In [1]:
```
pip install gap-stat
```

```
Requirement already satisfied: gap-stat in /Users/vishak/anaconda3/lib/pyth
on3.11/site-packages (2.0.3)
Requirement already satisfied: numpy in /Users/vishak/anaconda3/lib/python
3.11/site-packages (from gap-stat) (1.24.3)
Requirement already satisfied: pandas in /Users/vishak/anaconda3/lib/python
3.11/site-packages (from gap-stat) (2.1.4)
Requirement already satisfied: scipy in /Users/vishak/anaconda3/lib/python
3.11/site-packages (from gap-stat) (1.11.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/vishak/anac
onda3/lib/python3.11/site-packages (from pandas->gap-stat) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/vishak/anaconda3/lib/
python3.11/site-packages (from pandas->gap-stat) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /Users/vishak/anaconda3/li
b/python3.11/site-packages (from pandas->gap-stat) (2023.3)
Requirement already satisfied: six>=1.5 in /Users/vishak/anaconda3/lib/pyth
on3.11/site-packages (from python-dateutil>=2.8.2->pandas->gap-stat) (1.16.
0)
Note: you may need to restart the kernel to use updated packages.
```

About the Data

The dataset captures transactions from December 2009 to December 2011 from a UK-based online retailer. Fields include:

- InvoiceNo - Invoice number - a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter ‘c’, it indicates a cancellation.
- StockCode - a 5-digit integral number uniquely assigned to each distinct product
- Description - product name
- Quantity - the quantities of each product (item) per transaction
- InvoiceDate - the day and time when each transaction was generated
- UnitPrice - product price per unit
- CustomerID - a 5-digit integral number uniquely assigned to each customer
- Country - the name of the country where each customer resides For more details about the dataset, refer to the UCI Machine Learning Repository.

The project is structured into two main parts:

1. EDA (Exploratory Data Analysis) and Extraction of Business Insights:

- Identify the most frequently purchased products.
- Calculate the average price per order.
- Analyze overall sales trends throughout the quarters.
- Determine the distribution of order sizes per invoice.
- Explore the preferred shopping days during the week.
- Identify countries most represented in the dataset.
- Examine revenue by country.
- Calculate monthly revenue and determine the percentage revenue based on countries.
- Identify the month with the highest sales.
- Analyze popular products and their sales variation over time.

- Investigate the impact of cancellations on overall sales trends.

2. Customer Segmentation using RFM Analysis:

- Utilize RFM (Recency, Frequency, Monetary) analysis to segment customers based on their purchase patterns.
- Characterize and quantify customer personas based on their RFM profiles.

This comprehensive approach will help uncover valuable insights into product popularity, sales trends, and customer segmentation, providing a holistic understanding of the dataset.

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import requests
import plotly.express as px
import missingno as msno
import warnings
import scipy.stats as stats


from io import StringIO
from sklearn.preprocessing import LabelEncoder
from datetime import datetime
from tqdm import tqdm


label_encoder = LabelEncoder()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
warnings.filterwarnings('ignore')
```

In [3]:
```
data1=pd.read_csv('Online Retail.csv')
data2=pd.read_csv('online_retail_II.csv')
```

In [4]:
```
display(data1.head())
```

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

In [5]:
```
display(data2.head())
```

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

Understanding the data

In [6]:
```
print(f'''Data1 Column names
{data1.columns}
-------------------------
Data2 Column names:
{data2.columns}''')
```

```
Data1 Column names
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
-------------------------
Data2 Column names:
Index(['Invoice', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'Price', 'Customer ID', 'Country'],
      dtype='object')
```

In [7]:
```
# Renaming Column names
column_mapping = {
    'Invoice': 'InvoiceNo',
    'Price': 'UnitPrice',
    'Customer ID': 'CustomerID',
}

data2.rename(columns=column_mapping, inplace=True)

print(data2.columns)
```

```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

In [8]:
```
data = pd.merge(data1, data2, how='outer')
print(f'''Data1 Shape:{data1.shape},
Data2 Shape:{data2.shape},
Data Shape:{data.shape}''')
```

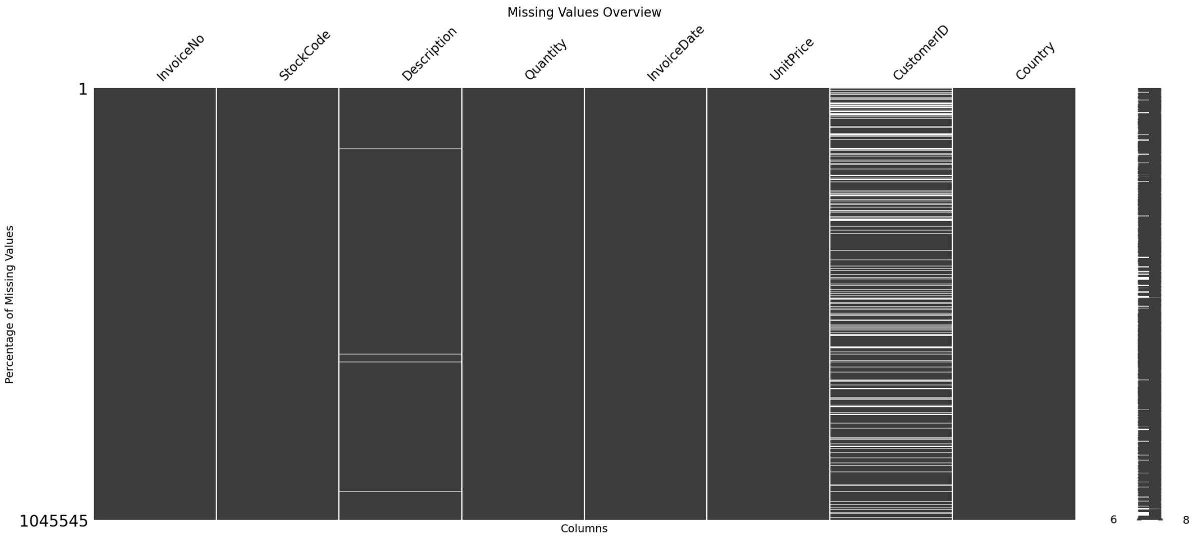
```
Data1 Shape:(541909, 8),
Data2 Shape:(525461, 8),
Data Shape:(1045545, 8)
```

In [9]:
```
def table_summary(data):
    summary_data = []

    for column in data.columns:
        num_unique_values = data[column].nunique()
        num_empty_values = data[column].isnull().sum()
        data_type = data[column].dtypes
        percentage_empty = (num_empty_values / len(data)) * 100
        summary_data.append([column, num_unique_values, num_empty_values, pe

    summary_df = pd.DataFrame(summary_data, columns=['Column Name', 'Unique

    print(f'''Table Summary
Dataframe Shape: {data.shape}''')


    display(summary_df)
    msno.matrix(data)
    plt.title('Missing Values Overview', fontsize=16)
    plt.xlabel('Columns', fontsize=14)
    plt.ylabel('Percentage of Missing Values', fontsize=14)
    plt.show()
```

In [10]:
```
table_summary(data)
```

```
Table Summary
Dataframe Shape: (1045545, 8)
```

	Column Name	Unique Values	Empty Values	Percentage Empty	Data Type
0	InvoiceNo	53628	0	0.000000	object
1	StockCode	5305	0	0.000000	object
2	Description	5698	4275	0.408878	object
3	Quantity	1057	0	0.000000	int64
4	InvoiceDate	47635	0	0.000000	object
5	UnitPrice	2807	0	0.000000	float64
6	CustomerID	5942	235289	22.503957	float64
7	Country	43	0	0.000000	object



Data Cleaning

We have missing values in the `Description` column. Initially, I'll examine whether these missing values are associated with valid `StockCode` entries and determine if there are corresponding descriptions for those `StockCode` entries.

In [11]:
```
desc_nan=data[data['Description'].isnull()]
display(desc_nan.head())
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
658	536414	22139	NaN	56	12/1/2010 11:52	0.0	NaN	U Kir
2046	536545	21134	NaN	1	12/1/2010 14:32	0.0	NaN	U Kir
2047	536546	22145	NaN	1	12/1/2010 14:33	0.0	NaN	U Kir
2048	536547	37509	NaN	1	12/1/2010 14:33	0.0	NaN	U Kir
2063	536549	85226A	NaN	1	12/1/2010 14:34	0.0	NaN	U Kir

In [12]:
```
# getting the list of unique StockCode from desc_nan dataset
stockcodes_list = list(desc_nan['StockCode'].unique())
```

```
# filtering the 'data' containing only the StockCode from stockcodes_list
mask = data['StockCode'].isin(stockcodes_list)
filtered_data = data[mask]

display(filtered_data.head())
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Countı
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	Unite Kingdo
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	Unite Kingdo
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	Unite Kingdo
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850.0	Unite Kingdo
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850.0	Unite Kingdo

In [13]:
```
filtered_data.shape
```

Out[13]:
```
(375540, 8)
```

Fill missing values in the 'Description' column by replacing them with the available descriptions associated with the respective 'StockCode' if such descriptions exist.

In [14]:
```
# Identify unique StockCodes where Description is NaN
missing_description_stockcodes = desc_nan['StockCode'].unique()

# go through each StockCodes
for stockcode in tqdm(missing_description_stockcodes):
    # Check if there are non-null Descriptions for the current StockCode
    non_null_descriptions = data[data['StockCode'] == stockcode]['Descripti

    if not non_null_descriptions.empty:
        # If non-null Descriptions exist, fill in missing values with the fi
        fill_value = non_null_descriptions.iloc[0]
        condition = (data['StockCode'] == stockcode) & (data['Description']
        data.loc[condition, 'Description'] = fill_value
```

```
100%|████████████████████████████████████████| 2451/2451 [03:04<00:00, 13.32
it/s]
```

In [15]:
```
def count_unique_values(data, columns):
    summary_data = []
    for column in columns:
```

```
        num_unique_values = data[column].nunique()
        summary_data.append([column, num_unique_values])
    summary_df = pd.DataFrame(summary_data, columns=['Column Name', 'Unique
    display(summary_df)
count_unique_values(data,['StockCode','Description'])
```

	Column Name	Unique Values
0	StockCode	5305
1	Description	5698

Counting the number of `Description` each `StockCode` has.

In [16]:
```
# Group by 'StockCode' and count the unique 'Description' values
stockcode_description_counts = data.groupby('StockCode')['Description'].nun
# display(stockcode_description_counts)

# Filter StockCodes with more than one unique Description
stockcodes_with_multiple_descriptions = stockcode_description_counts[stockc
# display(stockcodes_with_multiple_descriptions)
```

In [17]:
```
# Filter rows with StockCodes having more than one unique Description
multiple_description_stockcodes = stockcode_description_counts[stockcode_de

# Create a df containing StockCodes with multiple Descriptions
multiple_description_data = data[data['StockCode'].isin(multiple_descriptio

# Group by 'StockCode' and collect unique 'Description' values
stockcode_descriptions_multiple = multiple_description_data.groupby('StockC

# Display the result
display(stockcode_descriptions_multiple.head())
```

	StockCode	Description
0	10080	[GROOVY CACTUS INFLATABLE, check]
1	10120	[DOGGY RUBBER, Zebra invcing error]
2	10133	[COLOURING PENCILS BROWN TUBE, damaged]
3	15056N	[EDWARDIAN PARASOL NATURAL, wedding co returns?]
4	15058A	[BLUE POLKADOT GARDEN PARASOL, wet/rusty, BLUE...

