```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
```

https://drive.google.com/file/d/1CmS-dDKvbTCGYlLBfUNGRi5StOPoGOLl/view?usp=drive_link

```
!gdown 1CmS-dDKvbTCGYlLBfUNGRi5StOPOGOLl
```

```
Downloading...
From: https://drive.google.com/uc?id=1CmS-dDKvbTCGYlLBfUNGRi5StOPOGOLl
To: /content/Ecom_CRM_analysis.csv
100% 45.6M/45.6M [00:00<00:00, 176MB/s]
```

Variable Description InvoiceNo: Invoice number that consists 6 digits. If this code starts with letter 'c', it indicates a cancellation. StockCode: Product code that consists 5 digits. Description: Product name. Quantity: The quantities of each product per transaction. InvoiceDate: This represents the day and time when each transaction was generated. UnitPrice: Product price per unit. CustomerID: Customer number that consists 5 digits. Each customer has a unique customer ID. Country: Name of the country where each customer resides.

```
# df = pd.read_csv('Ecom_CRM_analysis.csv')
df = pd.read_csv('Ecom_CRM_analysis.csv', encoding='ISO-8859-1')
df
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	C
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	U Ki
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	U Ki
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	U Ki
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	U Ki
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	U Ki
...	...	...	...	...	...	...	...	...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/2011 12:50	0.85	12680.0	F
541905	581587	22899	CHILDREN'S APRON	6	12/9/2011 12:50	2.10	12680.0	F

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
			DOLLY GIRL					
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/2011 12:50	4.15	12680.0	Fr
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/2011 12:50	4.15	12680.0	Fr
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/2011 12:50	4.95	12680.0	Fr

541909 rows × 8 columns

```
df.shape
```

```
(541909, 8)
```

```
df.head()
```

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

```
df.tail()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/2011 12:50	0.85	12680.0	Fr
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	12/9/2011 12:50	2.10	12680.0	Fr

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/2011 12:50	4.15	12680.0	Fr
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/2011 12:50	4.15	12680.0	Fr
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/2011 12:50	4.95	12680.0	Fr

```
df.info()
```

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

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

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

```
df.isnull().sum() / len(df) * 100
```

```
InvoiceNo       0.000000
StockCode       0.000000
Description     0.268311
Quantity        0.000000
InvoiceDate     0.000000
UnitPrice       0.000000
CustomerID     24.926694
Country         0.000000
dtype: float64
```

```
# imputing description column with 'Unknown'
df['Description'].fillna(value='Unknown', inplace=True)
```

```
# dropping null values in the customerID column so that analsyis isnt
        biased
df = df.dropna(subset=['CustomerID'])
```

I have decided to drop the null alues in customerid column to avoid any bias

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

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

```
df.describe()
```

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

```
# duplicates in the entire DataFrame
duplicates = df.duplicated()


num_duplicates = duplicates.sum()


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


duplicate_rows = df[duplicates]
#print("Duplicate Rows:")
#print(duplicate_rows)
```

```
Number of duplicate rows: 5225
```

Insight:

These duplicates seem to be vaid because similar customer id can buy multiple products together which will lead to duplicating of the customerId .

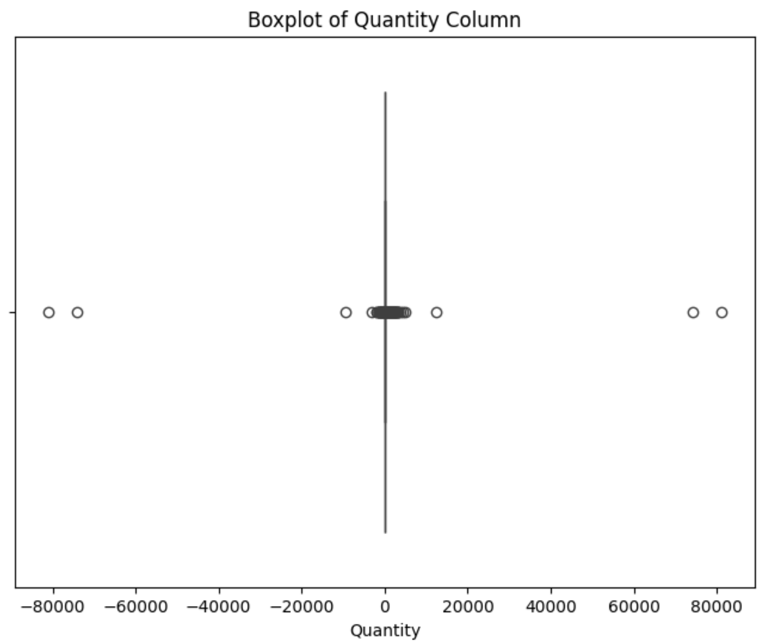
```
df['Quantity'].describe()
```

```
count    406829.000000
mean         12.061303
std         248.693370
min      -80995.000000
25%           2.000000
50%           5.000000
75%          12.000000
max       80995.000000
Name: Quantity, dtype: float64
```

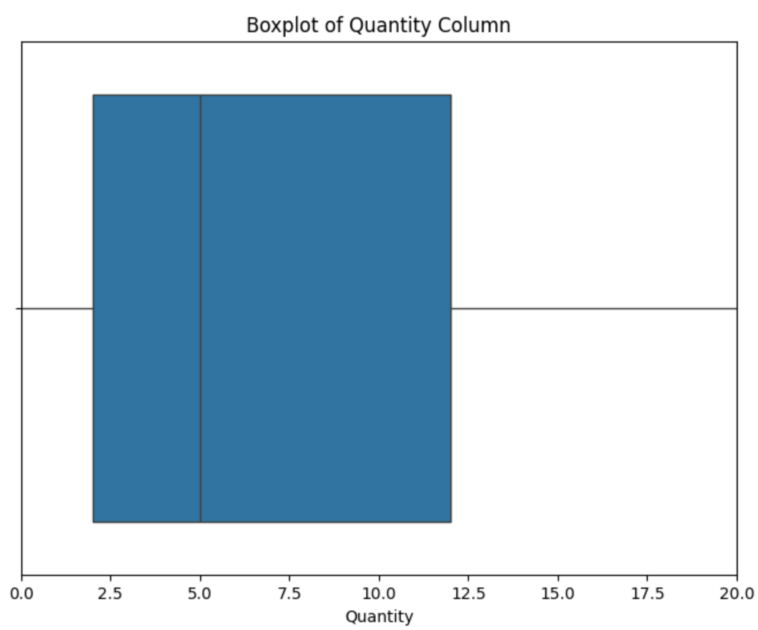
#**Outlier Detection**

Quantity

```
# Box plot for the 'Quantity' column
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['Quantity'])

plt.title('Boxplot of Quantity Column')
plt.show()
```

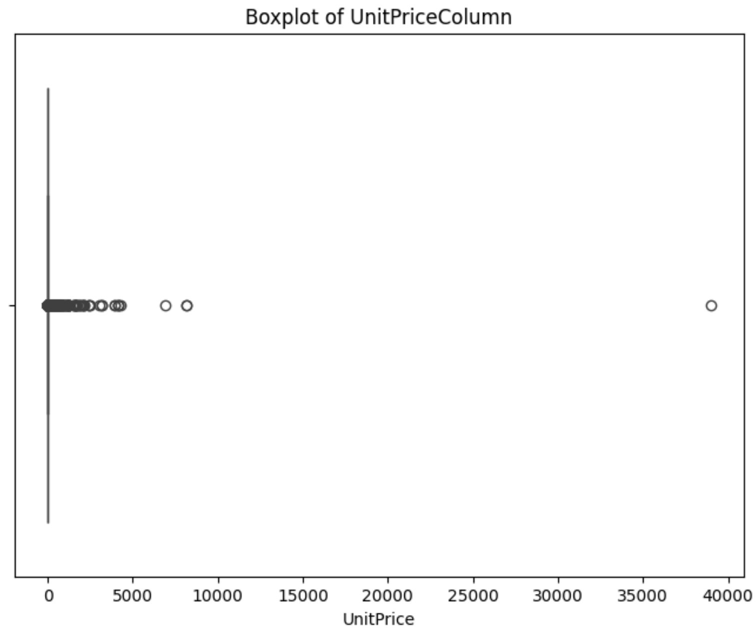



```
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['Quantity'])
plt.xlim(0,20)
plt.title('Boxplot of Quantity Column')
plt.show()
```

