS ZINC CHRISTMAS DECORATIONS	12 COLOURED PARTY BALLOONS	12 DAISY PEGS IN WOOD BOX	12 EGG HOUSE PAINTED WOOD	12 HANGING EGGS HAND PAINTED	12 IVOR ROSE PE PLAC SETTING
Customer ID								
12346.0	0	0	0	0	0	0	0	
12347.0	1	0	0	0	0	0	0	
12348.0	0	0	0	0	0	0	0	
12349.0	0	0	0	0	0	0	0	
12350.0	0	0	0	0	0	0	0	

5 rows × 5220 columns

```
basket.describe()
```

Description	10 COLOUR SPACEBOY PEN	11 PC CERAMIC TEA SET POLKADOT	12 ASS ZINC CHRISTMAS DECORATIONS	12 COLOURED PARTY BALLOONS	12 DAISY PEGS IN WOOD BOX	12 EGG HOUSE PAINTED WOOD
count	5833.000000	5833.000000	5833.000000	5833.000000	5833.000000	5833.000000
mean	0.056575	0.000171	0.006686	0.029659	0.022630	0.015772
std	0.231048	0.013093	0.081502	0.169659	0.148733	0.124604
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 5220 columns

```
# Record start time
start_time = time.time()
# Apply the Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(basket, min_support=0.05, use_colnames=True)
end_time = time.time()
# Calculate execution time
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")
# Display the frequent itemsets
print("Frequent Itemsets:")
frequent_itemsets
```

```
Execution Time: 7.038375616073608 seconds
Frequent Itemsets:
```

	support	itemsets

```
# Generate association rules
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)
# Display the association rules
print("\nAssociation Rules:")
rules
```

```
Association Rules:
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(3 STRIPEY MICE FELTCRAFT)	(FELTCRAFT 6 FLOWER FRIENDS)	0.086576	0.116921	0.050231	0.580198	4.962309
1	(FELTCRAFT 6 FLOWER FRIENDS)	(3 STRIPEY MICE FELTCRAFT)	0.116921	0.086576	0.050231	0.429619	4.962309
2	(60 TEATIME FAIRY CAKE CASES)	(72 SWEETHEART FAIRY CAKE CASES)	0.140237	0.104920	0.065832	0.469438	4.474232
3	(72 SWEETHEART FAIRY CAKE CASES)	(60 TEATIME FAIRY CAKE CASES)	0.104920	0.140237	0.065832	0.627451	4.474232
4	(BAKING SET 9 PIECE RETROSPOT)	(60 TEATIME FAIRY CAKE CASES)	0.194411	0.140237	0.050231	0.258377	1.842440
...	...	...	...	...	...	...	...
371	(WOODEN FRAME ANTIQUE WHITE, WHITE HANGING HEA...	(WOODEN PICTURE FRAME WHITE FINISH)	0.064289	0.102863	0.050746	0.789333	7.673636
372	(WOODEN PICTURE FRAME WHITE FINISH,	(WOODEN FRAME ANTIQUE WHITE)	0.064118	0.107835	0.050746	0.791444	7.339415

9b. Use FP Growth algorithm

- It is memory-efficient
- Better for sparse dataset
- Faster than apriori for large datasets

```
frequent_itemsets_fp = fpgrowth(basket, min_support=0.05, use_colnames=True)
# Display the frequent itemsets with FP-growth
print("Frequent Itemsets with FP-growth:")
frequent_itemsets_fp
```

```
Frequent Itemsets with FP-growth:
```

	support	itemsets
0	0.079719	(DOORMAT UNION FLAG)
1	0.066347	(DOORMAT NEW ENGLAND)
2	0.060689	(DOORMAT SPOTTY HOME SWEET HOME)
3	0.059318	(EDWARDIAN PARASOL NATURAL)
4	0.054860	(EDWARDIAN PARASOL BLACK)
...	...	...
553	0.050060	(HEART OF WICKER SMALL, ZINC METAL HEART DECOR...

```
rules = association_rules(frequent_itemsets_fp,metric = "confidence",min_threshold = 0.05)
rules.head(100)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(REGENCY CAKESTAND 3 TIER)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.225270	0.255100	0.083490	0.370624	1.452856
1	(WHITE HANGING HEART T-LIGHT HOLDER)	(REGENCY CAKESTAND 3 TIER)	0.255100	0.225270	0.083490	0.327285	1.452856
2	(REGENCY CAKESTAND 3 TIER)	(60 TEATIME FAIRY CAKE CASES)	0.225270	0.140237	0.055032	0.244292	1.742001
3	(60 TEATIME FAIRY CAKE CASES)	(REGENCY CAKESTAND 3 TIER)	0.140237	0.225270	0.055032	0.392421	1.742001
4	(BAKING SET 9 PIECE RETROSPOT)	(60 TEATIME FAIRY CAKE CASES)	0.194411	0.140237	0.050231	0.258377	1.842440
...	...	...	...	...	...	...	...
95	(NATURAL SLATE HEART CHALKBOARD)	(HEART OF WICKER SMALL)	0.149666	0.151723	0.059832	0.399771	2.634874
	(RECIPE BOX PANTRY	(WHITE HANGING					

```
rules[(rules['lift'] >= 3) & (rules['confidence'] >= 0.5)]
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	
8	(VINTAGE SNAP CARDS)	(VINTAGE HEADS AND TAILS CARD GAME)	0.135608	0.109721	0.074919	0.552465	5.03
9	(VINTAGE HEADS AND TAILS CARD GAME)	(VINTAGE SNAP CARDS)	0.109721	0.135608	0.074919	0.682813	5.03
11	(72 SWEETHEART FAIRY CAKE CASES)	(60 TEATIME FAIRY CAKE CASES)	0.104920	0.140237	0.065832	0.627451	4.47

Let's try it out with a input