In [18]:
```
# Iterate through StockCodes with multiple Descriptions
for stockcode in tqdm(stockcodes_with_multiple_descriptions):
    # Get the first Description for the current StockCode
    first_description = data[data['StockCode'] == stockcode]['Description']

    # Replace all Descriptions for the current StockCode with the first one
    data.loc[data['StockCode'] == stockcode, 'Description'] = first_descrip
```

```
100%|██████████████████████████████████████| 1232/1232 [01:20<00:00, 15.26
it/s]
```

In [19]:
```
count_unique_values(data,['StockCode','Description'])
```

	Column Name	Unique Values
0	StockCode	5305
1	Description	4743

In [20]:
```
# Filter rows with StockCodes having more than one unique Description
multiple_description_stockcodes = stockcode_description_counts[stockcode_des

# Create a DataFrame containing StockCodes with multiple Descriptions
multiple_description_data = data[data['StockCode'].isin(multiple_descriptio

# Group by 'StockCode' and collect unique 'Description' values
stockcode_descriptions_multiple = multiple_description_data.groupby('StockC

# Display the result
display(stockcode_descriptions_multiple.head())
```

	StockCode	Description
0	10080	[GROOVY CACTUS INFLATABLE]
1	10120	[DOGGY RUBBER]
2	10133	[COLOURING PENCILS BROWN TUBE]
3	15056N	[EDWARDIAN PARASOL NATURAL]
4	15058A	[BLUE POLKADOT GARDEN PARASOL]

In [21]:
```
count_unique_values(data,['StockCode','Description'])
```

	Column Name	Unique Values
0	StockCode	5305
1	Description	4743

Now there are `Description` with Multiple `StockCode` , Need to Clean this

In [22]:
```
# Group by 'Description' and collect unique 'StockCode' values
description_stockcodes_multiple = data.groupby('Description')['StockCode'].u

# Filter rows where there are multiple unique StockCodes for a Description
description_stockcodes_multiple = description_stockcodes_multiple[descripti

# Display the result
display(description_stockcodes_multiple.head())
```

	Description	StockCode
49	3 GARDENIA MORRIS BOXED CANDLES	[85034A, 85034a]
63	3 WHITE CHOC MORRIS BOXED CANDLES	[85034B, 85034b]
74	3D DOG PICTURE PLAYING CARDS	[84558A, 84558a]
76	3D SHEET OF CAT STICKERS	[84559B, 84559b]
77	3D SHEET OF DOG STICKERS	[84559A, 84559a]

In [23]:
```
# Iterate through rows with multiple StockCodes for a Description
for index, row in description_stockcodes_multiple.iterrows():
```

```
    # Get the first StockCode for the current Description
    first_stockcode = row['StockCode'][0]

    # Replace all StockCodes for the current Description with the first one
    data.loc[data['Description'] == row['Description'], 'StockCode'] = firs
```

In [24]:
```
# Display the result
description_stockcodes_multiple = data.groupby('Description')['StockCode'].u
display(description_stockcodes_multiple.head())
```

	Description	StockCode
0	4 PURPLE FLOCK DINNER CANDLES	[72800B]
1	50'S CHRISTMAS GIFT BAG LARGE	[23437]
2	DOLLY GIRL BEAKER	[23345]
3	HOME SWEET HOME BLACKBOARD	[21185]
4	I LOVE LONDON MINI BACKPACK	[23391]

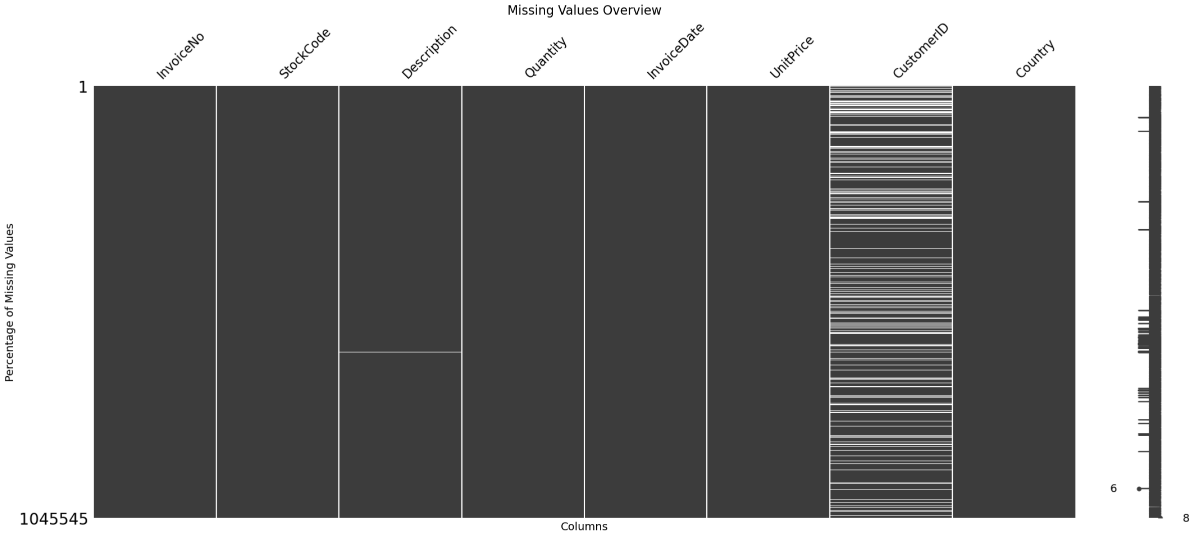
In [25]:
```
count_unique_values(data,['StockCode','Description'])
```

	Column Name	Unique Values
0	StockCode	5098
1	Description	4743

In [26]:
```
table_summary(data)
```

```
Table Summary
Dataframe Shape: (1045545, 8)
```

	Column Name	Unique Values	Empty Values	Percentage Empty	Data Type
0	InvoiceNo	53628	0	0.000000	object
1	StockCode	5098	0	0.000000	object
2	Description	4743	363	0.034719	object
3	Quantity	1057	0	0.000000	int64
4	InvoiceDate	47635	0	0.000000	object
5	UnitPrice	2807	0	0.000000	float64
6	CustomerID	5942	235289	22.503957	float64
7	Country	43	0	0.000000	object



There are still Missing values in Description, So i'll be dropping them

In [27]:
```
data = data.dropna(subset=['Description'])
```

In [28]:
```
data['Description'] = data['Description'].str.title()
```

In [29]:
```
data.shape
```

Out[29]:
```
(1045182, 8)
```

Preprocessing

In [30]:
```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])

# Extract Year, Quarter, Month, Week, and Day
data['Year'] = data['InvoiceDate'].dt.year
data['Quarter'] = data['InvoiceDate'].dt.quarter
data['Month'] = data['InvoiceDate'].dt.month
data['Week'] = data['InvoiceDate'].dt.isocalendar().week
data['Day'] = data['InvoiceDate'].dt.day
data['TotalPrice']=data['Quantity']*data['UnitPrice']
data['YearQuarter'] = data['InvoiceDate'].dt.to_period("Q")
data['YearMonth'] = data['InvoiceDate'].dt.to_period("M")
data['Date'] = data['InvoiceDate'].dt.date
data['DayOfWeek'] = data['InvoiceDate'].dt.day_name()
```

In [31]:
```
total_duplicates = data.duplicated(keep=False).sum()

# Display the total number of duplicate rows
print("Total Number of Duplicates:", total_duplicates)
```

```
Total Number of Duplicates: 24920
```

In [32]:
```
data=data.drop_duplicates(keep='first')
```

Although there are approximately 22% missing values in the `CustomerID` field, it's important to note that these entries contribute to the generated income. Therefore, I will retain these values for my analysis

In [33]:
```
# For analysis purposes, I'm filling the null values in the 'CustomerID' co
data['CustomerID'] = data['CustomerID'].fillna('00000')
```

```
data['CustomerID'] = data['CustomerID'].astype('int64')
```

In [34]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1031929 entries, 0 to 1045544
Data columns (total 18 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   InvoiceNo    1031929 non-null  object
 1   StockCode    1031929 non-null  object
 2   Description  1031929 non-null  object
 3   Quantity     1031929 non-null  int64
 4   InvoiceDate  1031929 non-null  datetime64[ns]
 5   UnitPrice    1031929 non-null  float64
 6   CustomerID   1031929 non-null  int64
 7   Country      1031929 non-null  object
 8   Year         1031929 non-null  int32
 9   Quarter      1031929 non-null  int32
 10  Month        1031929 non-null  int32
 11  Week         1031929 non-null  UInt32
 12  Day          1031929 non-null  int32
 13  TotalPrice   1031929 non-null  float64
 14  YearQuarter  1031929 non-null  period[Q-DEC]
 15  YearMonth    1031929 non-null  period[M]
 16  Date         1031929 non-null  object
 17  DayOfWeek    1031929 non-null  object
dtypes: UInt32(1), datetime64[ns](1), float64(2), int32(4), int64(2), objec
t(6), period[M](1), period[Q-DEC](1)
memory usage: 130.9+ MB
```

In [35]:
```
data.describe(include='all')
```

Out[35]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPr
count	1031929	1031929	1031929	1.031929e+06	1031929	1.031929e+
unique	53265	4743	4740	NaN	NaN	N
top	573585	85123A	White Hanging Heart T-Light Holder	NaN	NaN	N
freq	1114	5752	5752	NaN	NaN	N
mean	NaN	NaN	NaN	1.009616e+01	2011-01-03 16:20:29.697507328	4.616598e+
min	NaN	NaN	NaN	-8.099500e+04	2009-12-01 07:45:00	-5.359436e+
25%	NaN	NaN	NaN	1.000000e+00	2010-07-05 12:40:00	1.250000e+
50%	NaN	NaN	NaN	3.000000e+00	2010-12-09 13:34:00	2.100000e+
75%	NaN	NaN	NaN	1.000000e+01	2011-07-27 13:15:00	4.150000e+
max	NaN	NaN	NaN	8.099500e+04	2011-12-09 12:50:00	3.897000e+
std	NaN	NaN	NaN	1.751800e+02	NaN	1.224631e+

In [36]:
```
negative_quantity=data[data['Quantity']<0]
```

In [37]:
```
negative_quantity.head()
```

Out[37]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
141	C536379	D	Discount	-1	2010-12-01 09:41:00	27.50	14527	Ur King
154	C536383	35004C	Set Of 3 Coloured Flying Ducks	-1	2010-12-01 09:49:00	4.65	15311	Ur King
235	C536391	22556	Plasters In Tin Circus Parade	-12	2010-12-01 10:24:00	1.65	17548	Ur King
236	C536391	21984	Pack Of 12 Pink Paisley Tissues	-24	2010-12-01 10:24:00	0.29	17548	Ur King
237	C536391	21983	Pack Of 12 Blue Paisley Tissues	-24	2010-12-01 10:24:00	0.29	17548	Ur King

In [38]:
```
negative_quantity.shape
```

Out[38]:
```
(22165, 18)
```

There are over 20k observations with negative quantities. Most of the negative values are Cancelled orders. However, there are values where InvoiceNo number does not contain `'C'`

In [39]:
```
cancelled=data[data['InvoiceNo'].astype(str).str.contains('C')]
filtered_data = data[~data['InvoiceNo'].astype(str).str.contains('C')]
```

In [40]:
```
display(cancelled.head())
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
141	C536379	D	Discount	-1	2010-12-01 09:41:00	27.50	14527	Ur King
154	C536383	35004C	Set Of 3 Coloured Flying Ducks	-1	2010-12-01 09:49:00	4.65	15311	Ur King
235	C536391	22556	Plasters In Tin Circus Parade	-12	2010-12-01 10:24:00	1.65	17548	Ur King
236	C536391	21984	Pack Of 12 Pink Paisley Tissues	-24	2010-12-01 10:24:00	0.29	17548	Ur King
237	C536391	21983	Pack Of 12 Blue Paisley Tissues	-24	2010-12-01 10:24:00	0.29	17548	Ur King

In [41]:
```
cancelled.shape
```

Out[41]:
```
(19076, 18)
```

In [42]:
```
cancelled['Quantity'].max()
```
Out[42]:
```
1
```

In [43]:
```
cancelled[cancelled['Quantity']>0]
```
Out[43]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
619406	C496350	M	Manual	1	2010-02-01 08:24:00	373.57	0

In [44]:
```
filtered_data[filtered_data['Quantity']<0].head()
```
Out[44]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
2498	536589	21777	Recipe Box With Metal Heart	-10	2010-12-01 16:50:00	0.0	0	U Kir
4477	536764	84952C	Mirror Love Bird T-Light Holder	-38	2010-12-02 14:42:00	0.0	0	U Kir
7404	536996	22712	Card Dolly Girl	-20	2010-12-03 15:30:00	0.0	0	U Kir
7405	536997	22028	Penny Farthing Birthday Card	-20	2010-12-03 15:30:00	0.0	0	U Kir
7406	536998	85067	Cream Sweetheart Wall Cabinet	-6	2010-12-03 15:30:00	0.0	0	U Kir

In [45]:
```
filtered_data.shape
```
Out[45]:
```
(1012853, 18)
```

In [46]:
```
filtered_data['UnitPrice'].nunique()
```
Out[46]:
```
2280
```

In [47]:
```
filtered_data['InvoiceNo'].nunique()
```
Out[47]:
```
44973
```

In [48]:
```
filtered_data.describe(include='all')
```

Out[48]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPı
count	1012853	1012853	1012853	1.012853e+06	1012853	1.012853e·
unique	44973	4737	4734	NaN	NaN	ľ
top	573585	85123A	White Hanging Heart T-Light Holder	NaN	NaN	ľ
freq	1114	5618	5618	NaN	NaN	ľ
mean	NaN	NaN	NaN	1.075685e+01	2011-01-04 02:40:45.063932672	3.895506e·
min	NaN	NaN	NaN	-9.600000e+03	2009-12-01 07:45:00	-5.359436e·
25%	NaN	NaN	NaN	1.000000e+00	2010-07-05 16:48:00	1.250000e·
50%	NaN	NaN	NaN	3.000000e+00	2010-12-09 14:09:00	2.100000e·
75%	NaN	NaN	NaN	1.200000e+01	2011-07-27 15:16:00	4.130000e·
max	NaN	NaN	NaN	8.099500e+04	2011-12-09 12:50:00	2.511109e·
std	NaN	NaN	NaN	1.373152e+02	NaN	9.497313e

We can see there are some outliers in the TotalPrice, Quantity and UnitPrice column