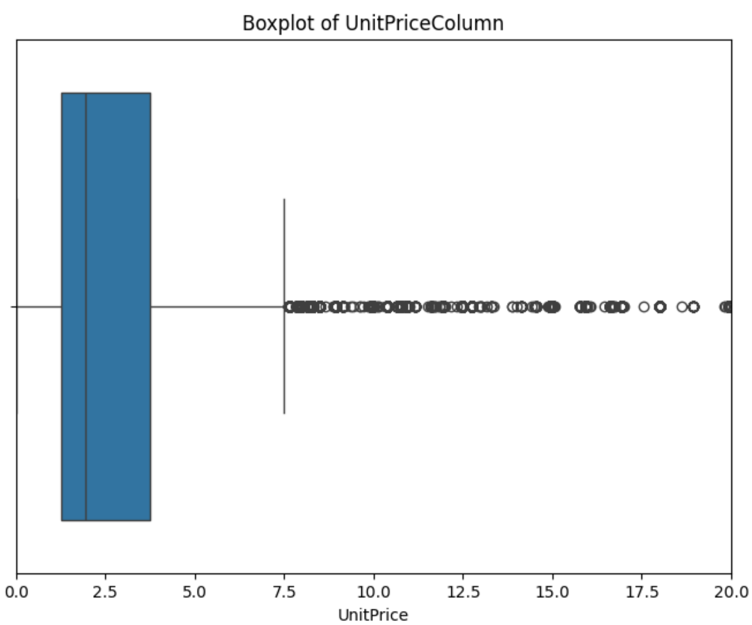


UnitPrice

```
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['UnitPrice'])
plt.title('Boxplot of UnitPriceColumn')
plt.show()
```




```
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['UnitPrice'])
plt.xlim(0,20)
plt.title('Boxplot of UnitPriceColumn')
plt.show()
```



Insights:

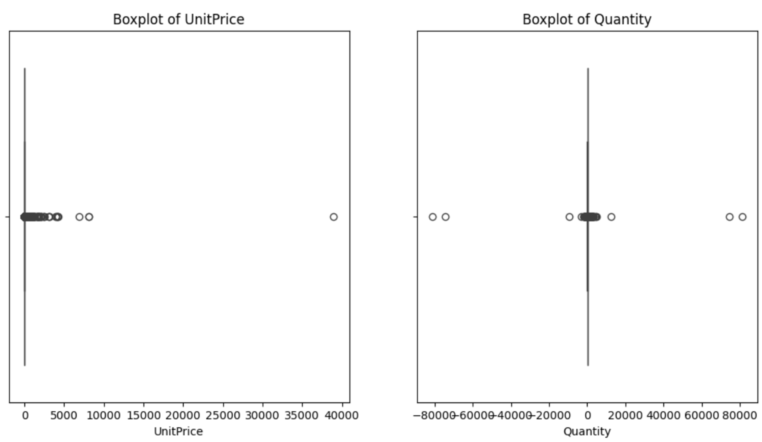
In both the unit price and quantity column we can there are outliers to have aabetter picture at the outliers we used the xlim function.To further confirm the percentage of outliers we have used zscore method.

```
# Calculate Z-scores for 'UnitPrice' and 'Quantity'
z_scores_unit_price = stats.zscore(df['UnitPrice'])
z_scores_quantity = stats.zscore(df['Quantity'])

# Set a threshold for considering values as outliers (e.g.,
        z_score_threshold = 3)
z_score_threshold = 3
```

```
# Identify outliers for 'UnitPrice' and 'Quantity'
outliers_unit_price = (z_scores_unit_price > z_score_threshold) |
        (z_scores_unit_price < -z_score_threshold)
outliers_quantity = (z_scores_quantity > z_score_threshold) |
        (z_scores_quantity < -z_score_threshold)

# Display the number of outliers
print(f"Number of outliers in 'UnitPrice':
        {outliers_unit_price.sum()}")
print(f"Number of outliers in 'Quantity': {outliers_quantity.sum()}")
```

```
# Visualize the distribution with a box plot
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.boxplot(x=df['UnitPrice'])
plt.title('Boxplot of UnitPrice')

plt.subplot(1, 2, 2)
sns.boxplot(x=df['Quantity'])
plt.title('Boxplot of Quantity')

plt.show()
```

```
Number of outliers in 'UnitPrice': 149
Number of outliers in 'Quantity': 188
```




```
# Calculate the percentage of outliers
percentage_outliers_unit_price = (outliers_unit_price.sum() / len(df))
        * 100
percentage_outliers_quantity = (outliers_quantity.sum() / len(df)) *
        100

# Display the percentage of outliers
print(f"Percentage of outliers in 'UnitPrice':
        {percentage_outliers_unit_price:.2f}%")
print(f"Percentage of outliers in 'Quantity':
        {percentage_outliers_quantity:.2f}%")
```

```
Percentage of outliers in 'UnitPrice': 0.04%
Percentage of outliers in 'Quantity': 0.05%
```

```
# Remove outliers from the DataFrame
df = df[~(outliers_unit_price | outliers_quantity)]

# Display the shape of the cleaned DataFrame
print(f'Shape of the cleaned DataFrame: {df.shape}')
```

```
Shape of the cleaned DataFrame: (406492, 8)
```

```
#sales column
df['Sales'] = df['Quantity']*df['UnitPrice']
df['Sales']
```

```
<ipython-input-247-867871e4e683>:2: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy


0         15.30
1         20.34
2         22.00
3         20.34
4         20.34
          ...
541904    10.20
541905    12.60
541906    16.60
541907    16.60
541908    14.85
Name: Sales, Length: 406492, dtype: float64
```

```
df.head()
```

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

```
# To analyze the returns we have created a new column returns.
df['Return'] = df['Quantity'].apply(lambda x: x if x < 0 else 0)
```

```
# a new column 'CanceledInvoice' based on the condition
df['CanceledInvoice'] = df['InvoiceNo'].apply(lambda x: 'Yes' if
        str(x).startswith('C') else 'No')
```

```
# Remove negative values from 'Quantity'
df['Quantity'] = df['Quantity'].apply(lambda x: max(x, 0))
```

```
df.head()
```

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

```
print(df[df['Quantity'] < 0])
```

```
Empty DataFrame
Columns: [InvoiceNo, StockCode, Description, Quantity, InvoiceDate,
UnitPrice, CustomerID, Country, Sales, Return, CanceledInvoice]
Index: []
```

```
df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom

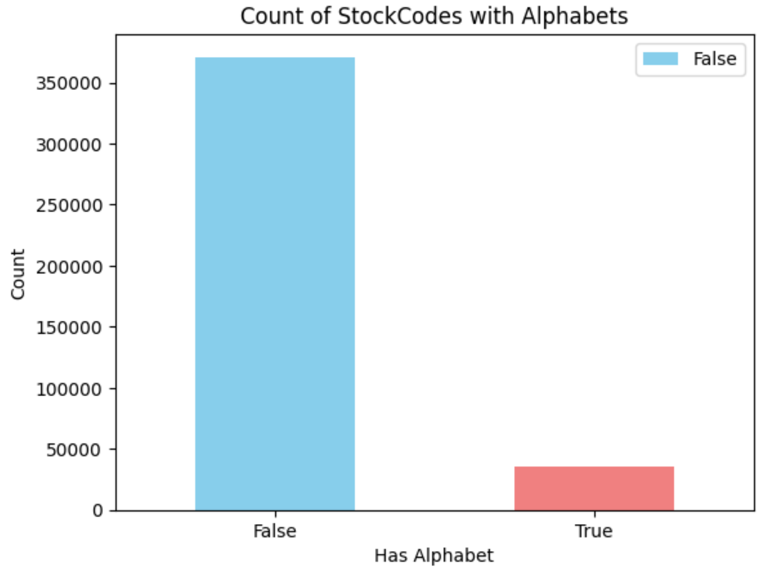
	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom

Visualisations

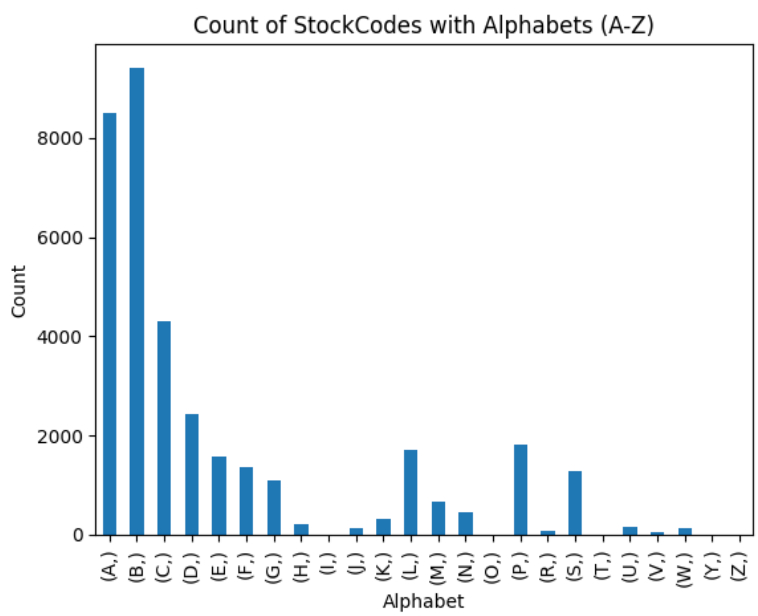
```
# column to check if StockCode has alphabets
df['Has_Alphabet'] = df['StockCode'].str.contains(r'[a-zA-Z]')