```
unique_values = df_unique['Description'].unique()
print(f"There are totally {len(unique_values)} items in our store, what do you want to pick :")

user_input = input("Enter the name of the item you want to pick: ")
user_input = user_input.strip()

filtered = rules[rules["antecedents"] == frozenset({user_input})]["consequents"]
filtered.size
if filtered.size == 0:
    print("There are no recommended items for this product, choose a different item")
else:
    print("Here are your recommended items for  "+user_input+"--")
    print(filtered)
    items = []
    for data in filtered:
        items.extend(data)
```

```
 There are totally 5220 items in our store, what do you want to pick :
 Enter the name of the item you want to pick: ROSES REGENCY TEACUP AND SAUCER
 Here are your recommended items for  ROSES REGENCY TEACUP AND SAUCER--
 25                          (REGENCY CAKESTAND 3 TIER)
 34                   (GREEN REGENCY TEACUP AND SAUCER)
 42    (REGENCY CAKESTAND 3 TIER, GREEN REGENCY TEACU...
 48                    (PINK REGENCY TEACUP AND SAUCER)
 55    (PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY...
 68    (REGENCY CAKESTAND 3 TIER, PINK REGENCY TEACUP...
 Name: consequents, dtype: object
```

```
item=input("What is the item you want to pick")
filtered = rules[rules["antecedents"] == frozenset({item})]["consequents"]
filtered.size
if filtered.size == 0:
    print("There are no recommended items for this product, choose a different item")
else:
    print("Here are your recommended items for  "+item+"--")
    print(filtered)
    items = []
    for data in filtered:
        items.extend(data)
```

```
 What is the item you want to pickROSES REGENCY TEACUP AND SAUCER
 Here are your recommended items for  ROSES REGENCY TEACUP AND SAUCER--
 25                          (REGENCY CAKESTAND 3 TIER)
 34                   (GREEN REGENCY TEACUP AND SAUCER)
 42    (REGENCY CAKESTAND 3 TIER, GREEN REGENCY TEACU...
 48                    (PINK REGENCY TEACUP AND SAUCER)
 55    (PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY...
 68    (REGENCY CAKESTAND 3 TIER, PINK REGENCY TEACUP...
 Name: consequents, dtype: object
```

9c. RFM analysis to calculate Recency, Frequency, and Monetary and use these further for Customer Segementation

Customer segmentation is the practice of grouping customers based on common characteristics. These customer segments are beneficial in marketing campaigns, in identifying potentially profitable customers, and in developing customer loyalty. A company might segment customers according to a wide range of factors, including: demographics (age, gender, location etc), behaviour (previous orders, responses to messaging), psychographics (values, interests, lifestyles) etc.

RFM (Recency-Frequency-Monetary) analysis is a simple technique for behaviour based customer segmentation. It groups customers based on their transaction history – how recently, how often and how much did they buy. It is a handy method to find the best customers, understand their behavior and then run targeted marketing campaigns to increase sales, satisfaction and customer lifetime value.

```
# Calculating RFM metrics
current_date = max(df['InvoiceDate'])  # Current date

# Calculate Recency, Frequency, and Monetary metrics for each customer
rfm_df = df.groupby('Customer ID').agg({
    'InvoiceDate': lambda x: (current_date - x.max()).days,  # Recency
    'Invoice': 'nunique',  # Frequency
    'Amount': 'sum'  # Monetary value
})

# Rename columns
rfm_df.rename(columns={
    'InvoiceDate': 'Recency',
    'Invoice': 'Frequency',
    'Amount': 'Monetary'
}, inplace=True)

# Print the first few rows of the RFM DataFrame
rfm_df = rfm_df.sort_values(by = 'Monetary',ascending = False)
rfm_df.head()
```

Customer ID	Recency	Frequency	Monetary
14646.0	1	146	479320.00
18102.0	0	134	429970.66
14156.0	9	146	279145.20
14911.0	0	376	269686.47
13694.0	3	134	182438.31

We are sorting them in descening order of Monetary value as the Monetary aspect is a crucial component of RFM analysis as it helps segment customers based on their purchasing behavior and contribution to revenue. Customers who spend more tend to be more valuable to a business, and identifying these high-value customers can assist in targeted marketing strategies, loyalty programs, and personalized offerings to maximize revenue and customer retention.

Customer Segmentation based on RFM values