In [49]:
```
fig = px.box(filtered_data, x='TotalPrice')
fig.show()
```

In [50]:
```
fig = px.box(filtered_data, x='Quantity')
fig.show()
```

In [51]:
```
fig = px.box(filtered_data, x='UnitPrice')
fig.show()
```

In [52]:

```
# Function to remove outliers based on IQR
def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return data[(data[column] >= lower_bound) & (data[column] <= upper_boun

# Remove outliers for each relevant column
no_outliers = remove_outliers_iqr(filtered_data, 'Quantity')
no_outliers = remove_outliers_iqr(filtered_data, 'UnitPrice')
no_outliers = remove_outliers_iqr(filtered_data, 'TotalPrice')

# Display the resulting DataFrame
display(no_outliers.head())
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Countr
0	536365	85123A	White Hanging Heart T-Light Holder	6	2010-12-01 08:26:00	2.55	17850	Unite Kingdo
1	536365	71053	White Metal Lantern	6	2010-12-01 08:26:00	3.39	17850	Unite Kingdo
2	536365	84406B	Cream Cupid Hearts Coat Hanger	8	2010-12-01 08:26:00	2.75	17850	Unite Kingdo
3	536365	84029G	Knitted Union Flag Hot Water Bottle	6	2010-12-01 08:26:00	3.39	17850	Unite Kingdo
4	536365	84029E	Red Woolly Hottie White Heart.	6	2010-12-01 08:26:00	3.39	17850	Unite Kingdo

In [53]:
```
no_outliers.shape
```
Out[53]:
```
(929013, 18)
```

In [54]:
```
df=no_outliers[(no_outliers['Quantity']>0) & (~no_outliers['InvoiceNo'].asty
```

In [55]:
```
df.head()
```
Out[55]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Countr
0	536365	85123A	White Hanging Heart T-Light Holder	6	2010-12-01 08:26:00	2.55	17850	Unite Kingdo
1	536365	71053	White Metal Lantern	6	2010-12-01 08:26:00	3.39	17850	Unite Kingdo
2	536365	84406B	Cream Cupid Hearts Coat Hanger	8	2010-12-01 08:26:00	2.75	17850	Unite Kingdo
3	536365	84029G	Knitted Union Flag Hot Water Bottle	6	2010-12-01 08:26:00	3.39	17850	Unite Kingdo
4	536365	84029E	Red Woolly Hottie White Heart.	6	2010-12-01 08:26:00	3.39	17850	Unite Kingdo

In [56]:
```
df.shape
```
Out[56]:
```
(925923, 18)
```

In [57]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 925923 entries, 0 to 1045542
Data columns (total 18 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    925923 non-null  object
 1   StockCode    925923 non-null  object
 2   Description  925923 non-null  object
 3   Quantity     925923 non-null  int64
 4   InvoiceDate  925923 non-null  datetime64[ns]
 5   UnitPrice    925923 non-null  float64
 6   CustomerID   925923 non-null  int64
 7   Country      925923 non-null  object
 8   Year         925923 non-null  int32
 9   Quarter      925923 non-null  int32
 10  Month        925923 non-null  int32
 11  Week         925923 non-null  UInt32
 12  Day          925923 non-null  int32
 13  TotalPrice   925923 non-null  float64
 14  YearQuarter  925923 non-null  period[Q-DEC]
 15  YearMonth    925923 non-null  period[M]
 16  Date         925923 non-null  object
 17  DayOfWeek    925923 non-null  object
dtypes: UInt32(1), datetime64[ns](1), float64(2), int32(4), int64(2), objec
t(6), period[M](1), period[Q-DEC](1)
memory usage: 117.4+ MB
```

In [58]:
```
df.describe(include='all')
```

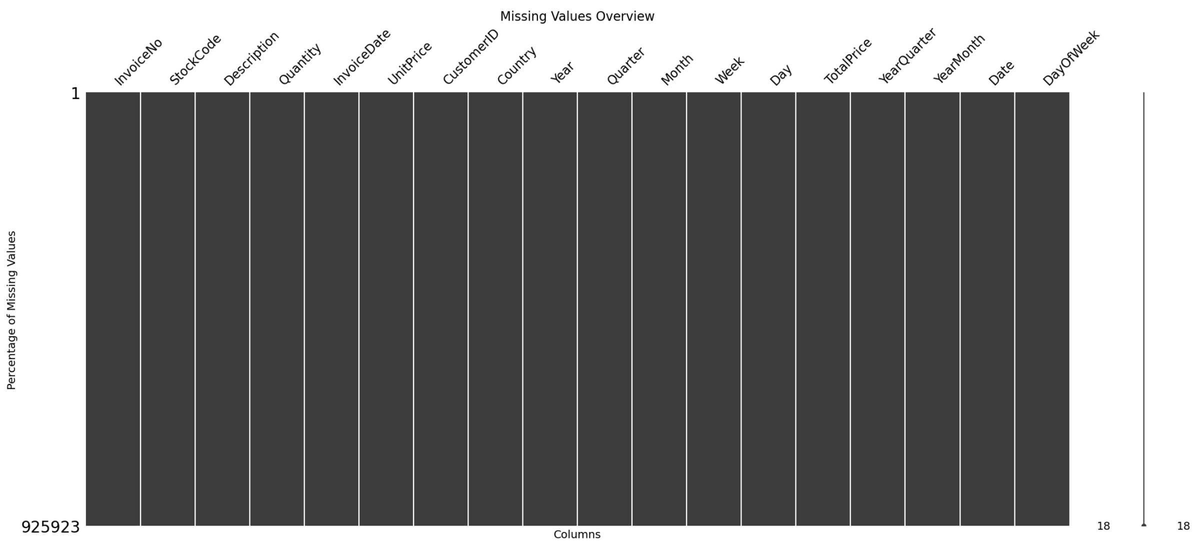
Out[58]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitP
count	925923	925923	925923	925923.000000	925923	925923.000
unique	37844	4694	4693	NaN	NaN	I
top	573585	85123A	White Hanging Heart T-Light Holder	NaN	NaN	I
freq	1043	4311	4311	NaN	NaN	I
mean	NaN	NaN	NaN	6.511160	2011-01-04 20:19:20.204769024	3.189
min	NaN	NaN	NaN	1.000000	2009-12-01 07:45:00	0.000
25%	NaN	NaN	NaN	1.000000	2010-07-06 13:13:00	1.250
50%	NaN	NaN	NaN	3.000000	2010-12-09 20:01:00	2.100
75%	NaN	NaN	NaN	9.000000	2011-07-28 15:31:00	4.130
max	NaN	NaN	NaN	12540.000000	2011-12-09 12:50:00	38.340
std	NaN	NaN	NaN	31.263738	NaN	3.316

In [59]:
```
table_summary(df)
```

```
Table Summary
Dataframe Shape: (925923, 18)
```

	Column Name	Unique Values	Empty Values	Percentage Empty	Data Type
0	InvoiceNo	37844	0	0.0	object
1	StockCode	4694	0	0.0	object
2	Description	4693	0	0.0	object
3	Quantity	242	0	0.0	int64
4	InvoiceDate	34942	0	0.0	datetime64[ns]
5	UnitPrice	618	0	0.0	float64
6	CustomerID	5669	0	0.0	int64
7	Country	43	0	0.0	object
8	Year	3	0	0.0	int32
9	Quarter	4	0	0.0	int32
10	Month	12	0	0.0	int32
11	Week	52	0	0.0	UInt32
12	Day	31	0	0.0	int32
13	TotalPrice	1762	0	0.0	float64
14	YearQuarter	9	0	0.0	period[Q-DEC]
15	YearMonth	25	0	0.0	period[M]
16	Date	604	0	0.0	object
17	DayOfWeek	7	0	0.0	object



EDA

In [60]:
```
# How many orders have been placed in total
print(f"There are {df['InvoiceNo'].nunique()} Orders in this Dataset")
```

```
There are 37844 Orders in this Dataset
```

In [61]:
```
# Find the number of items per order
product_counts = data.groupby('InvoiceNo')['StockCode'].count().sort_values
display(product_counts.head())
```

	InvoiceNo	StockCode
0	573585	1114
1	581219	748
2	581492	731
3	580729	721
4	558475	705

In [62]:
```
# Finding the order size
avg_size_per_invoice = df.groupby('InvoiceNo')['StockCode'].count().mean()
print(f'Avg Basket Size per order is {avg_size_per_invoice}')
```

```
Avg Basket Size per order is 24.466837543600043
```

In [63]:
```
# Finding the totalCost per order
totalCost_counts = df.groupby('InvoiceNo')['TotalPrice'].sum().reset_index(
display(totalCost_counts.head())
```

	InvoiceNo	TotalPrice
0	489434	60.00
1	489435	61.20
2	489436	260.69
3	489437	310.75
4	489438	61.94

In [64]:
```
# Finding the avg totalCost per order
avg_cost_per_invoice = df.groupby('InvoiceNo')['TotalPrice'].sum().mean()
print(f'Avg Cost per order is {avg_cost_per_invoice}')
```

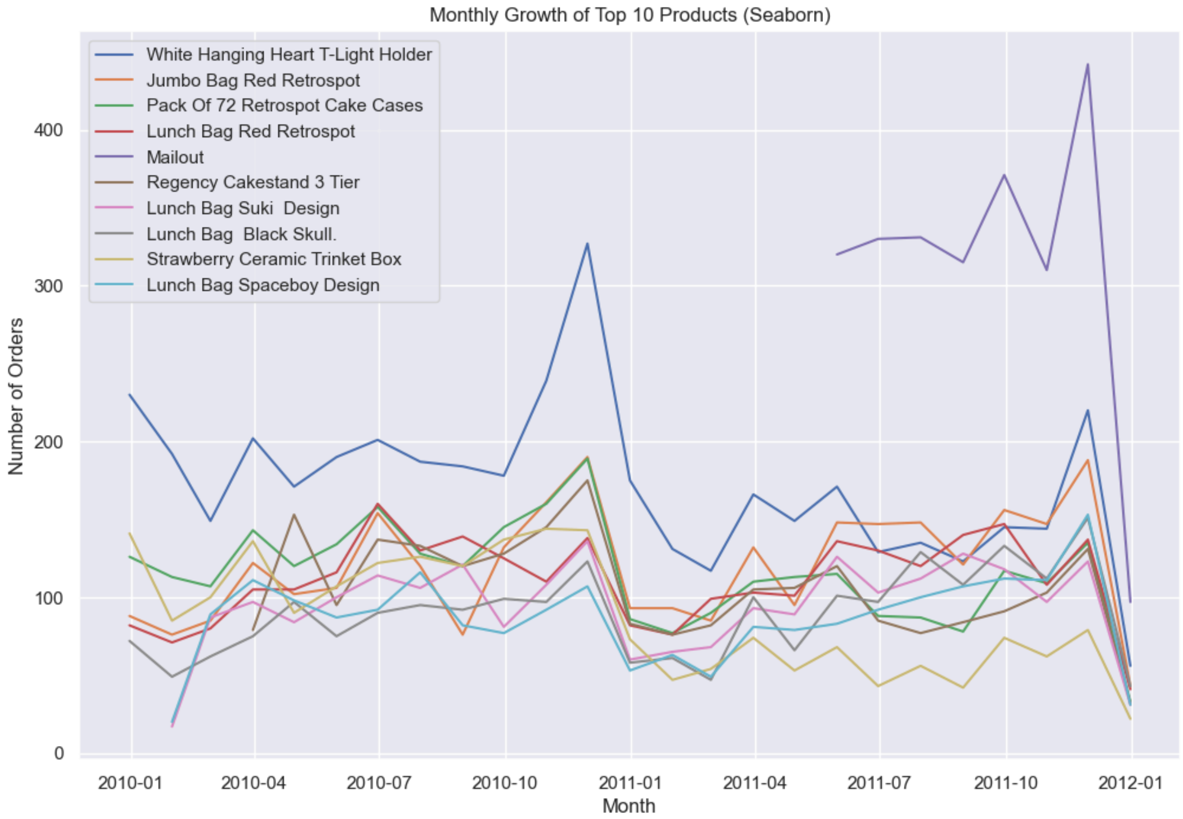
```
Avg Cost per order is 260.2887889229468
```

In [65]:
```
# Finding top 10 products
top_10=df.groupby(['StockCode','Description']).agg({'InvoiceNo':'count'}).s
top_10.head(10)
```

Out[65]:

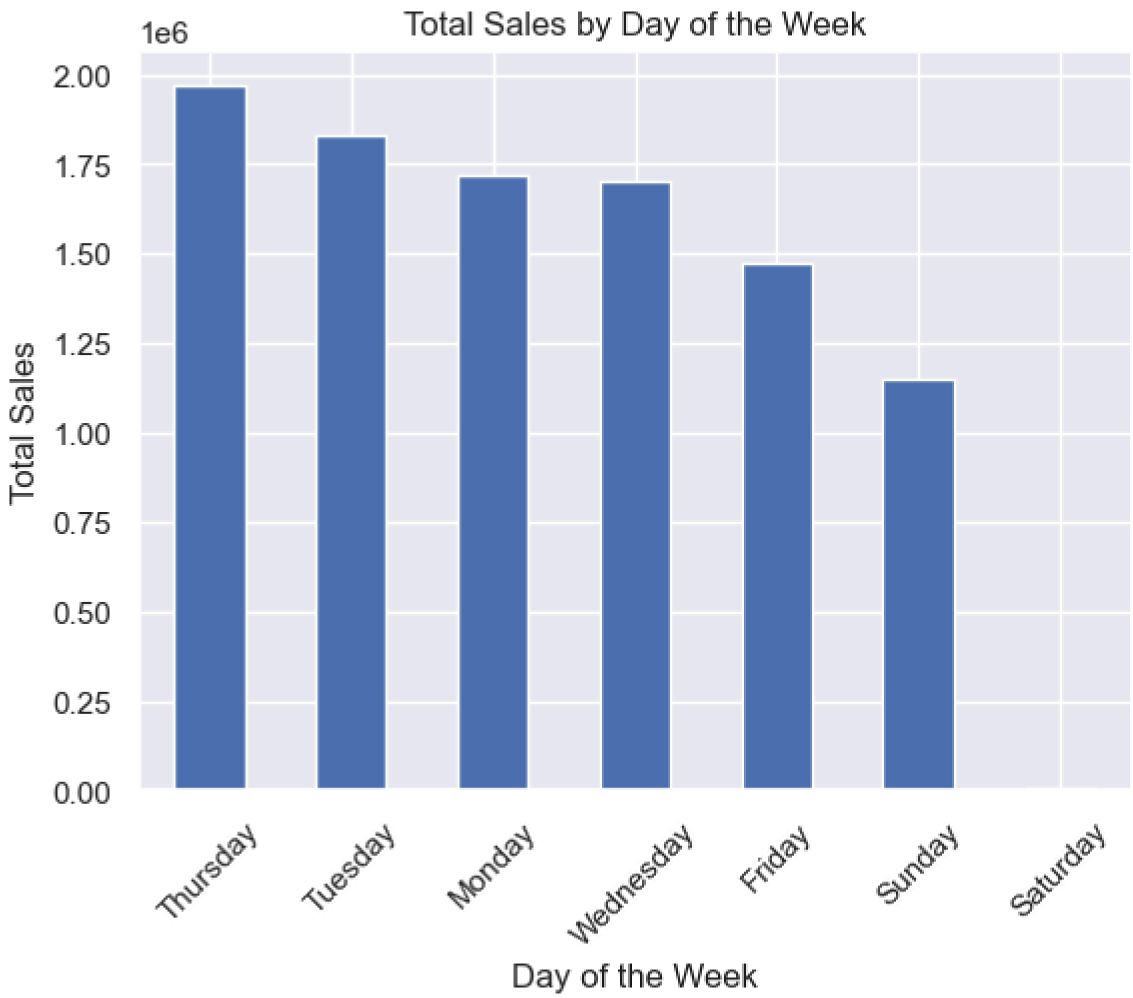
		InvoiceNo
StockCode	**Description**	
85123A	**White Hanging Heart T-Light Holder**	4311
85099B	**Jumbo Bag Red Retrospot**	3008
21212	**Pack Of 72 Retrospot Cake Cases**	2881
20725	**Lunch Bag Red Retrospot**	2781
23202	**Mailout**	2516
22423	**Regency Cakestand 3 Tier**	2340
22383	**Lunch Bag Suki Design**	2264
20727	**Lunch Bag Black Skull.**	2232
21232	**Strawberry Ceramic Trinket Box**	2198
22382	**Lunch Bag Spaceboy Design**	2085

In [66]:
```
# Top 10 Products over time

# Group by 'Description' and 'InvoiceDate', count 'InvoiceNo' for each montl
monthly_counts = df.groupby(['Description', pd.Grouper(key='InvoiceDate', fı

# Filter data for the top 10 products based on Description
top_products = df.groupby('Description')['InvoiceNo'].count().sort_values(as

# Set the plotting style
sns.set(style="darkgrid")

# Plotting individual growth for each of the top 10 products using Seaborn
plt.figure(figsize=(12, 8))
for product in top_products:
    product_data = monthly_counts[monthly_counts['Description'] == product]
    sns.lineplot(x=product_data['InvoiceDate'], y=product_data['InvoiceNo'],

plt.xlabel('Month')
plt.ylabel('Number of Orders')
plt.title('Monthly Growth of Top 10 Products (Seaborn)')
plt.legend()
plt.show()
```



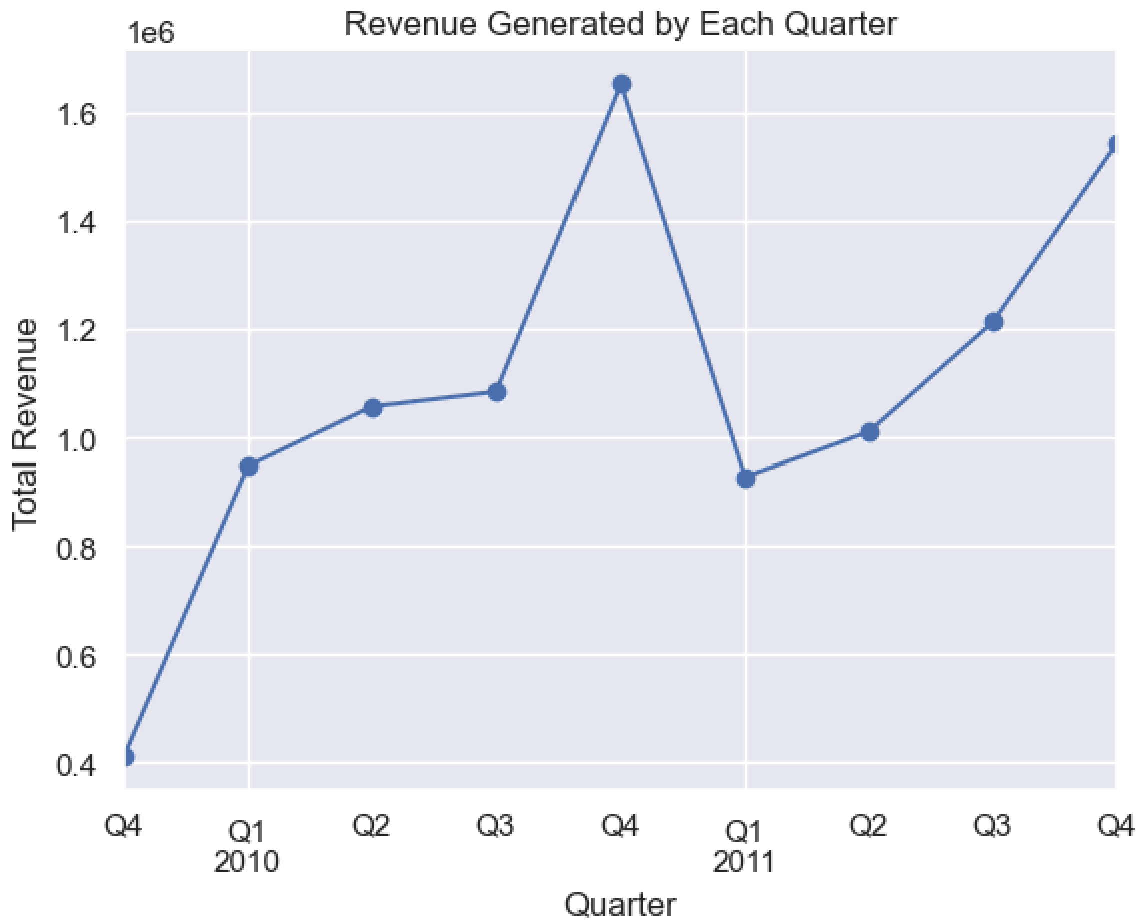
In [67]:
```
# Fing the day with the highest purchase

# Group by day of the week, then calculate total sales and sort
total_cost_per_day = df.groupby(['DayOfWeek'])['TotalPrice'].sum().reset_inc
total_cost_per_day = total_cost_per_day.sort_values(by='TotalPrice', ascendi

# Plot the bar chart
ax = total_cost_per_day.plot(kind='bar', x='DayOfWeek', y='TotalPrice', lege

# Set title, x-axis label, and y-axis label
plt.title('Total Sales by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Total Sales')
```

```
# Display the plot
plt.show()
```



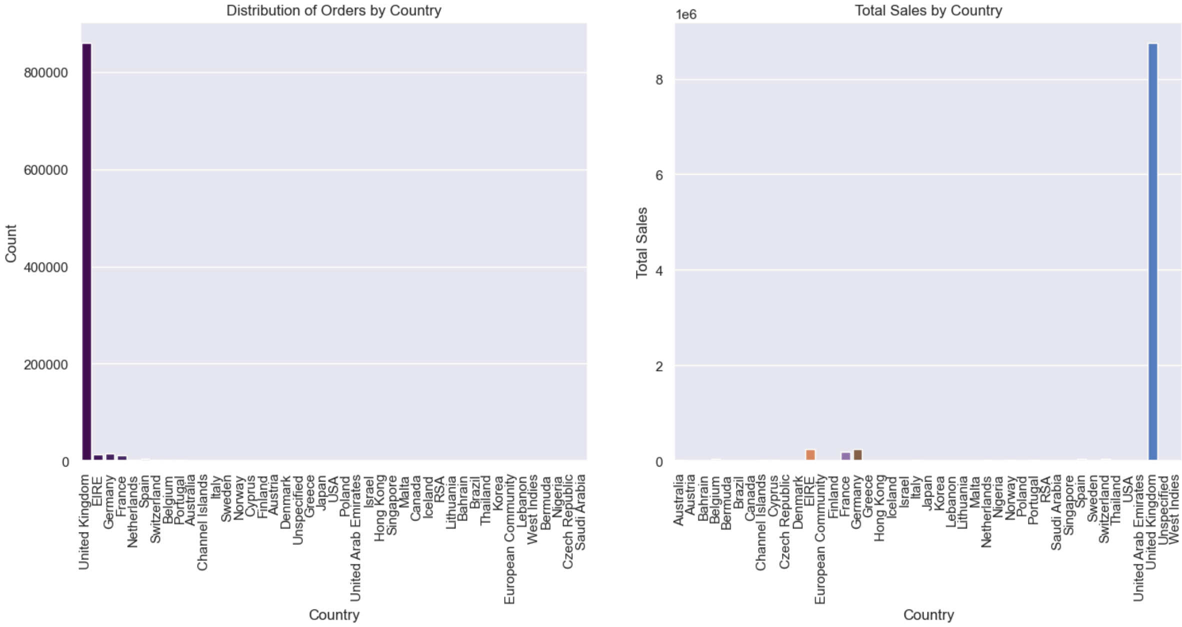
In [68]:

```
# Revenue Generated by Each Quarter

# Assuming 'YearQuarter' is the column representing the quarter
revenue_by_quarter = df.groupby('YearQuarter')['TotalPrice'].sum().reset_in

# Plot the line chart
ax = revenue_by_quarter.plot(kind='line', x='YearQuarter', y='TotalPrice', 

# Set title, x-axis label, and y-axis label
plt.title('Revenue Generated by Each Quarter')
plt.xlabel('Quarter')
plt.ylabel('Total Revenue')

# Display the plot
plt.show()
```



In [69]:
```
# Total Amount by each month

shopping_by_month = df.groupby('Month')['TotalPrice'].sum().reset_index()

# Plot the bar chart
plt.bar(shopping_by_month['Month'], shopping_by_month['TotalPrice'])

# Set title, x-axis label, and y-axis label
plt.title('Total Shopping Amount by Month')
plt.xlabel('Month')
plt.ylabel('Total Shopping Amount')

# Display the plot
plt.show()
```

In [70]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))

# Plot 1: Country distribution
sns.countplot(x='Country', data=df, ax=axes[0], order=data['Country'].value_
axes[0].set_title('Distribution of Orders by Country')
axes[0].set_xlabel('Country')
axes[0].set_ylabel('Count')

# Plot 2: TotalPrice by Country
sns.barplot(x='Country', y='TotalPrice', data=df.groupby('Country')['TotalPr
axes[1].set_title('Total Sales by Country')
axes[1].set_xlabel('Country')
axes[1].set_ylabel('Total Sales')

# Adjust layout to prevent overlapping
plt.tight_layout()

# # Rotate x-axis labels for better readability
axes[0].tick_params(axis='x', rotation=90)
axes[1].tick_params(axis='x', rotation=90)

# Display the subplots
plt.show()
```



In [71]:
```
# TotalPrice per Country
total_sales_by_country = df.groupby('Country')['TotalPrice'].sum().sort_valu
display(total_sales_by_country)
```