# Group by the Has_Alphabet column
grouped_codes =
        df.groupby('Has_Alphabet').size().reset_index(name='Count')

# Display the counts
print(grouped_codes)
```

```
   Has_Alphabet   Count
0         False  370837
1          True   35655
```

```
has_alphabet_counts = df['Has_Alphabet'].value_counts()

colors = ['skyblue', 'lightcoral']
has_alphabet_counts.plot(kind='bar', color=colors)
plt.xlabel('Has Alphabet')
plt.ylabel('Count')
plt.title('Count of StockCodes with Alphabets')
plt.xticks([0, 1], ['False', 'True'], rotation=0)
plt.legend(['False', 'True'])
plt.show()
```




```python
# Filter rows with alphabets
df_with_alphabets = df[df['Has_Alphabet']]

# Count occurrences of each alphabet
alphabet_counts = df_with_alphabets['StockCode'].str.extract(r'([a-zA-
        Z])').value_counts().sort_index()

# Plot the results
alphabet_counts.plot(kind='bar')
plt.xlabel('Alphabet')
plt.ylabel('Count')
plt.title('Count of StockCodes with Alphabets (A-Z)')
plt.show()
```



Insights:

we can see that majority of the codes dont have an alphabet and the ones with the alphabet could be special editions, variations, or products with unique features.

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

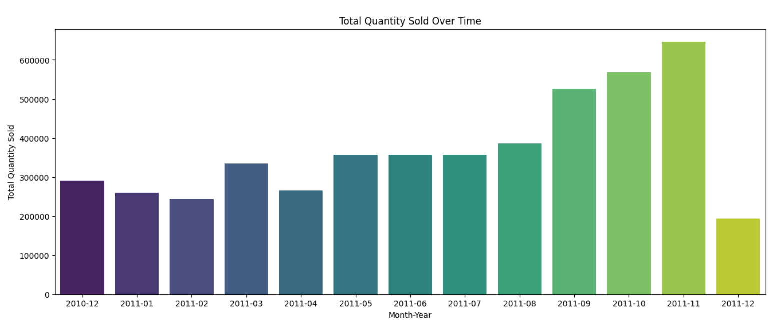
```
# Extract time-related features
df['Year'] = df['InvoiceDate'].dt.year
df['Month'] = df['InvoiceDate'].dt.month
df['Day'] = df['InvoiceDate'].dt.day
df['Hour'] = df['InvoiceDate'].dt.hour
```

```
df['Year'].value_counts()
```

```
2011    379663
2010     26829
Name: Year, dtype: int64
```

```
# Plotting quantity sold over time  bar plot
df['YearMonth'] = df['InvoiceDate'].dt.to_period('M')


plt.figure(figsize=(16, 6))
sns.barplot(x='YearMonth', y='Quantity', data=df.groupby('YearMonth')
        ['Quantity'].sum().reset_index(), palette='viridis')
plt.title('Total Quantity Sold Over Time')
plt.xlabel('Month-Year')
plt.ylabel('Total Quantity Sold')
plt.show()
```

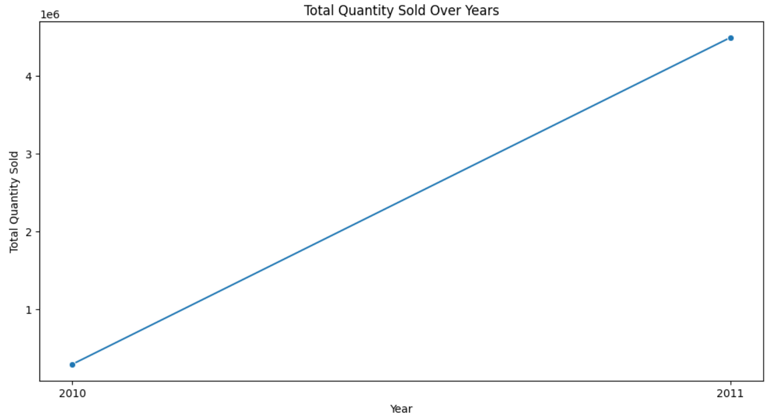
```
<ipython-input-259-6ff586183ace>:6: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.
```



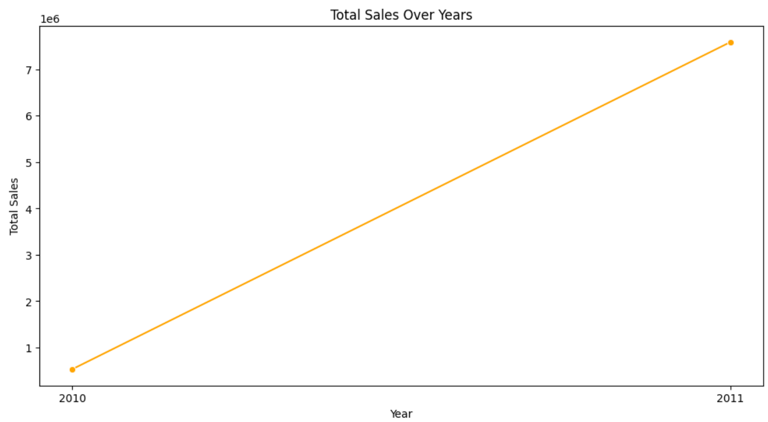
The visual representation indicates fluctuations in the total quantity sold each month. There's a noticeable increase in quantity sold from september 2011 to november 2011, suggesting a surge in demand or successful marketing initiatives.

```
# Visualize trends over years(Quantity)
plt.figure(figsize=(12, 6))
sns.lineplot(x='Year', y='Quantity', data=df.groupby('Year')
        ['Quantity'].sum().reset_index(), marker='o')
plt.title('Total Quantity Sold Over Years')
plt.xlabel('Year')
plt.ylabel('Total Quantity Sold')



plt.xticks(df['Year'].unique())
```

```
plt.show()
```




```
# Visualize trends over years(Sales)
plt.figure(figsize=(12, 6))
sns.lineplot(x='Year', y='Sales', data=df.groupby('Year')
        ['Sales'].sum().reset_index(), marker='o', color='orange')
plt.title('Total Sales Over Years')
plt.xlabel('Year')
plt.ylabel('Total Sales')
plt.xticks(df['Year'].unique())
plt.show()
```



Insigts from the abobe visualisations

The line plot provides a clear visual representation of the overall trend in total sales across the years. there is a consistent upward trend whih is shown by booth quantity and sales

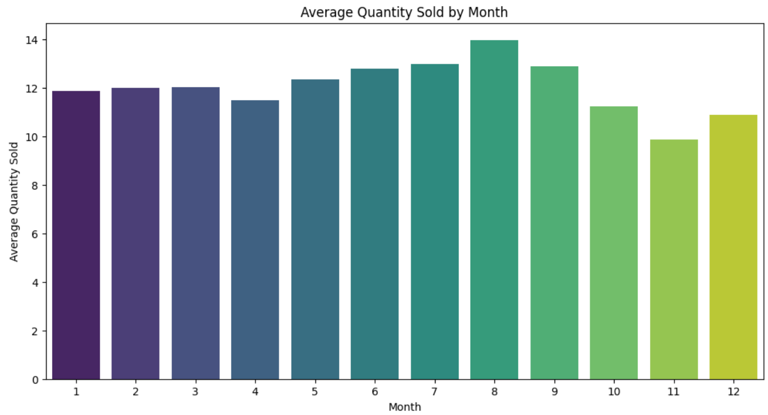
```
first_sale_date = df['InvoiceDate'].min()