```
def create_segments(row):
    if row['Recency'] <= 30 and row['Frequency'] >= 100 and row['Monetary'] >= 50000:
        return 'High-Value Customer'
    elif row['Recency'] > 90 and row['Frequency'] < 3:
        return 'Churn Risk'
    else:
        return 'Regular Customer'

rfm_df['Segment'] = rfm_df.apply(create_segments, axis=1)

print("RFM Analysis with Segments:")
rfm_df.head(20)
```

RFM Analysis with Segments:

Customer ID	Recency	Frequency	Monetary	Segment
14646.0	1	146	479320.00	High-Value Customer
18102.0	0	134	429970.66	High-Value Customer
14156.0	9	146	279145.20	High-Value Customer
14911.0	0	376	269686.47	High-Value Customer
13694.0	3	134	182438.31	High-Value Customer
17511.0	2	60	171344.92	Regular Customer
12415.0	23	26	134463.45	Regular Customer
16684.0	3	49	119972.57	Regular Customer
15061.0	3	123	117693.94	High-Value Customer
15311.0	0	208	114369.88	High-Value Customer
13089.0	2	202	113115.06	High-Value Customer
17450.0	7	44	90449.63	Regular Customer
14298.0	7	79	89680.15	Regular Customer
16029.0	38	103	78038.63	Regular Customer
13798.0	0	110	75260.39	High-Value Customer
17841.0	1	211	67019.32	High-Value Customer

This was one way to do this. Now we will be trying another method where we convert these values to the range from 1 to 5 across all the metrics for easier segmentation and for having a consistent range across all the metrics

Customers with the highest RFM score is considered the "best" customer. They have made recent purchases (high R), engage frequently (high F), and contribute significantly in terms of monetary value (high M). These customers are often considered the most valuable and important for the business.

```
r_labels, f_labels, m_labels = range(1, 6), range(1,6), range(1,6)
# Define custom bins (adjust according to your data distribution)
custom_bins = [0, 5, 10, 20, 30, 1000]

# Use pd.cut() with custom bins
rfm_df['r_score'] = pd.qcut(rfm_df['Recency'], q=5, labels=r_labels).astype(int)
rfm_df['f_score'] = pd.cut(rfm_df['Frequency'], bins=custom_bins, labels=f_labels).astype(int)
rfm_df['m_score'] = pd.qcut(rfm_df['Monetary'], q=5, labels=m_labels).astype(int)

rfm_df['rfm_sum'] = rfm_df['r_score'] + rfm_df['m_score'] + rfm_df['f_score']
rfm_df = rfm_df.sort_values(by = "rfm_sum",ascending = False)
rfm_df
```

Customer ID	Recency	Frequency	Monetary	Segment	r_score	f_score	m_score	rfm_sum
17448.0	496	46	14498.47	Regular Customer	5	5	5	15
12835.0	427	41	5982.53	Regular Customer	5	5	5	15
17850.0	371	155	51080.65	Regular Customer	4	5	5	14
12482.0	575	27	20807.00	Regular Customer	5	4	5	14
13564.0	353	36	15798.83	Regular Customer	4	5	5	14
...	...	...	...	...	...	...	...	...
14349.0	9	1	133.50	Regular Customer	1	1	1	3
18005.0	3	2	226.47	Regular Customer	1	1	1	3
16528.0	3	1	244.41	Regular Customer	1	1	1	3
12660.0	10	1	245.10	Regular Customer	1	1	1	3
16189.0	15	2	215.48	Regular Customer	1	1	1	3

5833 rows × 8 columns

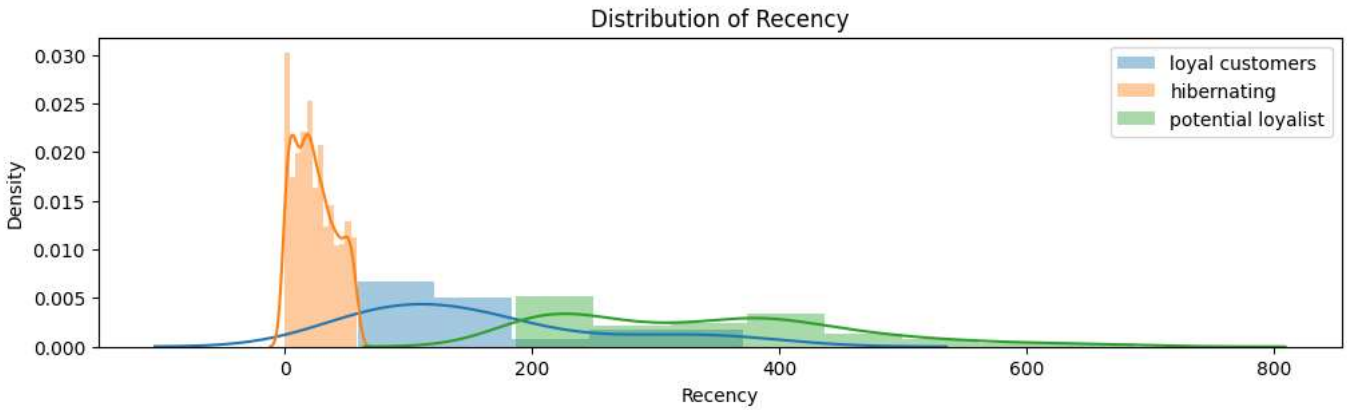
We will be assigning labels to the customers based on their transaction history

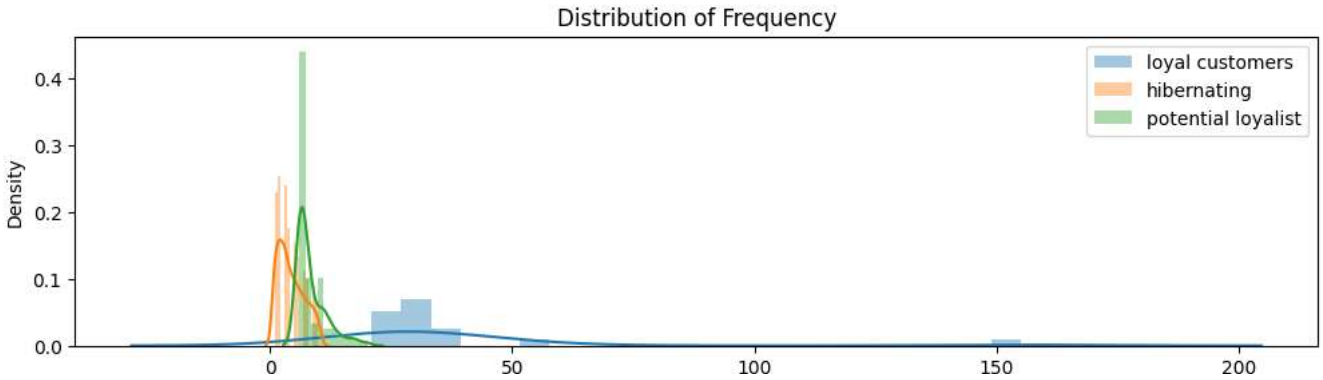
```
def assign_label(df, r_rule, fm_rule, label, colname='rfm_label'):
    df.loc[(df['r_score'].between(r_rule[0], r_rule[1]))
            & (df['f_score'].between(fm_rule[0], fm_rule[1])), colname] = label
    return df