```
Country
United Kingdom          8748184.717
EIRE                     244410.280
Germany                  240889.031
France                   193716.510
Spain                     48184.870
Belgium                   47454.920
Switzerland               42537.230
Portugal                  32624.350
Netherlands               27601.790
Channel Islands           24706.300
Italy                     22456.440
Norway                    17948.820
Australia                 16014.340
Cyprus                    15062.320
Finland                   14758.860
Sweden                    13523.670
Austria                   13351.850
Unspecified                8967.890
Denmark                    8907.660
Greece                     8721.230
Poland                     8124.220
United Arab Emirates       7053.670
USA                        6153.740
Israel                     4865.430
Hong Kong                  4415.940
Singapore                  4111.970
Malta                      4036.450
Iceland                    3667.950
Canada                     3425.380
RSA                        2189.790
Japan                      2183.870
Lithuania                  2121.860
Bahrain                    1841.510
Thailand                   1113.700
European Community         1012.250
Korea                       870.110
Lebanon                     754.880
Brazil                      684.610
Bermuda                     602.800
West Indies                 489.610
Czech Republic              339.800
Saudi Arabia                145.920
Nigeria                     140.390
Name: TotalPrice, dtype: float64
```

In [72]:

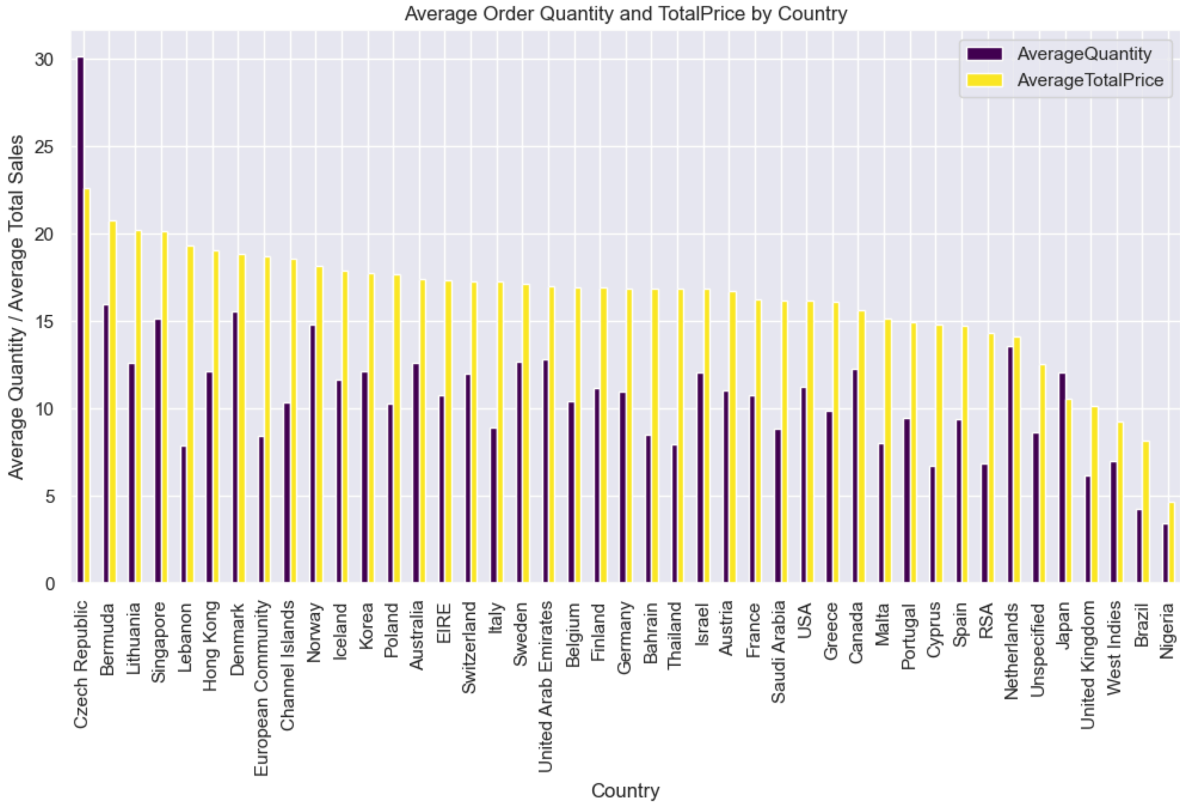
```python
average_order_by_country = df.groupby('Country').agg({'Quantity': 'mean', 'T
average_order_by_country.rename(columns={'Quantity': 'AverageQuantity', 'To
display(average_order_by_country)
```

Country	AverageQuantity	AverageTotalPrice
Czech Republic	30.133333	22.653333
Bermuda	16.000000	20.786207
Lithuania	12.609524	20.208190
Singapore	15.181373	20.156716
Lebanon	7.897436	19.355897
Hong Kong	12.150862	19.034224
Denmark	15.577167	18.832262
European Community	8.444444	18.745370
Channel Islands	10.388889	18.548273
Norway	14.784195	18.185228
Iceland	11.692683	17.892439
Korea	12.163265	17.757347
Poland	10.313725	17.699826
Australia	12.589771	17.425832
EIRE	10.778258	17.331604
Switzerland	12.036917	17.256483
Italy	8.927803	17.247650
Sweden	12.701266	17.118570
United Arab Emirates	12.833333	17.037850
Belgium	10.425813	16.966364
Finland	11.186712	16.905911
Germany	10.994107	16.899750
Bahrain	8.504587	16.894587
Thailand	7.939394	16.874242
Israel	12.100346	16.835398
Austria	11.050063	16.710701
France	10.790507	16.245933
Saudi Arabia	8.888889	16.213333
USA	11.234211	16.194053
Greece	9.885397	16.120573
Canada	12.292237	15.641005
Malta	8.000000	15.174624
Portugal	9.456293	14.931053
Cyprus	6.755665	14.839724
Spain	9.381607	14.721928
RSA	6.888889	14.312353

	AverageQuantity	AverageTotalPrice
Country		
Netherlands	13.606356	14.147509
Unspecified	8.673212	12.577686
Japan	12.053140	10.550097
United Kingdom	6.175554	10.177539
West Indies	7.000000	9.237925
Brazil	4.273810	8.150119
Nigeria	3.433333	4.679667

In [73]:
```
# Total Sales by Country - Bar Chart
plt.figure(figsize=(12, 6))

# Average Order Quantity and TotalPrice by Country - Bar Chart
average_order_by_country.plot(kind='bar', figsize=(12, 6), colormap='viridis
plt.xlabel('Country')
plt.ylabel('Average Quantity / Average Total Sales')
plt.title('Average Order Quantity and TotalPrice by Country')
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



In [74]:
```
# Top products purchased in each country
top_products_by_country = df.groupby(['Country', 'Description']).size().grou
top_products_by_country.rename(columns={0: 'TopProducts'}, inplace=True)
display(top_products_by_country.head())
```

	Country	TopProducts
0	Australia	Lunch Bag Red Retrospot
1	Austria	Red Retrospot Bowl
2	Bahrain	Ceramic Cake Bowl + Hanging Cakes
3	Belgium	Postage
4	Bermuda	Assorted Ice Cream Fridge Magnets

In [75]:

```
# Revenue By each month
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

# Extract month and year from 'InvoiceDate'
df['YearMonth'] = df['InvoiceDate'].dt.to_period('M')

# Calculate revenue for each month
monthly_revenue = df.groupby('YearMonth')['TotalPrice'].sum().reset_index()

# Display the results
print("Monthly Revenue:")
display(monthly_revenue)
```

```
Monthly Revenue:
```

	YearMonth	TotalPrice
0	2009-12	411499.140
1	2010-01	285474.502
2	2010-02	272770.716
3	2010-03	389954.401
4	2010-04	338727.422
5	2010-05	342924.110
6	2010-06	375388.840
7	2010-07	331304.270
8	2010-08	325391.880
9	2010-09	427555.391
10	2010-10	576678.730
11	2010-11	694274.902
12	2010-12	382810.570
13	2011-01	311091.520
14	2011-02	267710.400
15	2011-03	348702.680
16	2011-04	288207.001
17	2011-05	375335.140
18	2011-06	347271.400
19	2011-07	352199.171
20	2011-08	350416.820
21	2011-09	510527.142
22	2011-10	578636.760
23	2011-11	741418.140
24	2011-12	224097.880

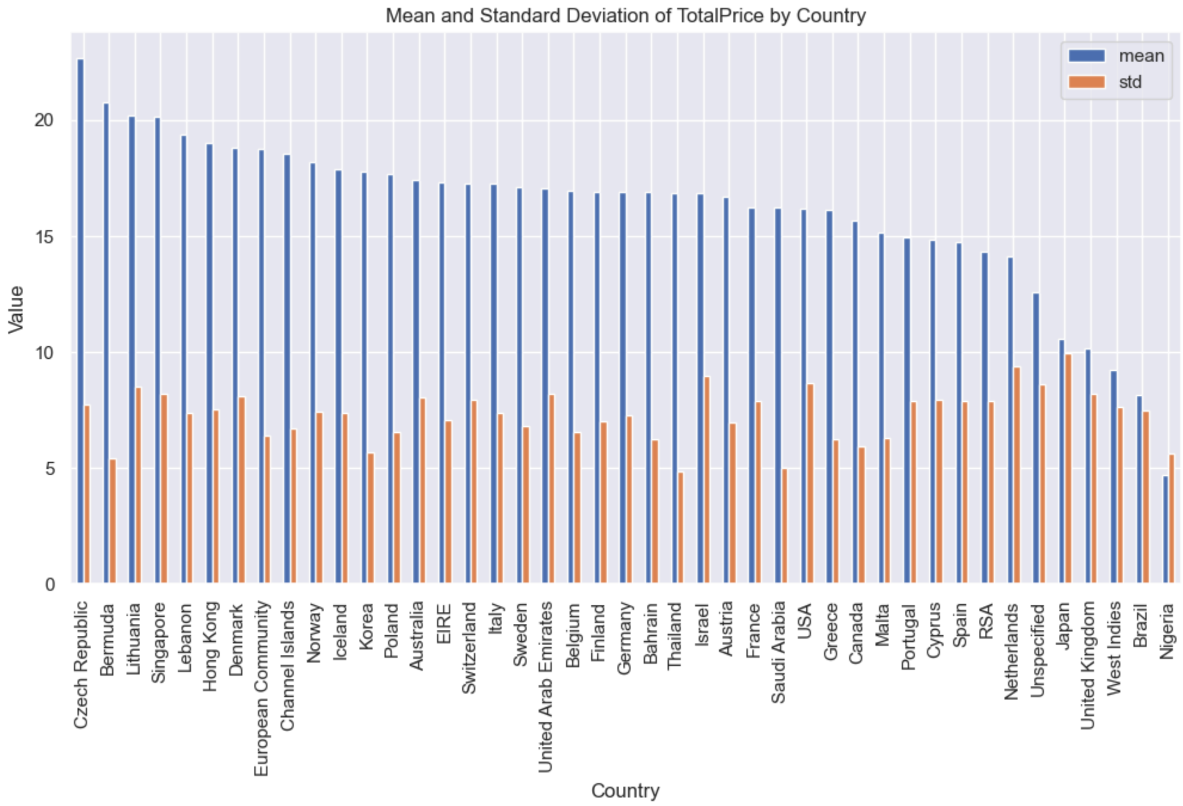
In [76]:
```
# Calculate percentage revenue based on countries
country_percentage_revenue = df.groupby('Country')['TotalPrice'].sum() / df

print("\nPercentage Revenue Based on Countries:")
display(country_percentage_revenue)
```

```
Percentage Revenue Based on Countries:
```

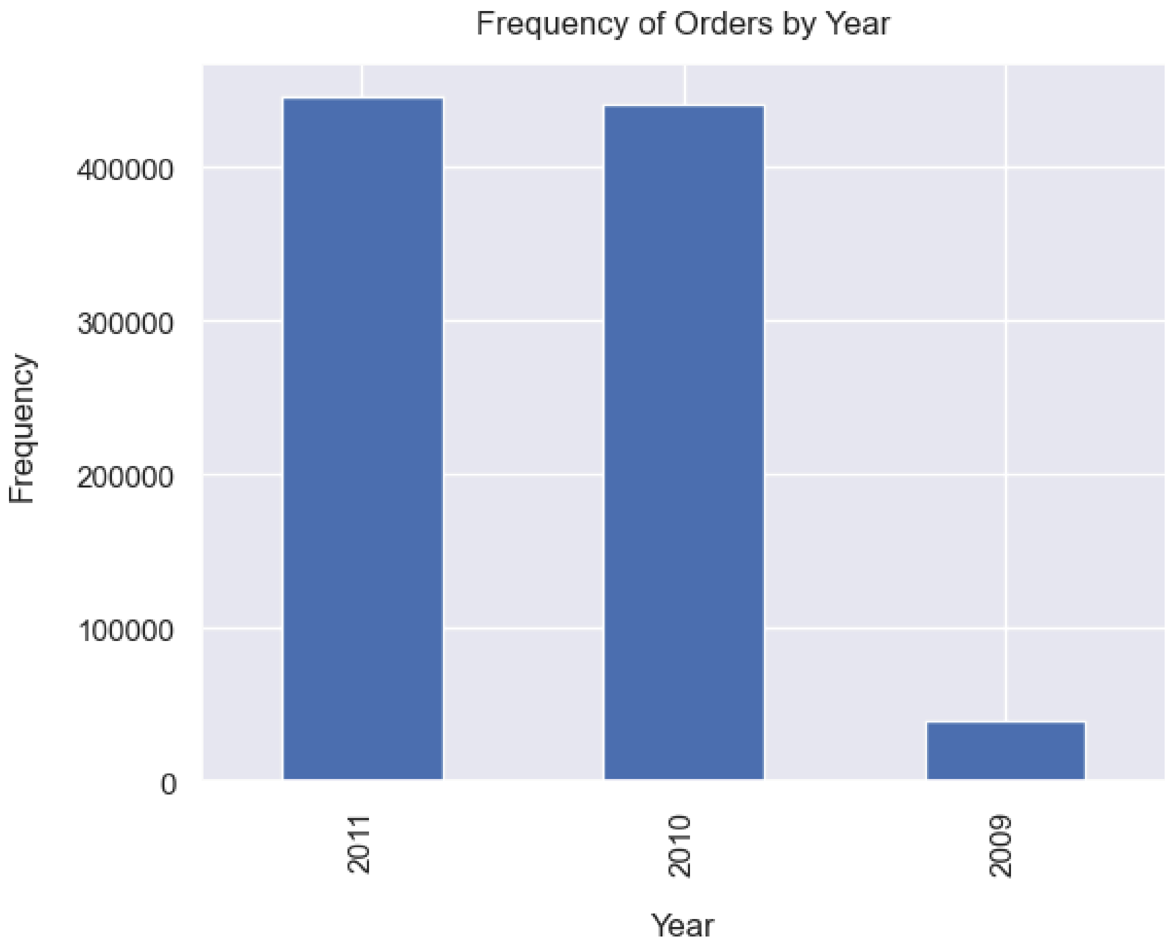
```
Country
Australia               0.162576
Austria                 0.135547
Bahrain                 0.018695
Belgium                 0.481758
Bermuda                 0.006120
Brazil                  0.006950
Canada                  0.034774
Channel Islands         0.250816
Cyprus                  0.152911
Czech Republic          0.003450
Denmark                 0.090430
EIRE                    2.481230
European Community      0.010276
Finland                 0.149831
France                  1.966591
Germany                 2.445482
Greece                  0.088537
Hong Kong               0.044830
Iceland                 0.037237
Israel                  0.049393
Italy                   0.227976
Japan                   0.022170
Korea                   0.008833
Lebanon                 0.007663
Lithuania               0.021541
Malta                   0.040978
Netherlands             0.280211
Nigeria                 0.001425
Norway                  0.182215
Poland                  0.082476
Portugal                0.331199
RSA                     0.022231
Saudi Arabia            0.001481
Singapore               0.041744
Spain                   0.489168
Sweden                  0.137291
Switzerland             0.431834
Thailand                0.011306
USA                     0.062472
United Arab Emirates    0.071608
United Kingdom         88.810732
Unspecified             0.091041
West Indies             0.004970
Name: TotalPrice, dtype: float64
```