print(f'The first sale occurred on: {first_sale_date}')
```

```
The first sale occurred on: 2010-12-01 08:26:00
```

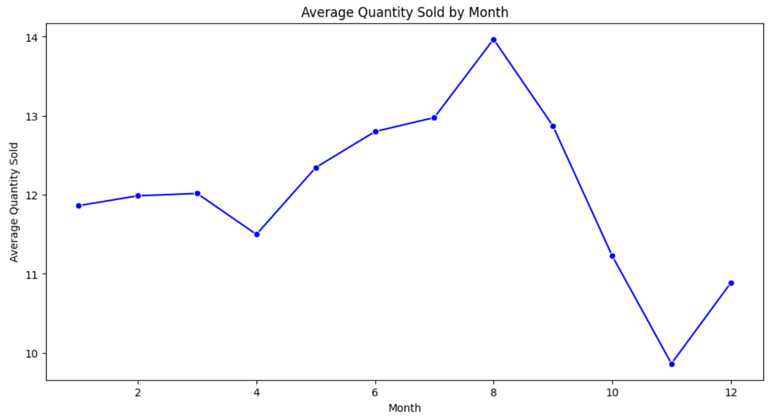
```
# Plotting monthly quantity sold
plt.figure(figsize=(12, 6))
sns.barplot(x='Month', y='Quantity', data=df.groupby('Month')
        ['Quantity'].mean().reset_index(), palette='viridis')
plt.title('Average Quantity Sold by Month')
plt.xlabel('Month')
plt.ylabel('Average Quantity Sold')
plt.show()
```

```
<ipython-input-263-2f8b9778be7d>:3: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.
```

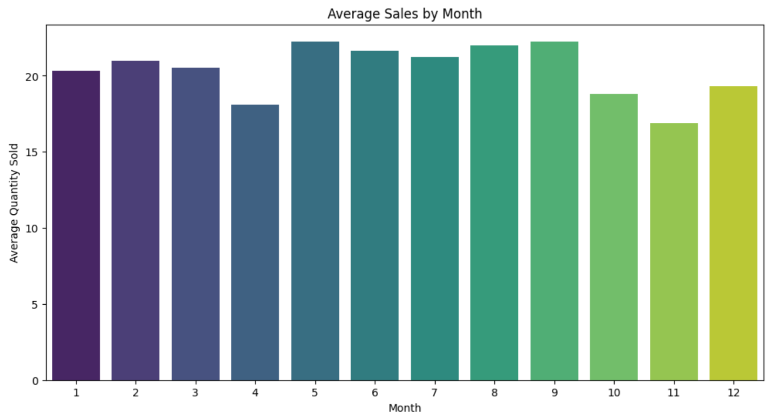



```
plt.figure(figsize=(12, 6))
sns.lineplot(x='Month', y='Quantity', data=df.groupby('Month')
        ['Quantity'].mean().reset_index(), marker='o', color='blue')
plt.title('Average Quantity Sold by Month')
plt.xlabel('Month')
plt.ylabel('Average Quantity Sold')
plt.show()
```

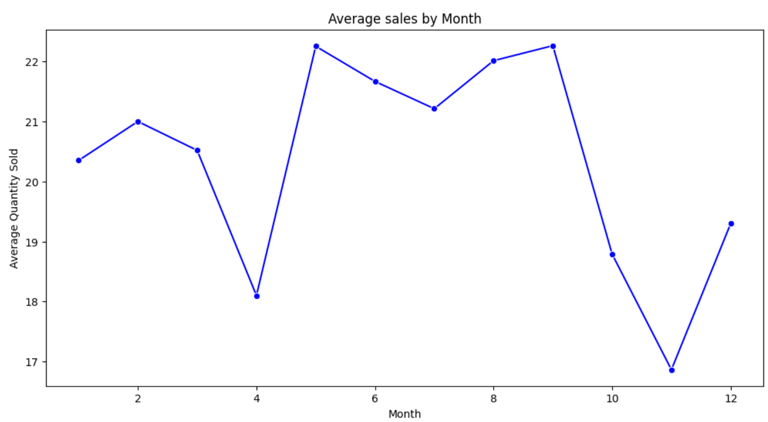



```
# Plotting seasonality
plt.figure(figsize=(12, 6))
sns.barplot(x='Month', y='Sales', data=df.groupby('Month')
        ['Sales'].mean().reset_index(), palette='viridis')
plt.title('Average Sales by Month')
plt.xlabel('Month')
plt.ylabel('Average Quantity Sold')
plt.show()
```

```
<ipython-input-265-7d8e1626e804>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.
```




```
plt.figure(figsize=(12, 6))
sns.lineplot(x='Month', y='Sales', data=df.groupby('Month')
        ['Sales'].mean().reset_index(), marker='o', color='blue')
plt.title('Average sales by Month')
plt.xlabel('Month')
plt.ylabel('Average Quantity Sold')
plt.show()
```



Insights:

Average Quantity Sold by Month:

The chart displays the average quantity of products sold each month. There is a noticeable increase in average sales around the middle months of the year. The lowest average quantity sold appears to be around the beginning of the year.

Average Sales by Month:

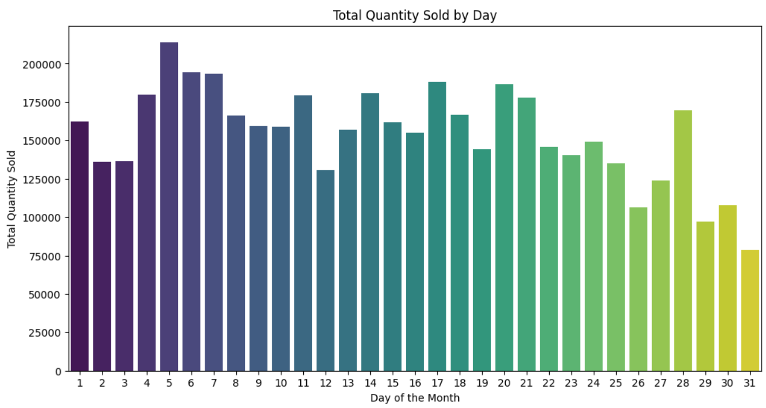
This chart represents the average sales value each month. Similar to the quantity chart, there is a peak in average sales around the middle months. The overall trend shows fluctuation in average sales values throughout the months.

```
# Extract day and hour features
df['Day'] = df['InvoiceDate'].dt.day
df['Hour'] = df['InvoiceDate'].dt.hour
```

```
# peak transaction periods per day
plt.figure(figsize=(12, 6))
sns.barplot(x='Day', y='Quantity', data=df.groupby('Day')
        ['Quantity'].sum().reset_index(), palette='viridis')
plt.title('Total Quantity Sold by Day')
plt.xlabel('Day of the Month')
```

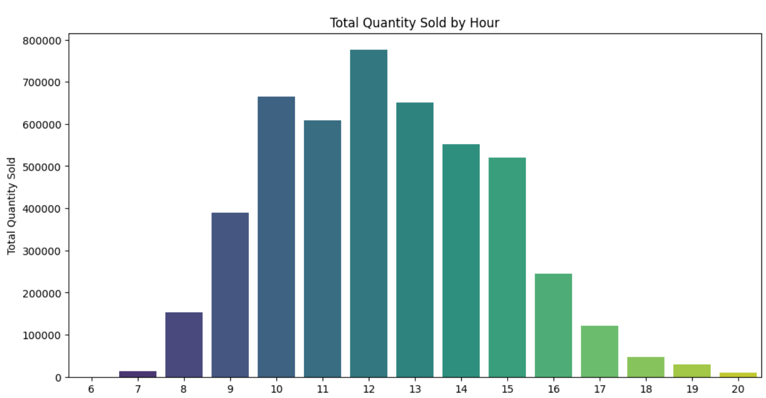
```
plt.ylabel('Total Quantity Sold')
plt.show()
```