rfm_df['rfm_label'] = ''

rfm_df = assign_label(rfm_df, (5,5), (4,5), 'champions')
rfm_df = assign_label(rfm_df, (3,4), (4,5), 'loyal customers')
rfm_df = assign_label(rfm_df, (4,5), (2,3), 'potential loyalist')
rfm_df = assign_label(rfm_df, (5,5), (1,1), 'new customers')
rfm_df = assign_label(rfm_df, (4,4), (1,1), 'promising')
rfm_df = assign_label(rfm_df, (3,3), (3,3), 'needing attention')
rfm_df = assign_label(rfm_df, (3,3), (1,2), 'about to sleep')
rfm_df = assign_label(rfm_df, (1,2), (3,4), 'at risk')
rfm_df = assign_label(rfm_df, (1,2), (5,5), 'cant loose them')
rfm_df = assign_label(rfm_df, (1,2), (1,2), 'hibernating')
rfm_df
```

Customer ID	Recency	Frequency	Monetary	Segment	r_score	f_score	m_score	rfm_sum	rfm_label
17448.0	496	46	14498.47	Regular Customer	5	5	5	15	champions
12835.0	427	41	5982.53	Regular Customer	5	5	5	15	champions
17850.0	371	155	51080.65	Regular Customer	4	5	5	14	loyal customers
12482.0	575	27	20807.00	Regular Customer	5	4	5	14	champions
13564.0	353	36	15798.83	Regular Customer	4	5	5	14	loyal customers
...	...	...	...	...	...	...	...	...	...
14349.0	9	1	133.50	Regular Customer	1	1	1	3	hibernating
18005.0	3	2	226.47	Regular Customer	1	1	1	3	hibernating
16528.0	3	1	244.41	Regular Customer	1	1	1	3	hibernating
12660.0	10	1	245.10	Regular Customer	1	1	1	3	hibernating
16189.0	15	2	215.48	Regular Customer	1	1	1	3	hibernating

5833 rows × 9 columns

```
segments = ['loyal customers', 'hibernating', 'potential loyalist']

for col in ['Recency','Frequency','Monetary']:
    fig, ax = plt.subplots(figsize=(12,3))
    for segment in segments:
        sns.distplot(rfm_df[rfm_df['rfm_label']==segment][col], label=segment)
    ax.set_title('Distribution of %s' % col)
    plt.legend()
    plt.show()
```





K means clustering for customer segmentation