In [77]:

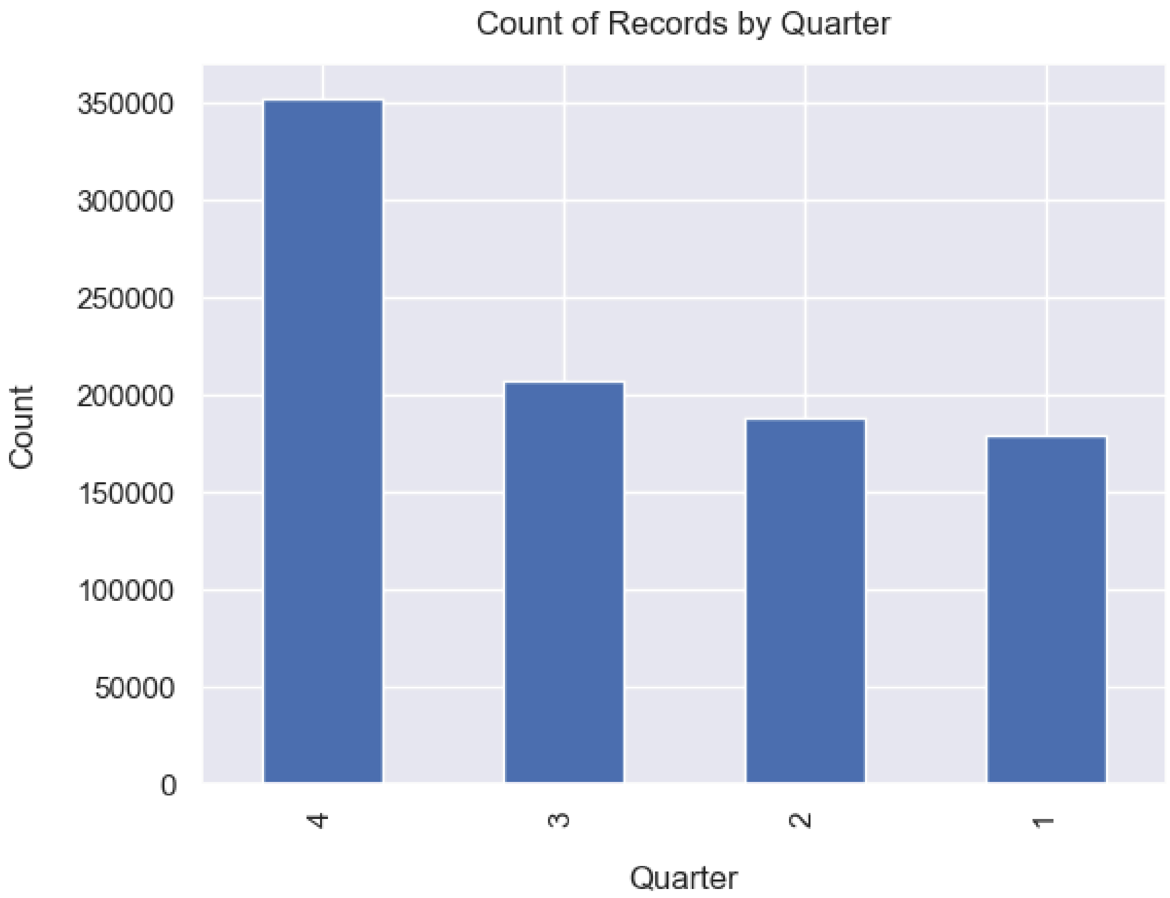
```
# Group by country and calculate mean and standard deviation of TotalPrice
country_stats = df.groupby('Country')['TotalPrice'].agg(['mean', 'std']).so

# Set the plot size
fig, ax = plt.subplots(figsize=(12, 6))

# Plot the bar chart
country_stats.plot(kind='bar', x='Country', y=['mean', 'std'], ax=ax, legen

# Set title, x-axis label, and y-axis label
plt.title('Mean and Standard Deviation of TotalPrice by Country')
plt.xlabel('Country')
plt.ylabel('Value')

# Display the plot
plt.show()
```



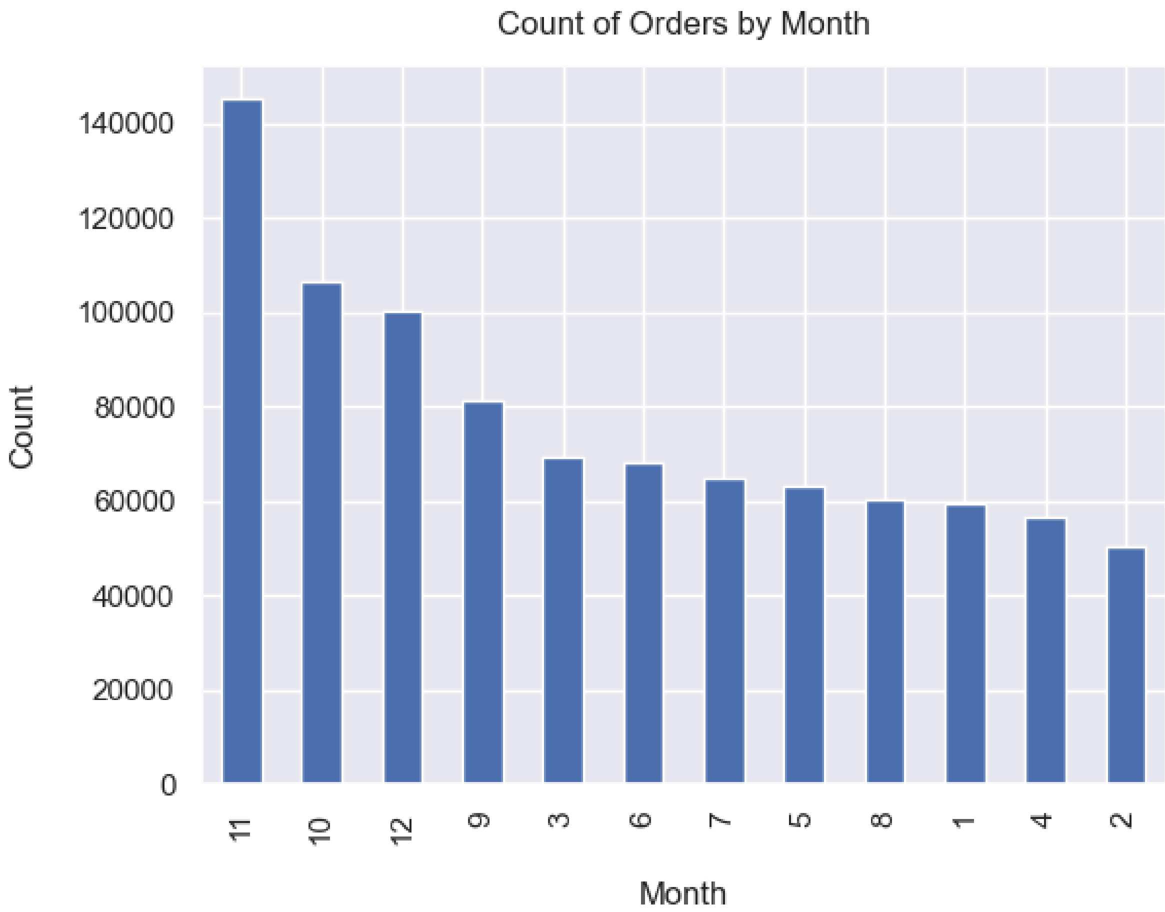
In [78]:
```
df['Year'].value_counts().plot(kind='bar')

plt.xlabel('Year', labelpad=14)  # Set x-axis label
plt.ylabel('Frequency', labelpad=14)  # Set y-axis label
plt.title('Frequency of Orders by Year', y=1.02)  # Set title

# Display the plot
plt.show()
```



In [79]:
```
df['Quarter'].value_counts().plot(kind='bar')
# Adding labels and title
plt.xlabel('Quarter', labelpad=14)  # Set x-axis label
plt.ylabel('Count', labelpad=14)  # Set y-axis label
plt.title('Count of Records by Quarter', y=1.02)  # Set title

# Display the plot
plt.show()
```



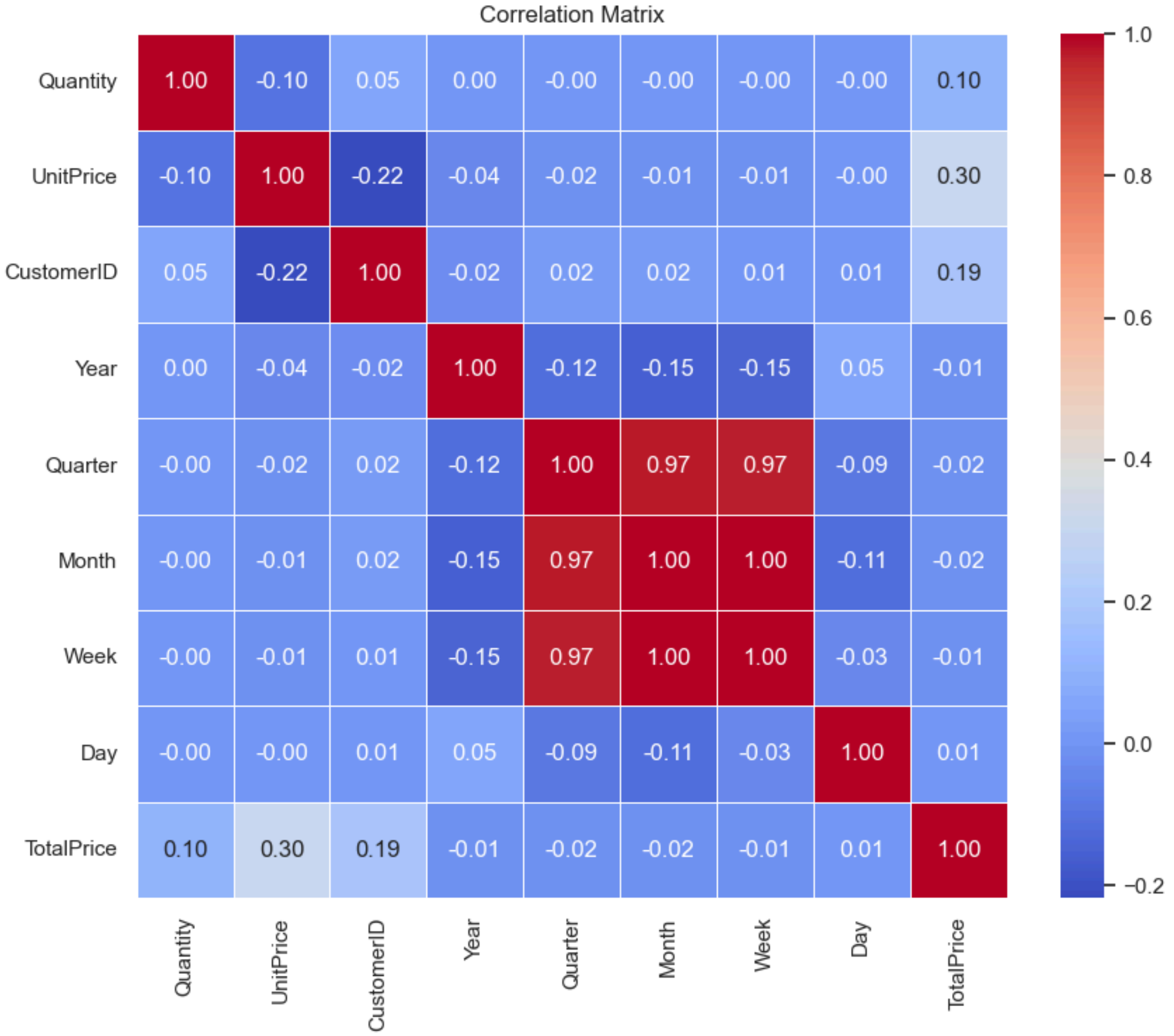
In [80]:
```
df['Month'].value_counts().plot(kind='bar')
# Adding labels and title
plt.xlabel('Month', labelpad=14)  # Set x-axis label
plt.ylabel('Count', labelpad=14)  # Set y-axis label
plt.title('Count of Orders by Month', y=1.02)  # Set title

# Display the plot
plt.show()
```