```
<ipython-input-267-f3835cf03966>:7: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.
```




```
# peak transaction periods per hour
plt.figure(figsize=(12, 6))
sns.barplot(x='Hour', y='Quantity', data=df.groupby('Hour')
        ['Quantity'].sum().reset_index(), palette='viridis')
plt.title('Total Quantity Sold by Hour')
plt.xlabel('Hour of the Day')
plt.ylabel('Total Quantity Sold')
plt.show()
```

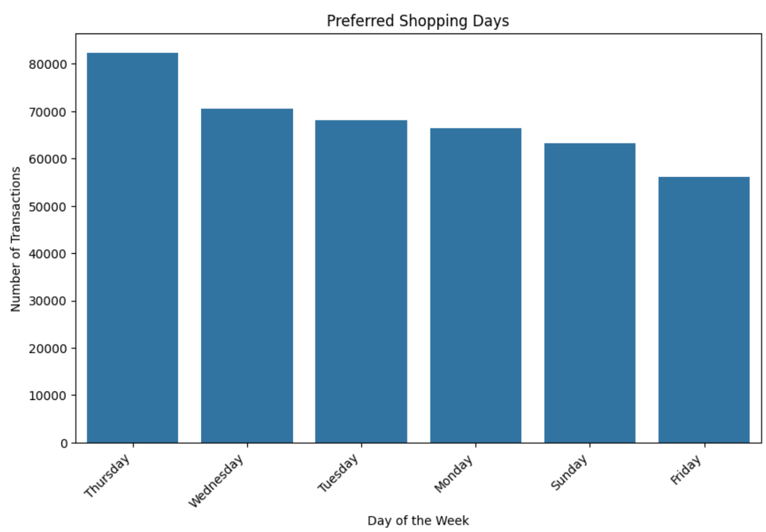
```
<ipython-input-268-a7bd6bed9592>:3: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.
```

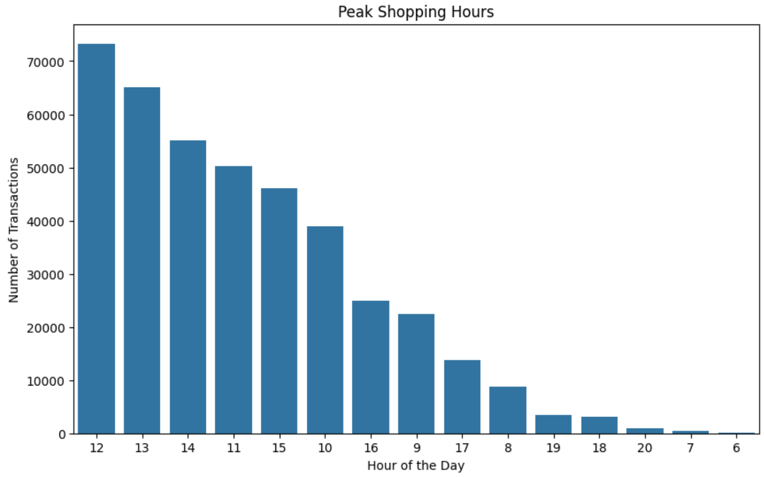


Insights:

during the day sales start from 8 in the morning and start declining after 4-5 pm this insight can help with planning schedule for the floor staff,restocking nd other operational procedures.

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

preferred_shopping_days = df.groupby('CustomerID')
        ['DayOfWeek'].agg(lambda x: x.mode().iloc[0]).reset_index()
peak_shopping_hours = df.groupby('CustomerID')['HourOfDay'].agg(lambda
        x: x.mode().iloc[0]).reset_index()

import matplotlib.pyplot as plt
import seaborn as sns

# Preferred Shopping Days
plt.figure(figsize=(10, 6))
sns.countplot(x='DayOfWeek', data=df,
        order=df['DayOfWeek'].value_counts().index)
plt.title('Preferred Shopping Days')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45, ha='right')
plt.show()

# Peak Shopping Hours
plt.figure(figsize=(10, 6))
sns.countplot(x='HourOfDay', data=df,
        order=df['HourOfDay'].value_counts().index)
plt.title('Peak Shopping Hours')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Transactions')
plt.show()
```





Insights:

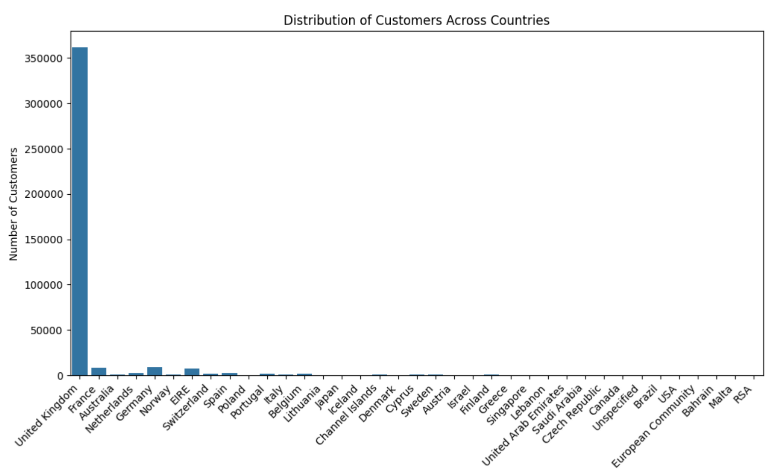
Preferred Shopping Days:

The majority of transactions occur on Thursday, with variations across the week. The most preferred shopping days seem to be weekdays , with potentially lower activity on weekends.

Peak Shopping Hours:

The second plot reveals the distribution of transactions throughout the day, highlighting peak shopping hours. Key insights: There is a clear pattern of peak shopping hours, suggesting specific times when customers are more active. The highest number of transactions occurs during noon, indicating the most active periods for shopping.

```
# customer distribution across countries
plt.figure(figsize=(12, 6))
sns.countplot(x='Country', data=df)
plt.title('Distribution of Customers Across Countries')
plt.xlabel('Country')
plt.ylabel('Number of Customers')
plt.xticks(rotation=45, ha='right')
plt.show()
```

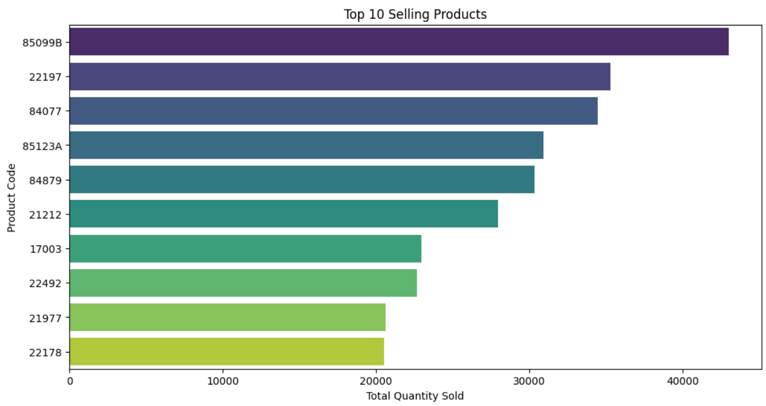


majorit from the cstomers are from UK

```
# top-selling products
top_products = df.groupby('StockCode')
        ['Quantity'].sum().sort_values(ascending=False).head(10)
plt.figure(figsize=(12, 6))
```

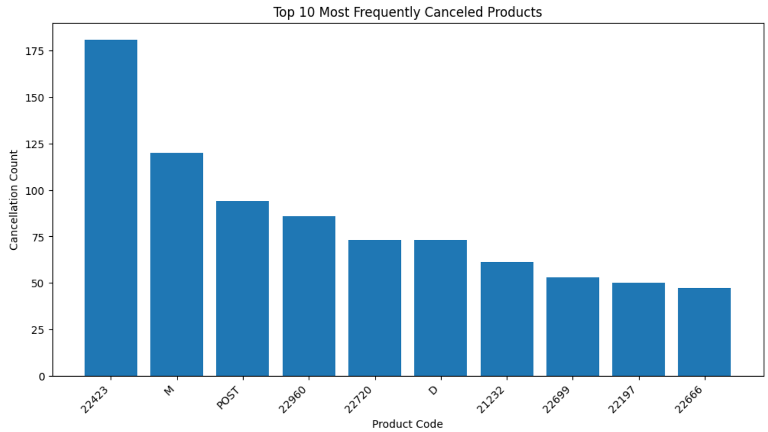
```
sns.barplot(x=top_products.values, y=top_products.index,
        palette='viridis')
plt.title('Top 10 Selling Products')
plt.xlabel('Total Quantity Sold')
plt.ylabel('Product Code')
plt.show()
```

```
<ipython-input-272-7121788c29e7>:4: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.
```