```
cluster_df = pd.DataFrame()
for i in ['Recency','Frequency','Monetary']:
    cluster_df[i] = rfm_df[i]
cluster_df
```

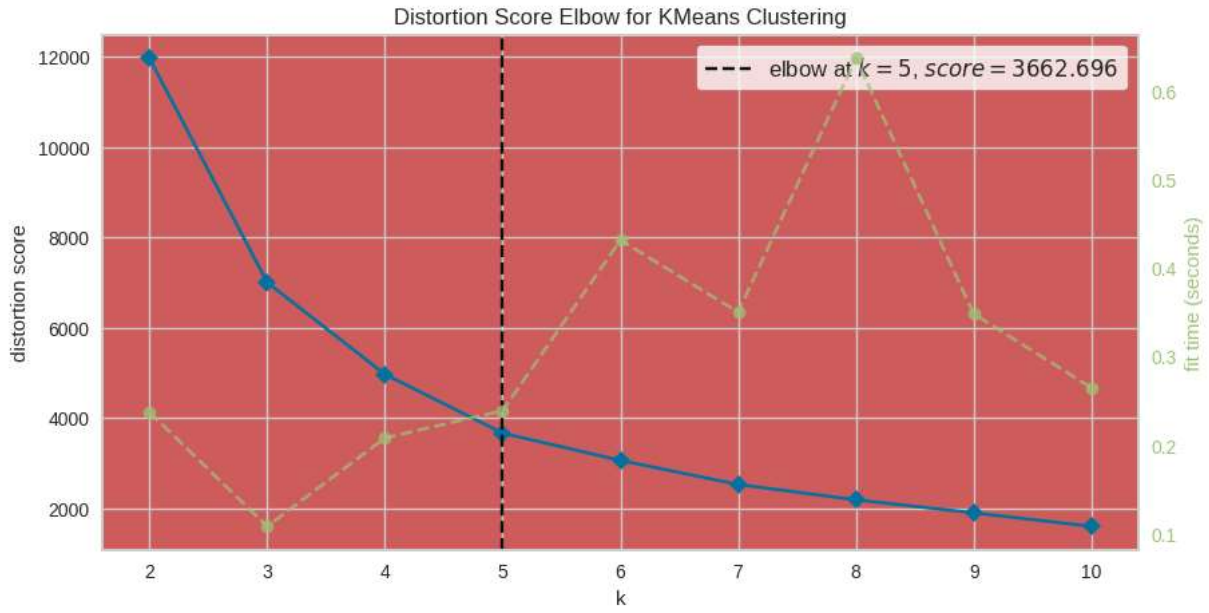
Customer ID	Recency	Frequency	Monetary
17448.0	496	46	14498.47
12835.0	427	41	5982.53
17850.0	371	155	51080.65
12482.0	575	27	20807.00
13564.0	353	36	15798.83
...	...	...	...
14349.0	9	1	133.50
18005.0	3	2	226.47
16528.0	3	1	244.41
12660.0	10	1	245.10
16189.0	15	2	215.48

5833 rows × 3 columns

```
scaler = StandardScaler()
rfm_cluster_scaled = scaler.fit_transform(cluster_df)
rfm_cluster_scaled
```

```
array([[ 1.41947546,  3.13718399,  0.98868456],
       [ 1.08905328,  2.74267387,  0.27684203],
       [ 0.82088456, 11.73750467,  4.04656794],
       ...,
       [-0.94136706, -0.41340711, -0.20280412],
       [-0.90784597, -0.41340711, -0.20274644],
       [-0.88390234, -0.33450509, -0.20522236]])
```

```
# Finding initial K value using Elbow Method
plt.figure(figsize=(10,5))
ax = plt.axes()
ax.set_facecolor("#cd5c5c")
Elbow_M = KElbowVisualizer(KMeans(), k=10)
Elbow_M.fit(rfm_cluster_scaled)
Elbow_M.show()
print("Therefore K = 5")
```




```
Therefore K = 5
```

```
kmeans = KMeans(n_clusters = 5,max_iter = 50)
kmeans.fit(rfm_cluster_scaled)
```

```
KMeans
KMeans(max_iter=50, n_clusters=5)
```

```
cluster_df['Clusters'] = kmeans.labels_
cluster_df
```

Customer ID	Recency	Frequency	Monetary	Clusters
17448.0	496	46	14498.47	2
12835.0	427	41	5982.53	2
17850.0	371	155	51080.65	4
12482.0	575	27	20807.00	3
13564.0	353	36	15798.83	2
...	...	...	...	...
14349.0	9	1	133.50	0
18005.0	3	2	226.47	0
16528.0	3	1	244.41	0
12660.0	10	1	245.10	0
16189.0	15	2	215.48	0

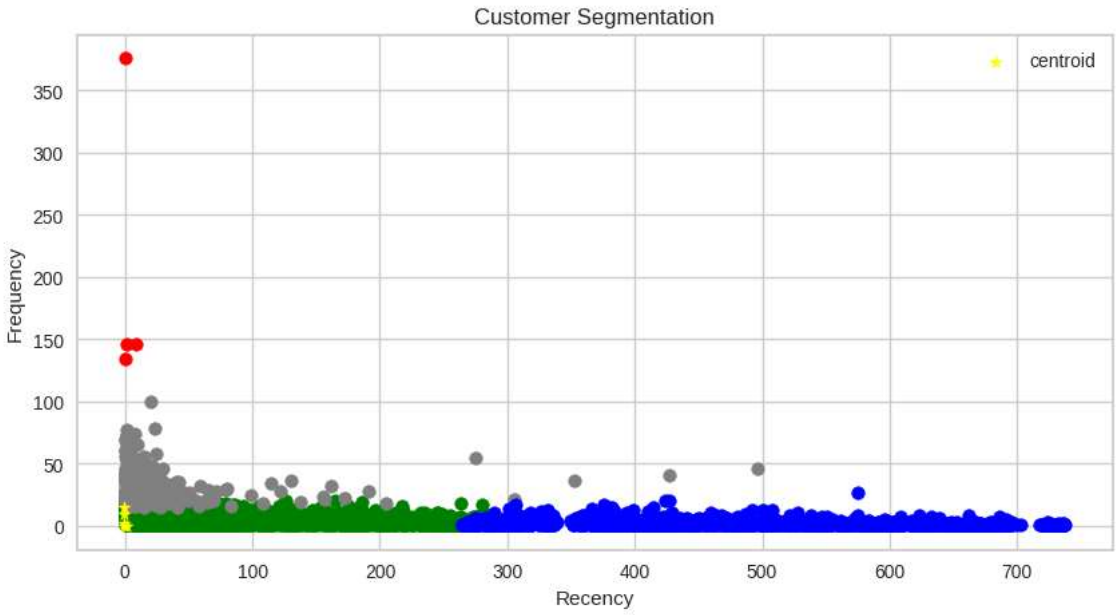
5833 rows × 4 columns

```
# Centroids of the clusters
kmeans.cluster_centers_
```

```
array([[-0.60363726, -0.07929907, -0.06882584],
       [-0.94376143, 15.32754678, 30.24766749],
```

```
            [-0.84426232,  1.87959646,  0.91879353],
            [ 1.28492501, -0.31984772, -0.1664163 ],
            [-0.83374672,  9.95598001,  6.71668351]])
```

```
# grouping the data in accorandance with each cluster seperately
one = cluster_df[cluster_df["Clusters"]==0]
two = cluster_df[cluster_df["Clusters"]==1]
three = cluster_df[cluster_df["Clusters"]==2]
four = cluster_df[cluster_df["Clusters"]==3]
five = cluster_df[cluster_df['Clusters']==4]

#Checking the quality of clustering in the data set
plt.figure(figsize=(10,5))
ax = plt.axes()
plt.scatter(one["Recency"],one["Frequency"],color='green')
plt.scatter(two["Recency"],two["Frequency"],color='red')
plt.scatter(three["Recency"],three["Frequency"],color='grey')
plt.scatter(four["Recency"],four["Frequency"],color='blue')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],color="yellow",marker="*",label="centroid")
plt.xlabel('Recency')
plt.ylabel('Frequency')
plt.title('Customer Segmentation')
plt.legend()
plt.show ()
```

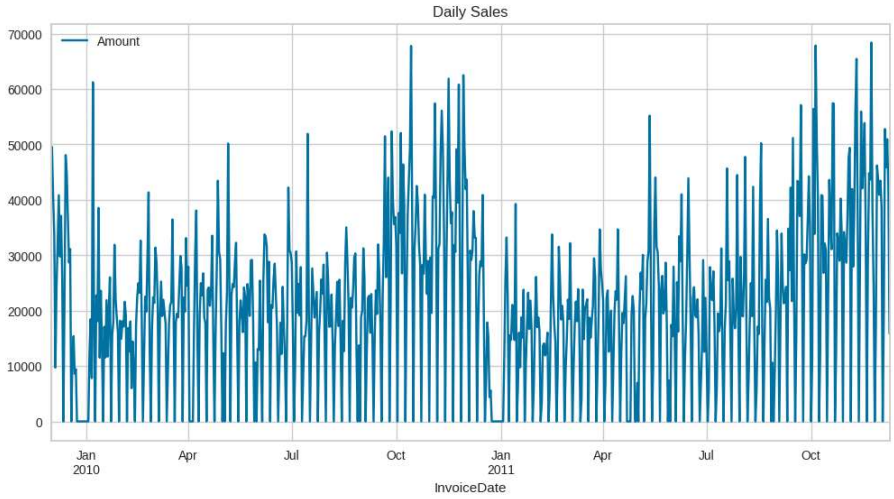


Time Series Forecasting

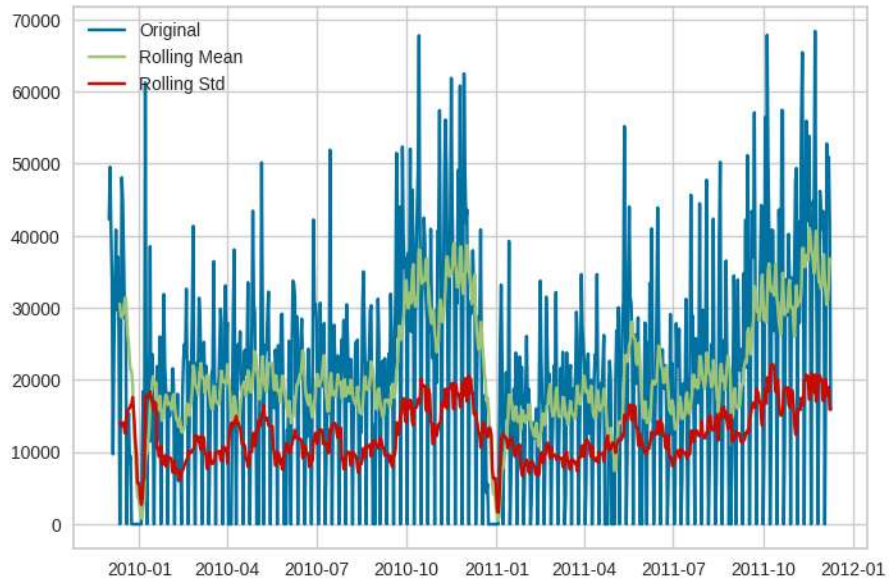
Check Stationarity:

Plot the time series data to visually inspect trends and seasonality. Use the Augmented Dickey-Fuller (ADF) test to check for stationarity.

```
# Extracting relevant columns for time series analysis
time_series_data = df[['InvoiceDate', 'Amount']]

# Set 'InvoiceDate' as the index
time_series_data.set_index('InvoiceDate', inplace=True)


# Resample data by a specific frequency (e.g., daily)
resampled_data = time_series_data.resample('D').sum()

# Plot the resampled data
resampled_data.plot(figsize=(12, 6), title='Daily Sales')
plt.show()
```




```
def test_stationarity(timeseries):
    # Plot rolling statistics
    rolling_mean = timeseries.rolling(window=12).mean()
    rolling_std = timeseries.rolling(window=12).std()
    plt.plot(figsize=(12,8))
    plt.plot(timeseries, label='Original')
    plt.plot(rolling_mean, label='Rolling Mean')
    plt.plot(rolling_std, label='Rolling Std')
    plt.legend()
    plt.show()

    # Perform ADF test
    result = adfuller(timeseries)
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    print('Critical Values:', result[4])


# Check stationarity
test_stationarity(resampled_data['Amount'])
```

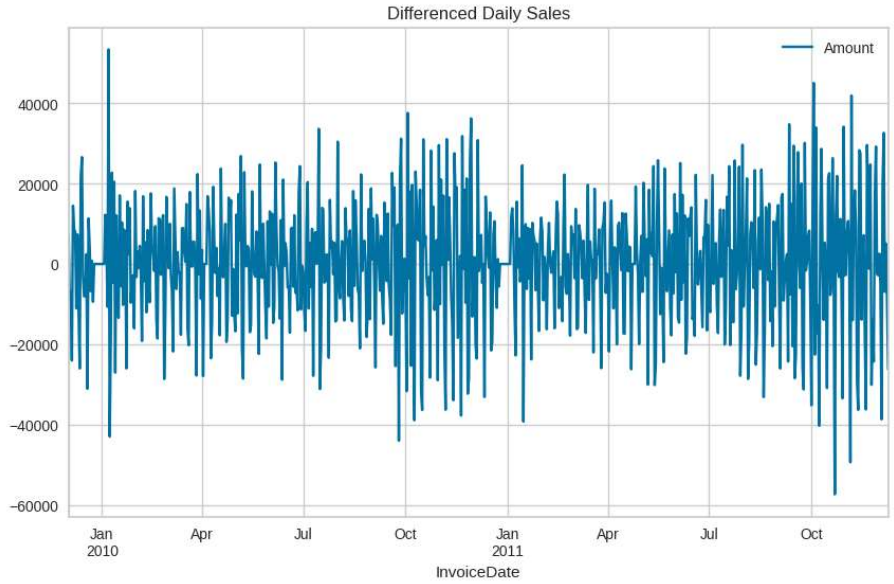



```
ADF Statistic: -2.621680358099256
p-value: 0.08859865237653525
Critical Values: {'1%': -3.439490435810785, '5%': -2.8655738086413374, '10%': -2.5689186
```

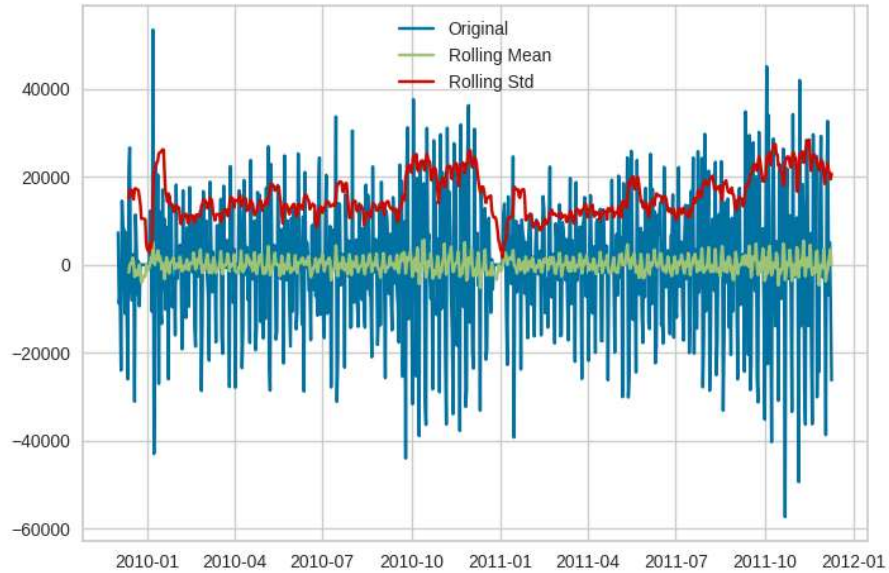
ADF Statistic (-2.62) is greater than the critical values at the 1% and 5% levels but less than the critical value at the 10% level.
p-value (0.089) is greater than the common significance level of 0.05.

```
# Make the time series stationary through differencing
resampled_data_diff = resampled_data.diff().dropna()
```

```
# Plot the differenced data
resampled_data_diff.plot(figsize=(10, 6), title='Differenced Daily Sales')
plt.show()
```




```
# Check stationarity again
test_stationarity(resampled_data_diff['Amount'])
```



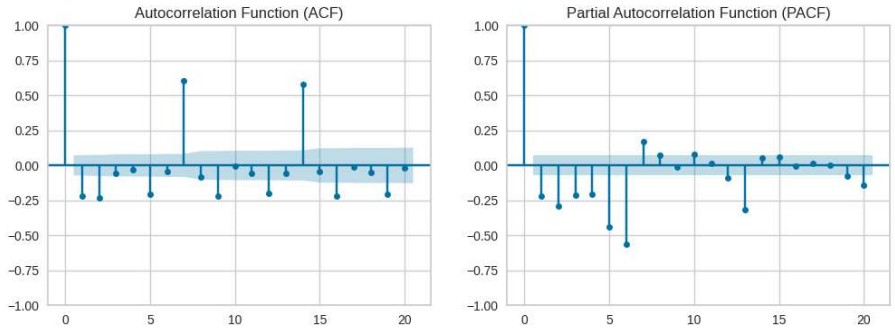

```
ADF Statistic: -9.811115628741115
p-value: 5.653726242069977e-17
Critical Values: {'1%': -3.439490435810785, '5%': -2.8655738086413374, '10%': -2.5689186
```

ADF Statistic: The ADF Statistic is significantly lower than the critical values at all common significance levels (1%, 5%, 10%). This suggests strong evidence against the null hypothesis of non-stationarity.

p-value: The p-value is extremely small (close to zero), providing strong evidence against the null hypothesis.

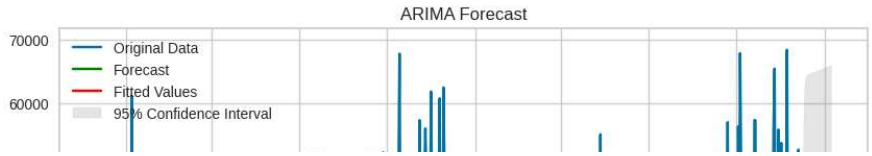
Conclusion : Based on these results, you can confidently say that the time series is now stationary after differencing.

```
# Plot ACF and PACF to determine ARIMA parameters
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

# ACF plot
plot_acf(resampled_data_diff, lags=20, ax=ax1)
ax1.set_title('Autocorrelation Function (ACF)')

# PACF plot
plot_pacf(resampled_data_diff, lags=20, ax=ax2)
ax2.set_title('Partial Autocorrelation Function (PACF)')

plt.show()
```




```
# Build ARIMA model
p, d, q = 1, 1, 1  # Adjust parameters based on your plots
model = ARIMA(resampled_data, order=(p, d, q))
results = model.fit()
```

```
# Make forecasts
forecast_steps = 30  # Adjust as needed
forecast = results.get_forecast(steps=forecast_steps)
forecast_values = forecast.predicted_mean# Visualize results
plt.figure(figsize=(10,6))
plt.plot(resampled_data.index, resampled_data['Amount'], label='Original Data')
plt.plot(forecast_values.index, forecast_values, color='green', label='Forecast')
plt.plot(results.fittedvalues.index, results.fittedvalues, color='red', label='Fitted Values')
# Add confidence intervals
conf_int = forecast.conf_int()
plt.fill_between(conf_int.index, conf_int.iloc[:, 0], conf_int.iloc[:, 1], color='gray', alpha=0.2, label='95% Confidence Interval')

# Add labels and legend
plt.xlabel('Date')
plt.ylabel('Sales Amount')
plt.title('ARIMA Forecast')
plt.legend()
plt.show()
```


ARIMA Forecast
70000
60000
Original Data
Forecast
Fitted Values
95% Confidence Interval