In [81]:
```
# Calculate correlation matrix for numeric columns only
correlation_matrix = df.select_dtypes(include=[np.number]).corr()

# Display the correlation matrix
display(correlation_matrix)
```

	Quantity	UnitPrice	CustomerID	Year	Quarter	Month	Week
Quantity	1.000000	-0.097673	0.051810	0.001357	-0.002349	-0.003135	-0.003257
UnitPrice	-0.097673	1.000000	-0.217483	-0.035040	-0.016096	-0.011056	-0.010114
CustomerID	0.051810	-0.217483	1.000000	-0.017834	0.021973	0.017478	0.014275
Year	0.001357	-0.035040	-0.017834	1.000000	-0.116838	-0.151448	-0.149035
Quarter	-0.002349	-0.016096	0.021973	-0.116838	1.000000	0.973007	0.970968
Month	-0.003135	-0.011056	0.017478	-0.151448	0.973007	1.000000	0.996584
Week	-0.003257	-0.010114	0.014275	-0.149035	0.970968	0.996584	1.000000
Day	-0.000413	-0.000008	0.009995	0.053007	-0.090002	-0.113541	-0.034105
TotalPrice	0.104585	0.301293	0.189247	-0.006835	-0.016077	-0.015011	-0.014392

In [82]:
```
# Plotting the correlation matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", line
plt.title('Correlation Matrix')
plt.show()
```



There are no correlation between the columns

RFM

RFM (Recency, Frequency, Monetary Value) is a marketing analysis model that categorizes customers based on their recent activity, purchase frequency, and monetary contributions. It helps businesses identify segments like high-value customers for targeted marketing, re-engage at-risk customers, and tailor strategies based on customer behavior.

- RECENCY (R): Days since last purchase
- FREQUENCY (F): Total number of purchases
- MONETARY VALUE (M): Total money this customer spent.

Recency

In [83]:
```
last_date=df['Date'].max()
```

In [84]:
```
# Finding the customers and their last purchase date
recency=df.groupby('CustomerID')['Date'].max().reset_index()
display(recency.head())
```

	CustomerID	Date
0	0	2011-12-09
1	12346	2010-06-28
2	12347	2011-12-07
3	12348	2011-04-05
4	12349	2011-11-21

In [85]:
```
recency['Recency']=recency['Date'].apply(lambda x: (last_date-x).days)
display(recency.head())
```

	CustomerID	Date	Recency
0	0	2011-12-09	0
1	12346	2010-06-28	529
2	12347	2011-12-07	2
3	12348	2011-04-05	248
4	12349	2011-11-21	18

We can see it has been 325 days since CustomerID 12346 purchased anything

Frequency

In [86]:
```
# Find the number of time a custeomer has made a purchase
frequency=df.groupby('CustomerID')['InvoiceNo'].count().reset_index()
frequency.columns = ['CustomerID','Frequency']
display(frequency.head())
```

	CustomerID	Frequency
0	0	221102
1	12346	32
2	12347	205
3	12348	27
4	12349	156

Monetary

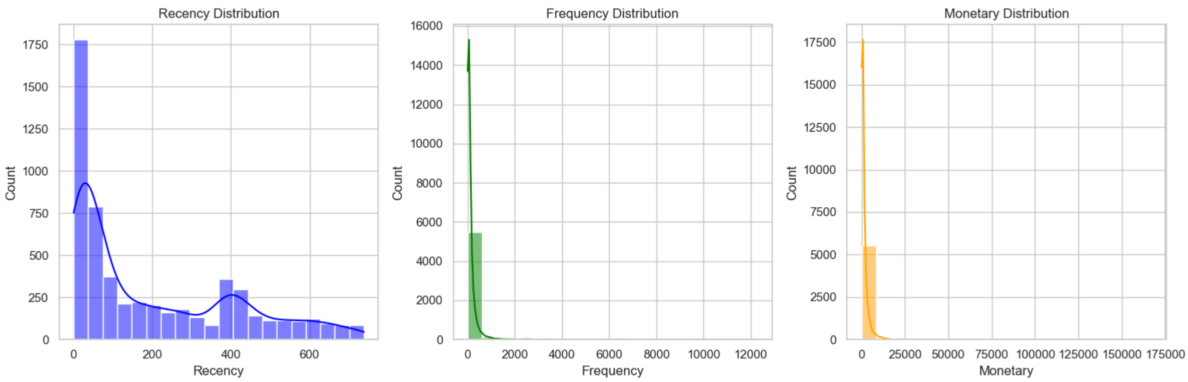
In [87]:
```
# Finding how much each customer has spent overall
monetary=df.groupby('CustomerID')['TotalPrice'].sum().reset_index()
monetary.columns = ['CustomerID','Monetary']
display(monetary.head())
```

	CustomerID	Monetary
0	0	1613001.77
1	12346	327.86
2	12347	3667.95
3	12348	333.24
4	12349	2723.64

In [88]:
```
# Creating a RFM Table
rf_table = recency.merge(frequency, on='CustomerID')
rfm_table=rf_table.merge(monetary, on='CustomerID')
rfm=rfm_table.drop('Date', axis=1)
rfm=rfm[rfm['CustomerID']!=0]
display(rfm.head())
```

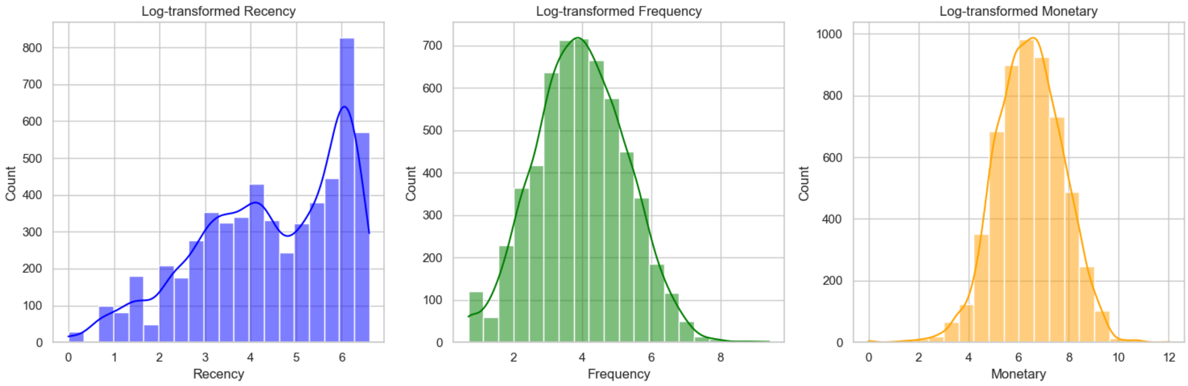
	CustomerID	Recency	Frequency	Monetary
1	12346	529	32	327.86
2	12347	2	205	3667.95
3	12348	248	27	333.24
4	12349	18	156	2723.64
5	12350	310	16	294.40

In [89]:
```
sns.set(style="whitegrid")

# Create a subplot layout
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Plotting displots for Recency, Frequency, and Monetary
sns.histplot(rfm['Recency'], bins=20, kde=True, color='blue', ax=axes[0])
axes[0].set_title('Recency Distribution')
axes[0].set_xlabel('Recency')

sns.histplot(rfm['Frequency'], bins=20, kde=True, color='green', ax=axes[1]
axes[1].set_title('Frequency Distribution')
axes[1].set_xlabel('Frequency')

sns.histplot(rfm['Monetary'], bins=20, kde=True, color='orange', ax=axes[2]
axes[2].set_title('Monetary Distribution')
axes[2].set_xlabel('Monetary')

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



To address the skewness observed in the distribution for RFM (Recency, Frequency, Monetary value) analysis, a transformation of the data will be applied using logarithmic operations. This approach aims to normalize the data distribution, enhancing the effectiveness of subsequent analyses.

In [90]:
```
rfm_log = np.log1p(rfm[['Recency', 'Frequency', 'Monetary']])

# Plot histograms of the log-transformed data
plt.figure(figsize=(15, 5))

# Recency
plt.subplot(1, 3, 1)
sns.histplot(rfm_log['Recency'], bins=20, kde=True, color='blue')
plt.title('Log-transformed Recency')

# Frequency
plt.subplot(1, 3, 2)
sns.histplot(rfm_log['Frequency'], bins=20, kde=True, color='green')
plt.title('Log-transformed Frequency')

# Monetary
plt.subplot(1, 3, 3)
sns.histplot(rfm_log['Monetary'], bins=20, kde=True, color='orange')
plt.title('Log-transformed Monetary')

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

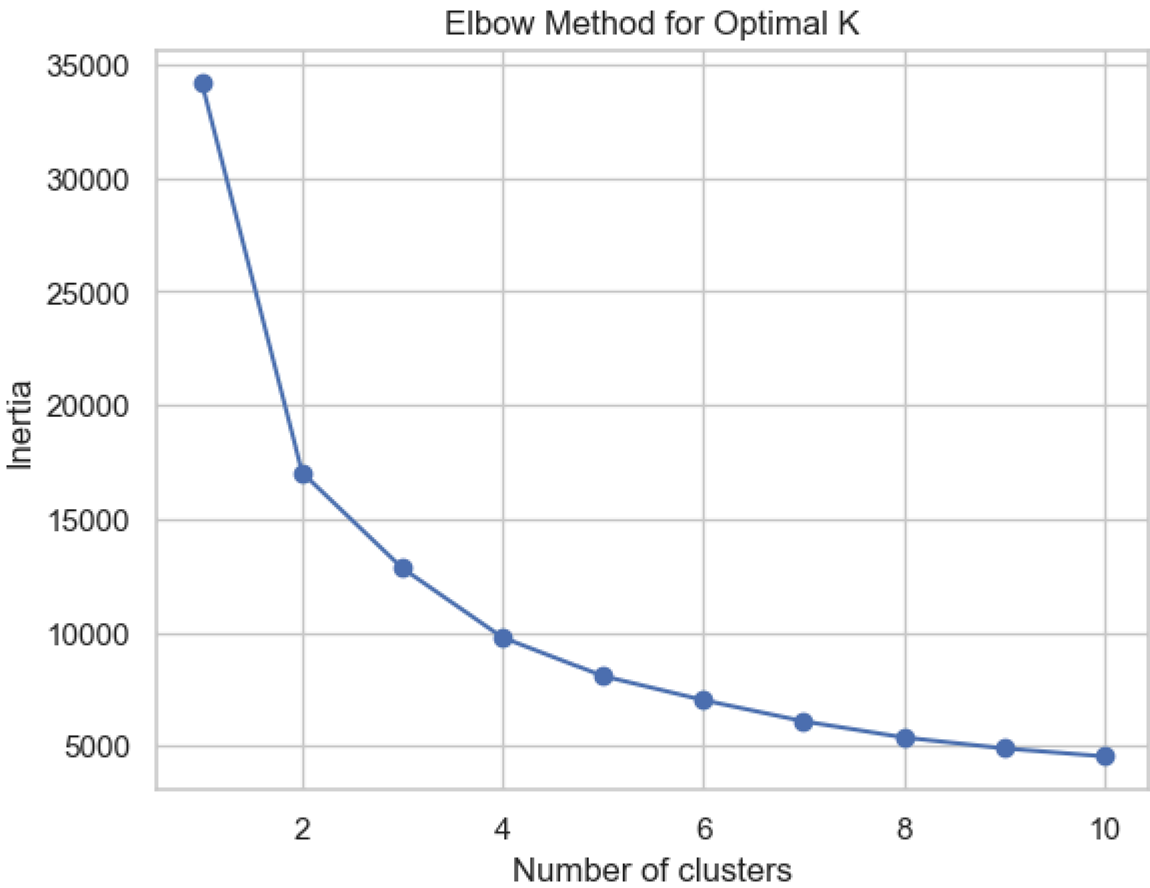


Customer Segmentation using K-Means clustering (RFM)

To determine the optimal number of clusters for customer segmentation using K-means clustering, the Elbow Method, Silhouette Score, and OptimalK will be employed. These techniques help in identifying the most suitable number of clusters by analyzing the dataset's structure and ensuring the chosen clusters are meaningful and distinct.