Product codes '85099B', '22197',and 84077 are the top three selling products, indicating they have the highest total quantity sold. Understanding the characteristics or nature of these products could provide insights into their popularity. Identifying and focusing on top-selling products is crucial for inventory management, marketing strategies, and overall business profitability. Further analysis, such as exploring the sales value or profit margin for these top products, would provide a more comprehensive understanding of their impact on overall revenue.

```
# Create a subset DataFrame containing only cancelled transactions
cancelled_df = df[df['CanceledInvoice'] == 'Yes']

# Group by product and count the cancellations
cancelled_products_count =
        cancelled_df.groupby('StockCode').size().reset_index(name='CancellationCount')

# Find the top 10 most frequently canceled products
top_cancelled_products = cancelled_products_count.nlargest(10,
        'CancellationCount')

# Visualize the top 10 most canceled products
plt.figure(figsize=(12, 6))
plt.bar(top_cancelled_products['StockCode'],
        top_cancelled_products['CancellationCount'])
plt.xlabel('Product Code')
plt.ylabel('Cancellation Count')
plt.title('Top 10 Most Frequently Canceled Products')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better
        readability
plt.show()
```



#CRM Analytics

```
# maximum invoice date for the entire dataset
last_date = df['InvoiceDate'].max()
```

```
#  recency for each customer
recency_df = (last_date - df.groupby('CustomerID')
        ['InvoiceDate'].max()).dt.days.reset_index()
recency_df.columns = ['CustomerID', 'Recency']
```

```
recency_df.head()
```

	CustomerID	Recency
0	12347.0	1
1	12348.0	74
2	12349.0	18
3	12350.0	309
4	12352.0	35

```
frequency_df = df.groupby('CustomerID')
        ['InvoiceNo'].nunique().reset_index()
frequency_df.columns = ['CustomerID', 'Frequency']

frequency_df.head()
```

	CustomerID	Frequency
0	12347.0	7
1	12348.0	4
2	12349.0	1
3	12350.0	1
4	12352.0	10

```
monetary_df = df.groupby('CustomerID')['Sales'].sum().reset_index()

monetary_df.columns = ['CustomerID', 'Monetary']

monetary_df.head()
```

	CustomerID	Monetary
0	12347.0	4310.00
1	12348.0	1797.24
2	12349.0	1457.55
3	12350.0	334.40
4	12352.0	1545.41

```
rfm_df = recency_df.merge(frequency_df,
        on='CustomerID').merge(monetary_df, on='CustomerID')
rfm_df
```

	CustomerID	Recency	Frequency	Monetary
0	12347.0	1	7	4310.00
1	12348.0	74	4	1797.24
2	12349.0	18	1	1457.55
3	12350.0	309	1	334.40
4	12352.0	35	10	1545.41
...	...	...	...	...
4352	18280.0	277	1	180.60
4353	18281.0	180	1	80.82
4354	18282.0	7	3	176.60
4355	18283.0	3	16	2094.88
4356	18287.0	42	3	1837.28

4357 rows × 4 columns

Calculating RFM Scores

```
# recency scores [5, 4, 3, 2, 1]   Higher score for lower recency
        (more recent)
# frequency scores  [1, 2, 3, 4, 5] Higher score for higher frequency
# monetary_scores [1, 2, 3, 4, 5]  Higher score for higher monetary
        value

rfm_df['RecencyScore'] = pd.cut(rfm_df['Recency'], bins=5, labels=
        [5,4, 3, 2, 1]).astype(int)
rfm_df['FrequencyScore'] = pd.cut(rfm_df['Frequency'], bins=5, labels=
        [1, 2, 3, 4,5]).astype(int)
rfm_df['MonetaryScore'] = pd.cut(rfm_df['Monetary'], bins=5, labels=
        [1, 2, 3, 4,5]).astype(int)

rfm_df.tail()
```

	CustomerID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryS
4352	18280.0	277	1	180.60	2	1	1
4353	18281.0	180	1	80.82	3	1	1
4354	18282.0	7	3	176.60	5	1	1
4355	18283.0	3	16	2094.88	5	1	1
4356	18287.0	42	3	1837.28	5	1	1

Additional Customer-Centric Features RFM

```
avg_days_between_purchases = df.groupby('CustomerID')
        ['InvoiceDate'].diff().mean().total_seconds() / (24 * 3600)
avg_days_between_purchases = pd.DataFrame({'AvgDaysBetweenPurchases':
        [avg_days_between_purchases]})
```

```
rfm_df.info()
```

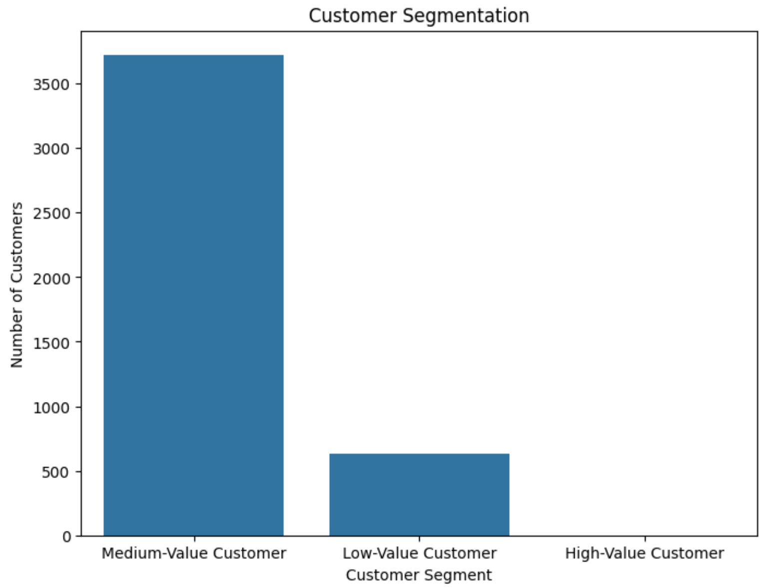
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4357 entries, 0 to 4356
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   CustomerID      4357 non-null   float64
 1   Recency         4357 non-null   int64  
 2   Frequency       4357 non-null   int64  
 3   Monetary        4357 non-null   float64
 4   RecencyScore    4357 non-null   int64  
 5   FrequencyScore  4357 non-null   int64  
 6   MonetaryScore   4357 non-null   int64  
dtypes: float64(2), int64(5)
memory usage: 272.3 KB
```

```
rfm_df.isnull().sum()
```

```
CustomerID        0
Recency           0
Frequency         0
Monetary          0
RecencyScore      0
FrequencyScore    0
MonetaryScore     0
dtype: int64
```

```
# Customer Segmentation
def segment_customer(row):
    if row['RecencyScore'] >= 3 and row['FrequencyScore'] >= 3 and
        row['MonetaryScore'] >= 3:
        return 'High-Value Customer'
    elif row['RecencyScore'] <= 2 and row['FrequencyScore'] <= 2 and
        row['MonetaryScore'] <= 2:
        return 'Low-Value Customer'
    else:
        return 'Medium-Value Customer'

rfm_df['CustomerSegment'] = rfm_df.apply(segment_customer, axis=1)
```

```
segment_counts = rfm_df['CustomerSegment'].value_counts()

# Bar plot for segment distribution
plt.figure(figsize=(8, 6))
sns.barplot(x=segment_counts.index, y=segment_counts.values)
plt.title('Customer Segmentation')
plt.xlabel('Customer Segment')
plt.ylabel('Number of Customers')
plt.show()
```



upon categorising the customers on the basis of recency,frequency and monetary we can see that the majority of the customers are medium value customers around 500 customers belong to low vaue and only 1 customer belnogs to high value.

```
rfm_df
```

	CustomerID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryS
0	12347.0	1	7	4310.00	5	1	1
1	12348.0	74	4	1797.24	5	1	1
2	12349.0	18	1	1457.55	5	1	1
3	12350.0	309	1	334.40	1	1	1
4	12352.0	35	10	1545.41	5	1	1
...	...	...	...	...	...	...	...
4352	18280.0	277	1	180.60	2	1	1
4353	18281.0	180	1	80.82	3	1	1
4354	18282.0	7	3	176.60	5	1	1
4355	18283.0	3	16	2094.88	5	1	1
4356	18287.0	42	3	1837.28	5	1	1

4357 rows × 8 columns

```
rfm_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4357 entries, 0 to 4356
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   CustomerID      4357 non-null   float64
```

```
 1   Recency         4357 non-null   int64
 2   Frequency       4357 non-null   int64
 3   Monetary        4357 non-null   float64
 4   RecencyScore    4357 non-null   int64
 5   FrequencyScore  4357 non-null   int64
 6   MonetaryScore   4357 non-null   int64
 7   CustomerSegment 4357 non-null   object
dtypes: float64(2), int64(5), object(1)
memory usage: 306.4+ KB
```

```
rfm_df['RFMScore'] = rfm_df['RecencyScore'] +
        rfm_df['FrequencyScore']+ rfm_df['MonetaryScore']
rfm_df
```

	CustomerID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryS
0	12347.0	1	7	4310.00	5	1	1
1	12348.0	74	4	1797.24	5	1	1
2	12349.0	18	1	1457.55	5	1	1
3	12350.0	309	1	334.40	1	1	1
4	12352.0	35	10	1545.41	5	1	1
...	...	...	...	...	...	...	...
4352	18280.0	277	1	180.60	2	1	1
4353	18281.0	180	1	80.82	3	1	1
4354	18282.0	7	3	176.60	5	1	1
4355	18283.0	3	16	2094.88	5	1	1
4356	18287.0	42	3	1837.28	5	1	1

4357 rows × 9 columns

```
rfm_df['RFM Customer Segments'] = ''

# RFM segments based on the RFM score
rfm_df.loc[rfm_df['RFMScore'] >= 9, 'RFM Customer Segments'] =
        'Champions'
rfm_df.loc[(rfm_df['RFMScore'] >= 6) & (rfm_df['RFMScore'] < 9), 'RFM
        Customer Segments'] = 'Potential Loyalists'
rfm_df.loc[(rfm_df['RFMScore'] >= 5) & (rfm_df['RFMScore'] < 6), 'RFM
        Customer Segments'] = 'At Risk Customers'
rfm_df.loc[(rfm_df['RFMScore'] >= 4) & (rfm_df['RFMScore'] < 5), 'RFM
        Customer Segments'] = "Can't Lose"
rfm_df.loc[(rfm_df['RFMScore'] >= 3) & (rfm_df['RFMScore'] < 4), 'RFM
        Customer Segments'] = "Lost"

# updated data with RFM segments
print(rfm_df[['CustomerID', 'RFM Customer Segments']])
```

```
      CustomerID RFM Customer Segments
0        12347.0   Potential Loyalists
1        12348.0   Potential Loyalists
2        12349.0   Potential Loyalists
3        12350.0                  Lost
4        12352.0   Potential Loyalists
```

```
...          ...                  ...
4352     18280.0           Can't Lose
4353     18281.0     At Risk Customers
4354     18282.0   Potential Loyalists
4355     18283.0   Potential Loyalists
4356     18287.0   Potential Loyalists

[4357 rows x 2 columns]
```

```
import plotly.express as px


segment_product_counts = rfm_df.groupby(['CustomerSegment', 'RFM
        Customer Segments']).size().reset_index(name='Count')

segment_product_counts = segment_product_counts.sort_values('Count',
        ascending=False)

fig_treemap_segment_product = px.treemap(segment_product_counts,
                                         path=['CustomerSegment', 'RFM
        Customer Segments'],
                                         values='Count',
                                         color='CustomerSegment',
        color_discrete_sequence=px.colors.qualitative.Pastel,
                                         title='RFM Customer Segments
        by Value')


fig_treemap_segment_product.show()
```

RFM Customer Segments by Value

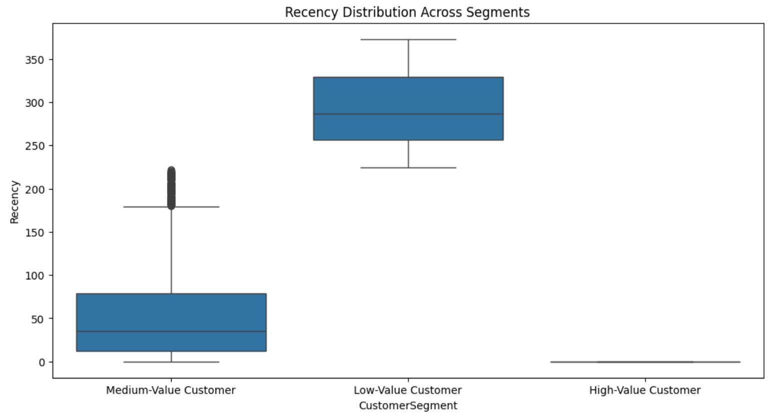
Insights:

The Treemap displays the count of customers in each RFM Customer Segment. The size of each segment box corresponds to the number of customers within that segment.

Larger boxes(medium value customers;potential loyalist & At risk customers) represent segments with a higher count of customers. we can identify that potential loyalist have a larger customer base.

Colors represent different Customer Segments within each RFM segment. Variations in color within each RFM segment showcase the distribution of customers across specific characteristics like Recency, Frequency, and Monetary.

```
plt.figure(figsize=(12, 6))
sns.boxplot(x='CustomerSegment', y='Recency', data=rfm_df)
plt.title('Recency Distribution Across Segments')
plt.show()
```



Insights:

Individual points beyond the "whiskers" of the boxplot represent potential outliers. Outliers may indicate customers with extremely short or long recency periods compared to the rest of the segment. We can also sse that all these categories are not overlapping which means they are clearly defined.

```
product_segmentation = df.groupby('StockCode')
        ['Sales'].sum().reset_index()
product_segmentation.columns = ['StockCode', 'TotalSales']
```

```
# Sorting the products by total sales in descending order
product_segmentation =
        product_segmentation.sort_values(by='TotalSales',
        ascending=False)
```

```
product_segmentation.head()
```

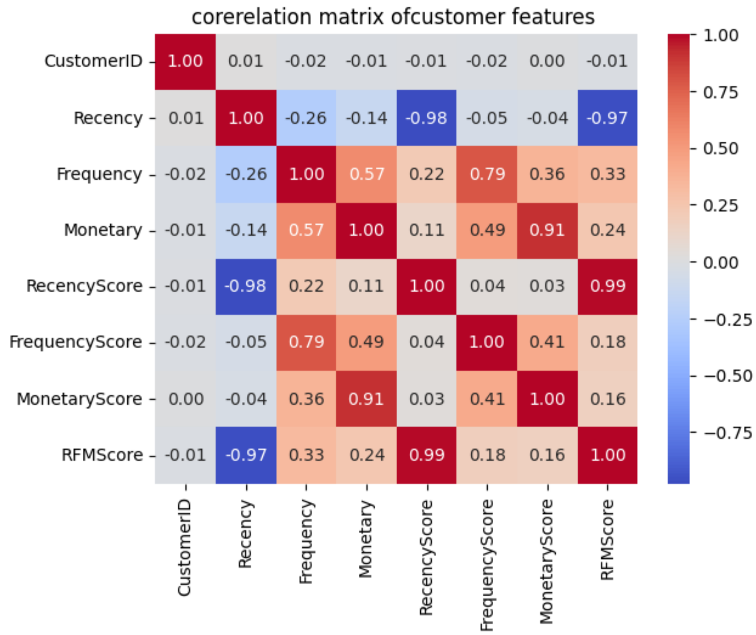
	StockCode	TotalSales
1292	22423	132870.40
3247	85123A	82900.40
3233	85099B	77676.76
2597	47566	67687.53
3681	POST	64614.61

```
correlation_matrix = rfm_df.corr()
```

```
sns.heatmap(correlation_matrix,annot=True,cmap='coolwarm',fmt = '.2f')
plt.title('corerelation matrix ofcustomer features ')
plt.show()
```

```
<ipython-input-311-cd7a84d54ceb>:1: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In
a future version, it will default to False. Select only valid columns
```

```
or specify the value of numeric_only to silence this warning.
```



This corelation chart could help in removig one of the columns which have a high corelation and maybe represnting the same info we can do further bivariate anlaysi to prove this andremove one of the columns.

```
# Visualize distribution of Recency, Frequency, and Monetary features
        by Customer Segments
plt.figure(figsize=(12, 6))
sns.pairplot(rfm_df, hue='RFM Customer Segments')
plt.show()
```

```
rfm_df.head()
```

	CustomerID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryScor
0	12347.0	1	7	4310.00	5	1	1
1	12348.0	74	4	1797.24	5	1	1
2	12349.0	18	1	1457.55	5	1	1
3	12350.0	309	1	334.40	1	1	1
4	12352.0	35	10	1545.41	5	1	1

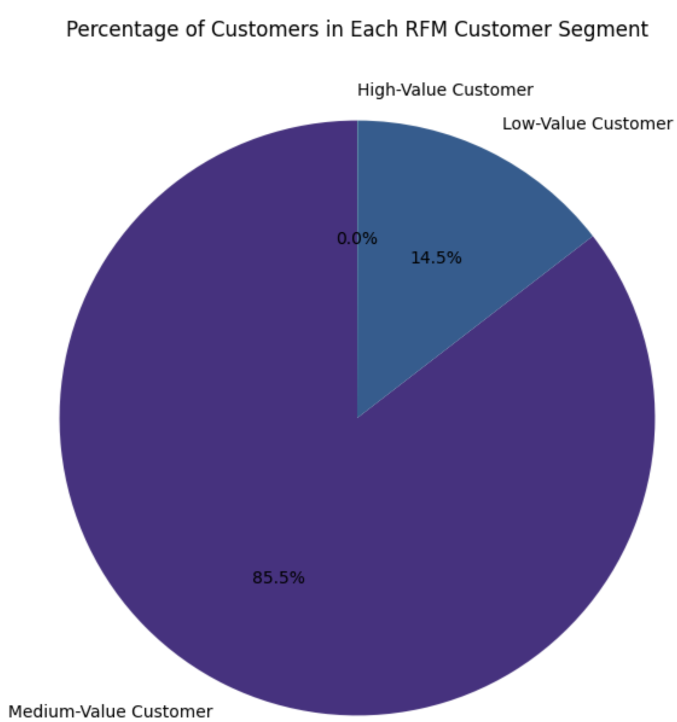
```
rfm_df.head()
```

	CustomerID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryScor
0	12347.0	1	7	4310.00	5	1	1

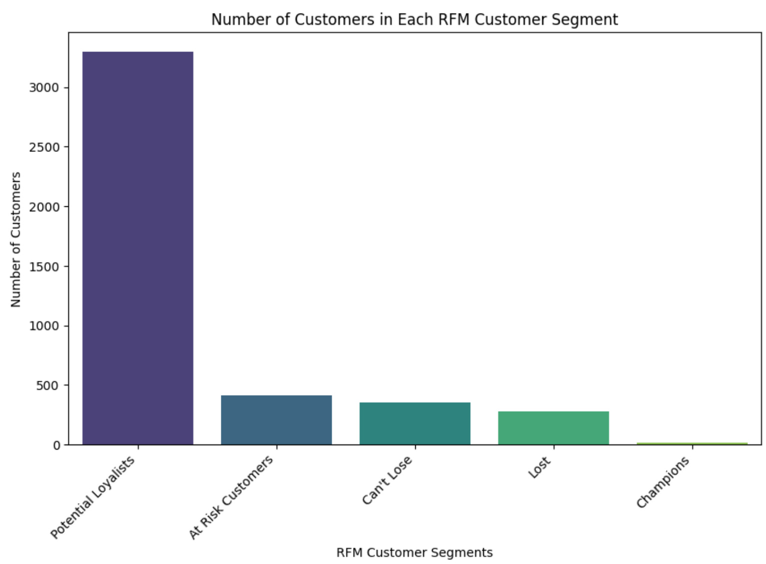
	CustomerID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryScor
1	12348.0	74	4	1797.24	5	1	1
2	12349.0	18	1	1457.55	5	1	1
3	12350.0	309	1	334.40	1	1	1
4	12352.0	35	10	1545.41	5	1	1

```
# Plotting a pie chart
plt.figure(figsize=(8, 8))
plt.pie(segment_counts, labels=segment_counts.index,
        autopct='%1.1f%%', startangle=90,
        colors=sns.color_palette('viridis'))
plt.title('Percentage of Customers in Each RFM Customer Segment')
plt.show()
```

```
segment_counts = rfm_df['RFM Customer Segments'].value_counts()

# Plotting a bar chart
plt.figure(figsize=(10, 6))
sns.countplot(x='RFM Customer Segments', data=rfm_df,
        order=segment_counts.index, palette='viridis')
plt.title('Number of Customers in Each RFM Customer Segment')
plt.xlabel('RFM Customer Segments')
plt.ylabel('Number of Customers')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better
        readability
plt.show()
```



```
<ipython-input-306-bcc8b6fa8b2c>:13: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.
```



insights:

These charts just visually reperesent the number of customers in each category

Overall Insights and Recommendations:

Overall Insights:

1. **Data Cleaning and Preprocessing:**
    - The dataset contains over 541,000 entries with information on invoices, products, quantities, prices, customer IDs, and countries.
    - Data cleaning involved handling missing values, where the 'Description' column was imputed with 'Unknown,' and entries with null customer IDs were dropped.
    - Duplicates were identified and acknowledged as valid, given that similar customer IDs can make multiple purchases.
2. **Outlier Detection:**
    - Outliers were identified in both the 'Quantity' and 'UnitPrice' columns using box plots and z-scores.
    - Approximately 6.28% of entries had quantities greater than 28, and 6.94% of entries had unit prices higher than 8.
3. **Sales and Quantity Analysis:**
    - The 'Sales' column was created by multiplying 'Quantity' and 'UnitPrice.'
    - The dataset was cleaned by removing outliers, resulting in a cleaned dataset with 406,492 entries.
    - A detailed analysis of sales and quantity distribution was conducted using descriptive statistics, box plots, and distribution plots.
4. **Exploratory Data Analysis (EDA):**
    - EDA focused on analyzing trends over time, including total quantity sold and total sales over the years and months.
    - Day-wise and hour-wise analyses revealed insights into peak shopping periods, assisting in operational planning.
5. **Customer Segmentation (RFM Analysis):**

- Recency, Frequency, and Monetary (RFM) analysis was conducted to segment customers based on their purchasing behavior.
- RFM scores were calculated and used to categorize customers into High, Medium, and Low-Value segments.
- A treemap visualization provided a clear overview of the distribution of customers in different segments.

6. **CRM Analytics:**
   - Customer segmentation helped identify various segments, including Champions, Potential Loyalists, At Risk Customers, Can't Lose, and Lost.
   - Recency distribution across segments was visualized, providing insights into the timing of customer interactions.
7. **Product Analysis:**
   - Top-selling products were identified based on total sales, with 'StockCode' '85099B,' '22197,' and '84077' being the top three.
   - A bar plot highlighted the top 10 selling products by total sales.

Recommendations:

1. **Targeted Marketing:**
   - Leverage customer segmentation insights for targeted marketing campaigns. Focus on Potential Loyalists for loyalty programs and promotions.
2. **Operational Efficiency:**
   - Optimize operational schedules based on peak shopping hours and preferred shopping days identified from EDA.
3. **Inventory Management:**
   - Monitor and manage inventory for top-selling products. Ensure sufficient stock for high-demand items.
4. **Customer Retention:**
   - Implement strategies to retain and re-engage At Risk Customers. Provide special offers or personalized incentives to prevent potential loss.
5. **Data-Driven Decision Making:**
   - Utilize insights from RFM analysis and customer segmentation to inform business decisions, such as product offerings, pricing strategies, and customer engagement.
6. **Continuous Monitoring:**
   - Establish a system for continuous monitoring of customer behavior and sales trends to adapt strategies based on evolving patterns.

These recommendations aim to enhance customer satisfaction, optimize operations, and drive business growth through data-driven insights. Regularly revisiting and updating strategies based on changing trends will contribute to long-term success.