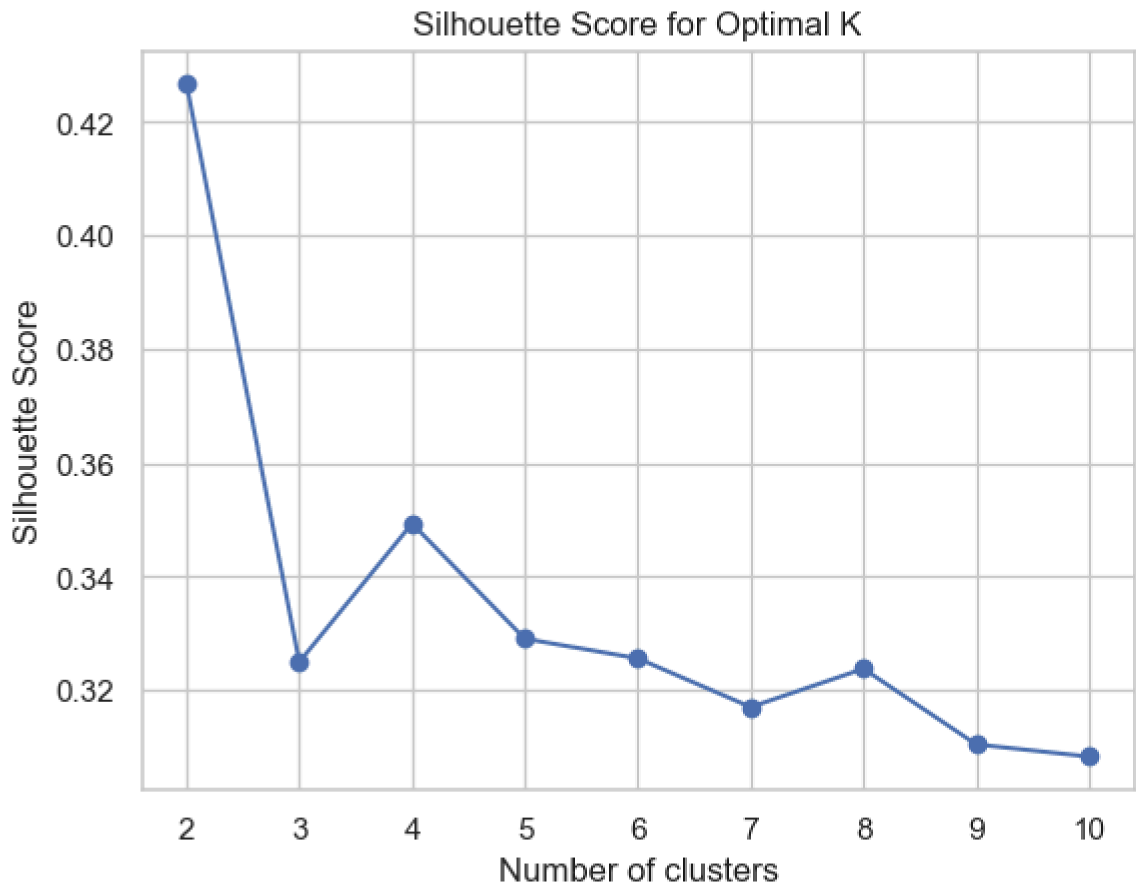
In [91]:
```
# Elbow Method
# In the Elbow Method graph, look for the point where the inertia (within-c

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

inertias = []
for k in range(1, 11):
```

```
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(rfm_log)  # 'data' is your RFM table
    inertias.append(kmeans.inertia_)

plt.plot(range(1, 11), inertias, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.show()
```



By the looks of it, The optimal number of Clusters is 3

In [92]:
```
# Silhouette Score
# In the Silhouette Score graph, aim to find the K value that results in the
# The silhouette score measures how similar an object is to its own cluster
# Higher silhouette scores indicate well-defined clusters.

from sklearn.metrics import silhouette_score

silhouette_scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(rfm_log)  # 'data' is your RFM table
    score = silhouette_score(rfm_log, labels)
    silhouette_scores.append(score)

plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score for Optimal K')
plt.show()
```



From the graph, 2 is the optimal number of clusters

In [93]:
```
pip install optimal_k
```

```
ERROR: Could not find a version that satisfies the requirement optimal_k (f
rom versions: none)
ERROR: No matching distribution found for optimal_k
Note: you may need to restart the kernel to use updated packages.
```

In [94]:
```
from optimal_k import OptimalK

optimalK = OptimalK(parallel_backend='multiprocessing')
n_clusters = optimalK(rfm_log, cluster_array=np.arange(1, 11))
print(f'Optimal number of clusters: {n_clusters}')
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[94], line 1
----> 1 from optimal_k import OptimalK
      3 optimalK = OptimalK(parallel_backend='multiprocessing')
      4 n_clusters = optimalK(rfm_log, cluster_array=np.arange(1, 11))

ModuleNotFoundError: No module named 'optimal_k'
```

All 3 are giving different number of clusters, so i'll take the avg of it. So the ideal number of clusters is 4

In [95]:
```
n_clusters=4
# Selecting only the relevant columns for clustering
rfm_for_clustering = rfm_log[['Recency', 'Frequency', 'Monetary']]

# Initializing KMeans with the optimal number of clusters
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
```

```
# Fitting the KMeans model to your data
rfm['Cluster'] = kmeans.fit_predict(rfm_for_clustering)

# Displaying the resulting clusters in your DataFrame
display(rfm.head())
```

	CustomerID	Recency	Frequency	Monetary	Cluster
1	12346	529	32	327.86	2
2	12347	2	205	3667.95	0
3	12348	248	27	333.24	2
4	12349	18	156	2723.64	0
5	12350	310	16	294.40	2

In [96]:
```
rfm['Cluster'].value_counts()
```

Out[96]:
```
Cluster
1    1605
2    1533
0    1292
3    1238
Name: count, dtype: int64
```

In [97]:
```
rfm.groupby('Cluster').mean()
```

Out[97]:

Cluster	CustomerID	Recency	Frequency	Monetary
0	15245.890867	24.648607	373.366099	4293.217320
1	15287.077259	285.137695	88.774455	1061.576988
2	15396.729289	395.026745	13.887802	174.843622
3	15340.179321	32.936187	47.382068	580.504074

Cluster 0 is the High-Value Customers (Champions).

Clsuter 1 is the Potential Loyalists (Loyal Customers).

Cluster 2 is At-Risk Customers (Needs Attention).

Clsuter 3 is the Lost Customers (Require Re-engagement).

How do cancellations impact overall sales trends?

In [98]:
```
data['IsCancelled'] = data['InvoiceNo'].astype(str).str.startswith('C')
data.head()
```

Out[98]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Counti
0	536365	85123A	White Hanging Heart T-Light Holder	6	2010-12-01 08:26:00	2.55	17850	Unite Kingdo
1	536365	71053	White Metal Lantern	6	2010-12-01 08:26:00	3.39	17850	Unite Kingdo
2	536365	84406B	Cream Cupid Hearts Coat Hanger	8	2010-12-01 08:26:00	2.75	17850	Unite Kingdo
3	536365	84029G	Knitted Union Flag Hot Water Bottle	6	2010-12-01 08:26:00	3.39	17850	Unite Kingdo
4	536365	84029E	Red Woolly Hottie White Heart.	6	2010-12-01 08:26:00	3.39	17850	Unite Kingdo

In [99]:

```
data.columns
```

Out[99]:

```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country', 'Year', 'Quarter', 'Month',
       'Week', 'Day', 'TotalPrice', 'YearQuarter', 'YearMonth', 'Date',
       'DayOfWeek', 'IsCancelled'],
      dtype='object')
```

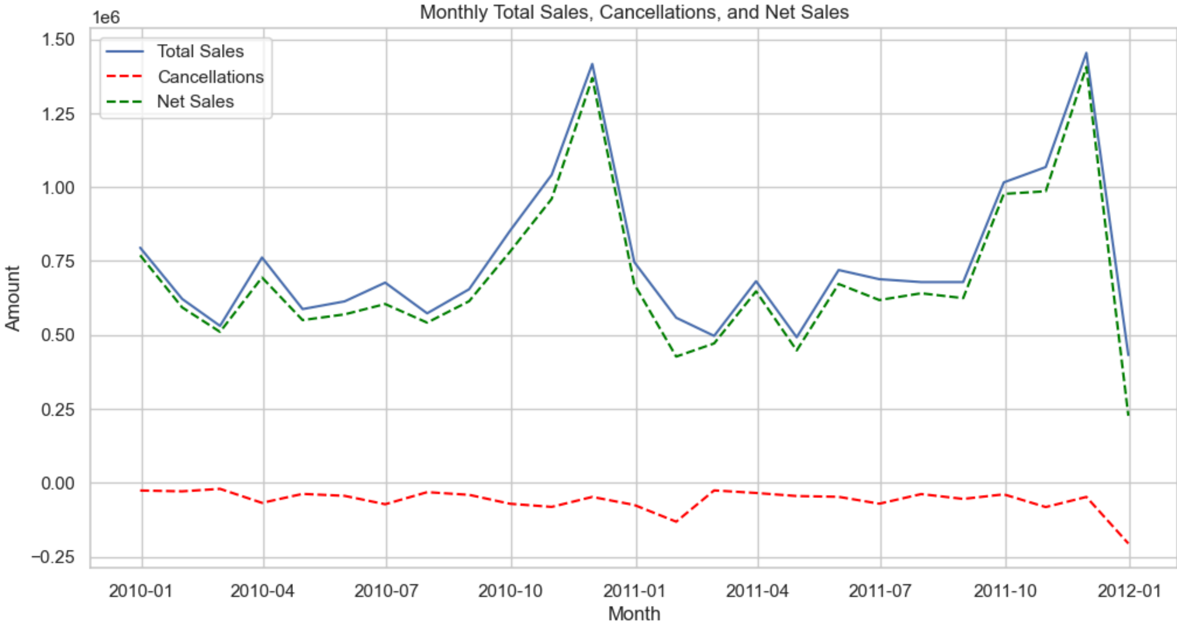
In [100…

```
# Create a new DataFrame with monthly sales data
monthly_sales_data = data.groupby(pd.Grouper(key='InvoiceDate', freq='M')).a
    monthly_total_sales=('TotalPrice', 'sum'),
    monthly_cancellations=('TotalPrice', lambda x: x[data['IsCancelled']].su
)

# Calculate monthly_net_sales by adding total sales and cancellations
monthly_sales_data['monthly_net_sales'] = (
    monthly_sales_data['monthly_total_sales'] + monthly_sales_data['monthly_
)

# Reset index to make the 'InvoiceDate' a regular column
monthly_sales_data.reset_index(inplace=True)

# Display the resulting DataFrame
display(monthly_sales_data)
```

	InvoiceDate	monthly_total_sales	monthly_cancellations	monthly_net_sales
0	2009-12-31	795342.460	-25835.45	769507.010
1	2010-01-31	622051.682	-28668.86	593382.822
2	2010-02-28	530828.266	-20239.36	510588.906
3	2010-03-31	762153.711	-67661.27	694492.441
4	2010-04-30	587666.222	-37284.60	550381.622
5	2010-05-31	613668.160	-43967.33	569700.830
6	2010-06-30	677148.560	-72348.69	604799.870
7	2010-07-31	573283.690	-31444.79	541838.900
8	2010-08-31	654764.360	-40477.52	614286.840
9	2010-09-30	851093.441	-70591.03	780502.411
10	2010-10-31	1041671.840	-81290.74	960381.100
11	2010-11-30	1416664.642	-47595.94	1369068.702
12	2010-12-31	746659.940	-74720.72	671939.220
13	2011-01-31	558390.600	-131363.05	427027.550
14	2011-02-28	497021.450	-25519.15	471502.300
15	2011-03-31	681986.560	-34201.28	647785.280
16	2011-04-30	492357.921	-44600.65	447757.271
17	2011-05-31	719771.530	-47202.51	672569.020
18	2011-06-30	688798.960	-70569.78	618229.180
19	2011-07-31	678984.411	-37919.13	641065.281
20	2011-08-31	679014.940	-54330.80	624684.140
21	2011-09-30	1016088.522	-38838.51	977250.012
22	2011-10-31	1067934.890	-81895.50	986039.390
23	2011-11-30	1454133.110	-47537.03	1406596.080
24	2011-12-31	432164.800	-205089.27	227075.530

In [101…

```
# Plotting Monthly Total Sales, Cancellations, and Net Sales
plt.figure(figsize=(12, 6))
plt.plot(monthly_sales_data['InvoiceDate'], monthly_sales_data['monthly_tota
plt.plot(monthly_sales_data['InvoiceDate'], monthly_sales_data['monthly_canc
plt.plot(monthly_sales_data['InvoiceDate'], monthly_sales_data['monthly_net_

plt.xlabel('Month')
plt.ylabel('Amount')
plt.title('Monthly Total Sales, Cancellations, and Net Sales')
plt.legend()
plt.show()
```


Monthly Total Sales, Cancellations, and Net Sales
1e6
Total Sales
Cancellations
Net Sales
Amount
Month
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
2010-01
2010-04
2010-07
2010-10
2011-01
2011-04
2011-07
2011-10
2012-01

In [ ]: