Project : Customer Segmentation using RFM Analysis

Project Description:

In this project assignment, I worked with the eCommerce dataset provided (https://www.kaggle.com/datasets/carrie1/ecommerce-data) to create a Customer Segmentation model using the RFM (Recency, Frequency, Monetary) analysis method. RFM segmentation is a powerful technique I used to group customers based on their recent purchasing behavior, purchase frequency, and monetary value, enabling more targeted marketing and customer engagement strategies.

Objective:

My objective was to perform RFM analysis on the dataset and segment the customers into distinct groups based on their RFM scores. These segments provided valuable insights for marketing and customer retention strategies.

In [1]:

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.express as px
from datetime import datetime as dt
import calendar
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Data Preprocessing

In [2]:
```python
# Loading Crime Dataset
df = pd.read_csv('data.csv',encoding='unicode_escape')
```

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

Out[3]:

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

1) Data Overview

In [4]:
```python
# Displaying the number of rows and columns of the dataset
df.shape
```

Out[4]:
```
(541909, 8)
```

In [5]:
```python
# Displaying the types of Datatypes
df.dtypes
```

Out[5]:
```
InvoiceNo       object
StockCode       object
Description     object
Quantity         int64
InvoiceDate     object
UnitPrice      float64
CustomerID     float64
Country         object
dtype: object
```

In [6]:
```
# Displaying the all the columns of the Dataset
df.columns
```

Out[6]:
```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

In [7]:
```
# Displaying the basic information of the dataset
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
```

In [8]:
```
# checking for any null values
df.isna().sum()
```

Out[8]:
```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

In [9]:
```
# Calculating the missing value percentage
missing_percentage = df.isnull().mean() * 100
```

In [10]:
```
print("Percentage of Missing Values per Column:\n", missing_percentage[missing_percentage > 0])
```

```
Percentage of Missing Values per Column:
 Description     0.268311
CustomerID     24.926694
dtype: float64
```

In [11]:
```
# Checking for any duplicate values in the dataset
df.duplicated().sum()
```

Out[11]:
```
5268
```

In [12]:
```
# Dropping duplicate values in the dataset
df.drop_duplicates(inplace=True)
```

In [13]:
```
# Checking for any Negative values in the Quantity Columns
cnt = df['Quantity']<0
cnt.sum()
```

Out[13]:
```
10587
```

In [14]:
```
# Checking for any Negative values in the Unit Price Columns
cnt1 = df['UnitPrice']<0
cnt1.sum()
```

Out[14]:
```
2
```

In [15]:
```
# Displaying the Descriptive Statistics
df.describe().T
```

Out[15]:

	count	mean	std	min	25%	50%	75%	max
Quantity	536641.0	9.620029	219.130156	-80995.00	1.00	3.00	10.00	80995.0
UnitPrice	536641.0	4.632656	97.233118	-11062.06	1.25	2.08	4.13	38970.0
CustomerID	401604.0	15281.160818	1714.006089	12346.00	13939.00	15145.00	16784.00	18287.0

In [16]:
```
# Dropping any null Values
df.dropna(inplace=True)
```

In [17]:
```
# Filtering the quantity columns for values less than 0
df = df[~df['Quantity']<0]
```

In [18]:
```
# Performing outlier removal on the DataFrame for the columns 'Quantity' and 'UnitPrice'.
check_items = ['Quantity','UnitPrice']
for i in check_items:
    low,high = df[i].quantile([0,0.95])
    mask = df[i].between(low,high)
    df = df[mask]
```

In [19]:
```
# Converting date time format
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
df['CustomerID'] = df['CustomerID'].astype(str)
```

In [20]:
```
# Creating time period using the minimum and maximum dates from the 'InvoiceDate' column.
time_period = {
    "Start Date": df['InvoiceDate'].min(),
    "End Date": df['InvoiceDate'].max()
}
```

In [21]:
```
print("Maximum and Minimum Time Period:",time_period)
```

```
Maximum and Minimum Time Period: {'Start Date': Timestamp('2010-12-01 08:26:00'), 'End Date': Timestamp('2011-12
-09 12:50:00')}
```

2) Customer Analysis

In [22]:
```
# Find Unique Customers
unique_customers = df['CustomerID'].nunique()

print("Unique Number of Customers in the dataset:",unique_customers)
```

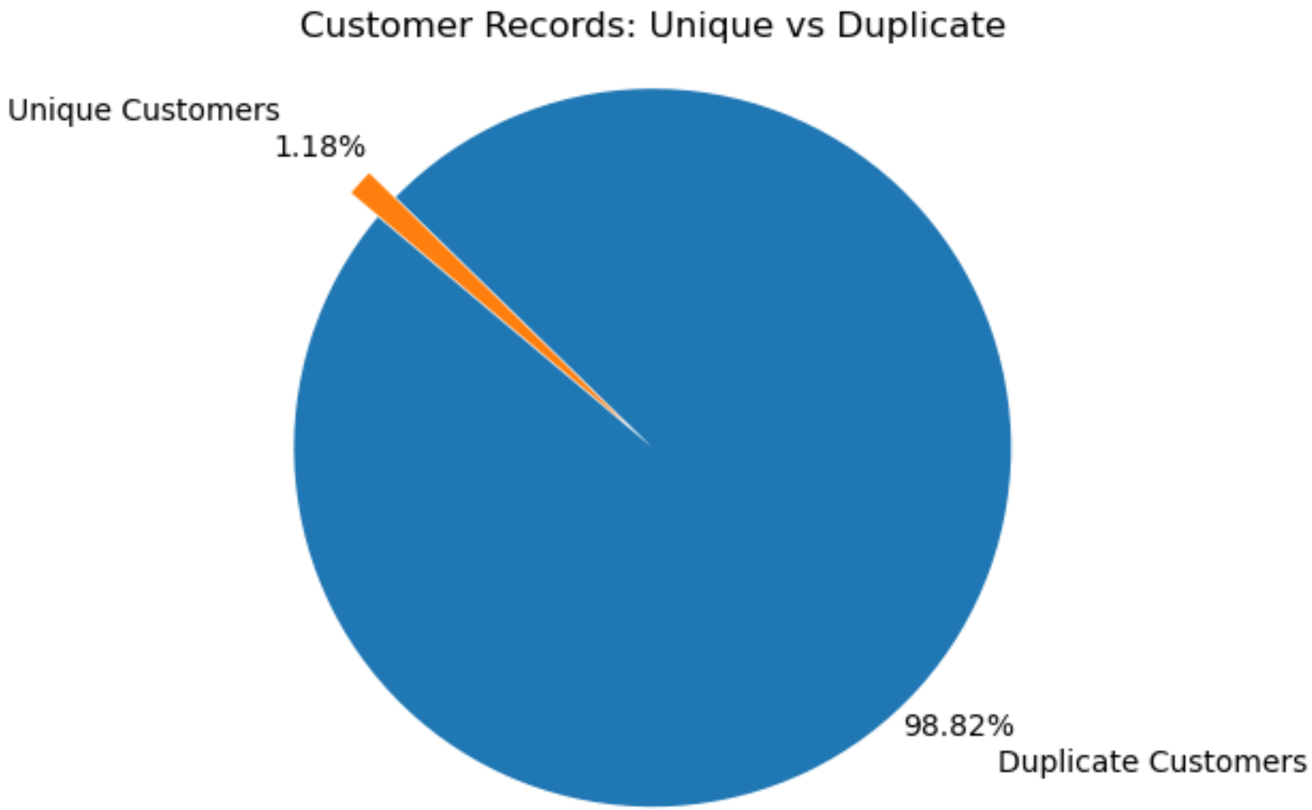
```
Unique Number of Customers in the dataset: 4215
```

In [23]:
```
total_customers = len(df)
```

In [24]:
```
# This section calculates the total and unique number of customers.
sizes = [total_customers - unique_customers, unique_customers]
labels = ['Duplicate Customers', 'Unique Customers']
explode = (0.1, 0)
```

In [25]:
```
def custom_autopct(pct):
    return '{:.2f}%'.format(pct) if pct > 0 else ''
```

In [26]:
```
# Plotting a pie chart to visualize customer records: Unique vs Duplicate
plt.pie(sizes, labels=labels, autopct=lambda pct: custom_autopct(pct), startangle=140,
        pctdistance=1.15, labeldistance=1.30, explode=explode)
plt.axis('equal')
plt.title('Customer Records: Unique vs Duplicate')
plt.show()
```



In [27]:

```
# Displaying the Distribution of Number of Orders per customer
customer = df.groupby('CustomerID')['StockCode'].count()
customer = customer[customer.values<1000]
plt.figure(figsize=(10, 6))
plt.hist(customer.values, bins=50,edgecolor='black')
plt.title('Distribution of Number of Orders per Customer')
plt.xlabel('Number of Orders')
plt.ylabel('Number of Customers')
plt.show()
```

In [28]:
```
# Analyzing Customer Orders
orders_per_customer = df.groupby('CustomerID')['InvoiceNo'].nunique()

top_5_customers_by_orders = orders_per_customer.sort_values(ascending=False).head(5)
```

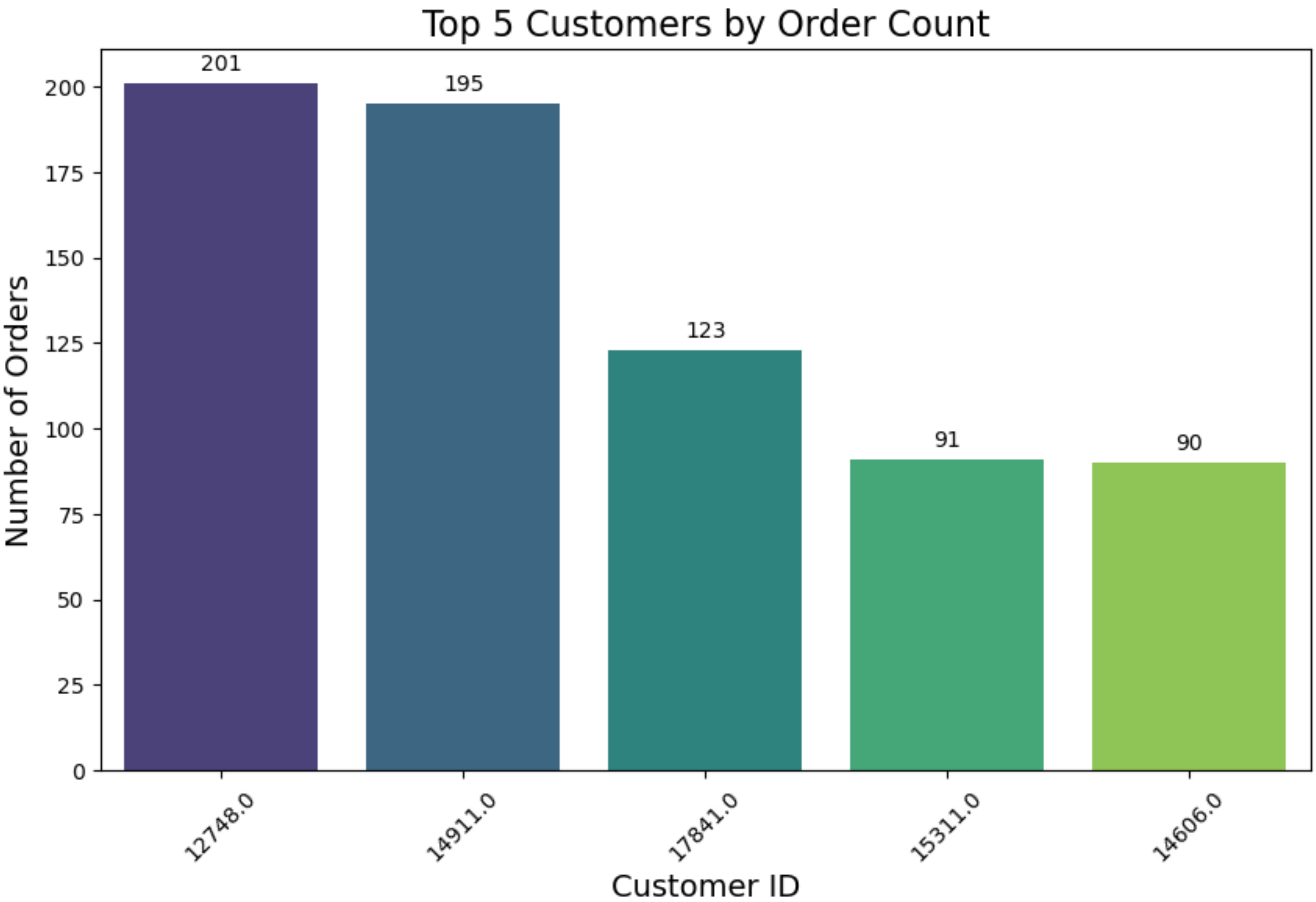
In [29]:
```
# Creating top 5 customer dataframe
top_5_customers_df = pd.DataFrame({'CustomerID': top_5_customers_by_orders.index,
                                   'Orders': top_5_customers_by_orders.values})
```

In [30]:
```
top_5_customers_df
```

Out[30]:

	CustomerID	Orders
0	12748.0	201
1	14911.0	195
2	17841.0	123
3	15311.0	91
4	14606.0	90

In [31]:
```
# Displaying the top 5 customers by Order Count
plt.figure(figsize=(10, 6))
barplot = sns.barplot(x='CustomerID', y='Orders', data=top_5_customers_df, palette='viridis')

plt.title('Top 5 Customers by Order Count', fontsize=16)
plt.xlabel('Customer ID', fontsize=14)
plt.ylabel('Number of Orders', fontsize=14)
plt.xticks(rotation=45)

for p in barplot.patches:
    barplot.annotate(format(p.get_height(), '.0f'),
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='center',
                     xytext=(0, 9),
                     textcoords='offset points')

plt.show()
```



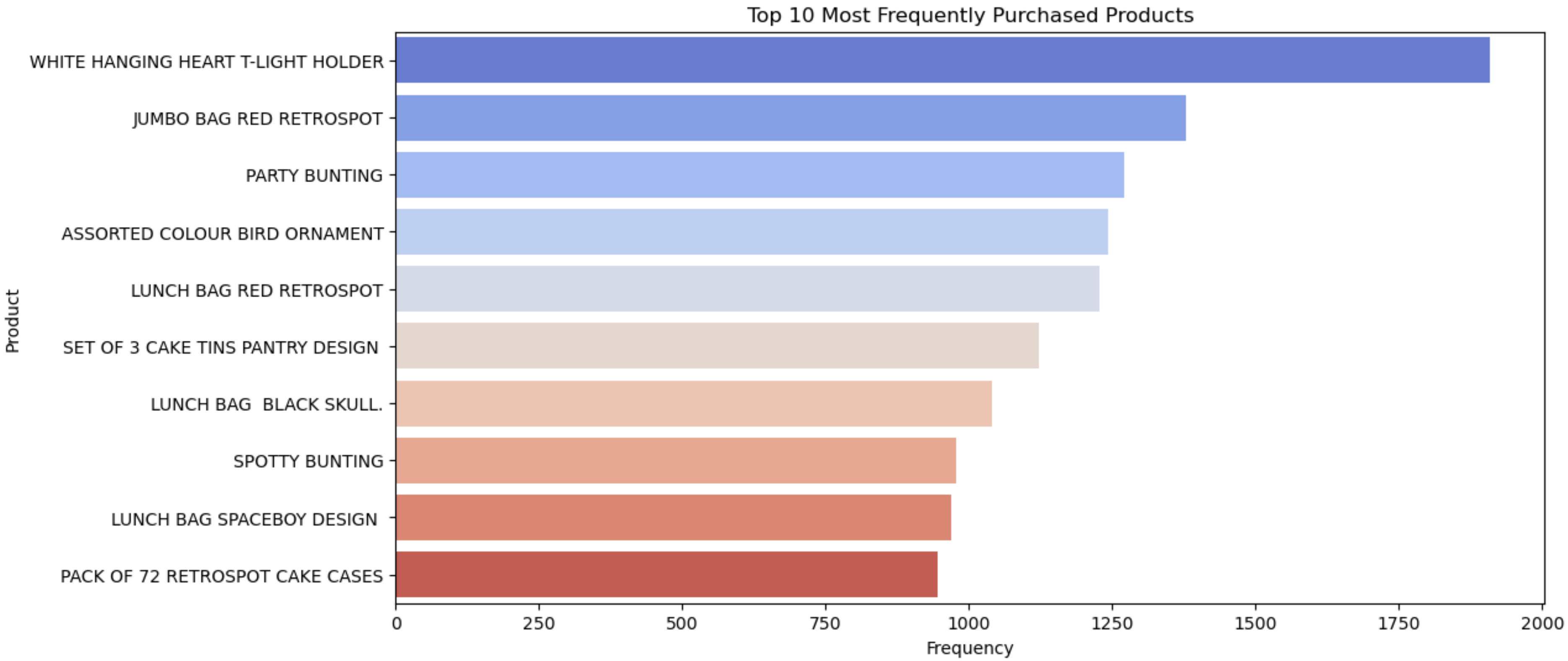
3) Product Analysis

In [32]:
```python
# Top 10 most frequently purchased products
top_10_products = df['Description'].value_counts().head(10)

# Average price of products in the dataset
average_price = df['UnitPrice'].mean()

# Generating revenue by product
df['Revenue'] = df['Quantity'] * df['UnitPrice']
revenue_by_product = df.groupby('Description')['Revenue'].sum()

# Finding the product that generates the highest revenue
highest_revenue_product = revenue_by_product.idxmax()
highest_revenue = revenue_by_product.max()

print ("\nThe Top 10 products are:\n\n",top_10_products)
print ("\nAverage price of products in the dataset:\n\n" ,average_price)
print ("\nThe product that generates the highest revenue is:\n\n" ,(highest_revenue_product, highest_revenue))
```

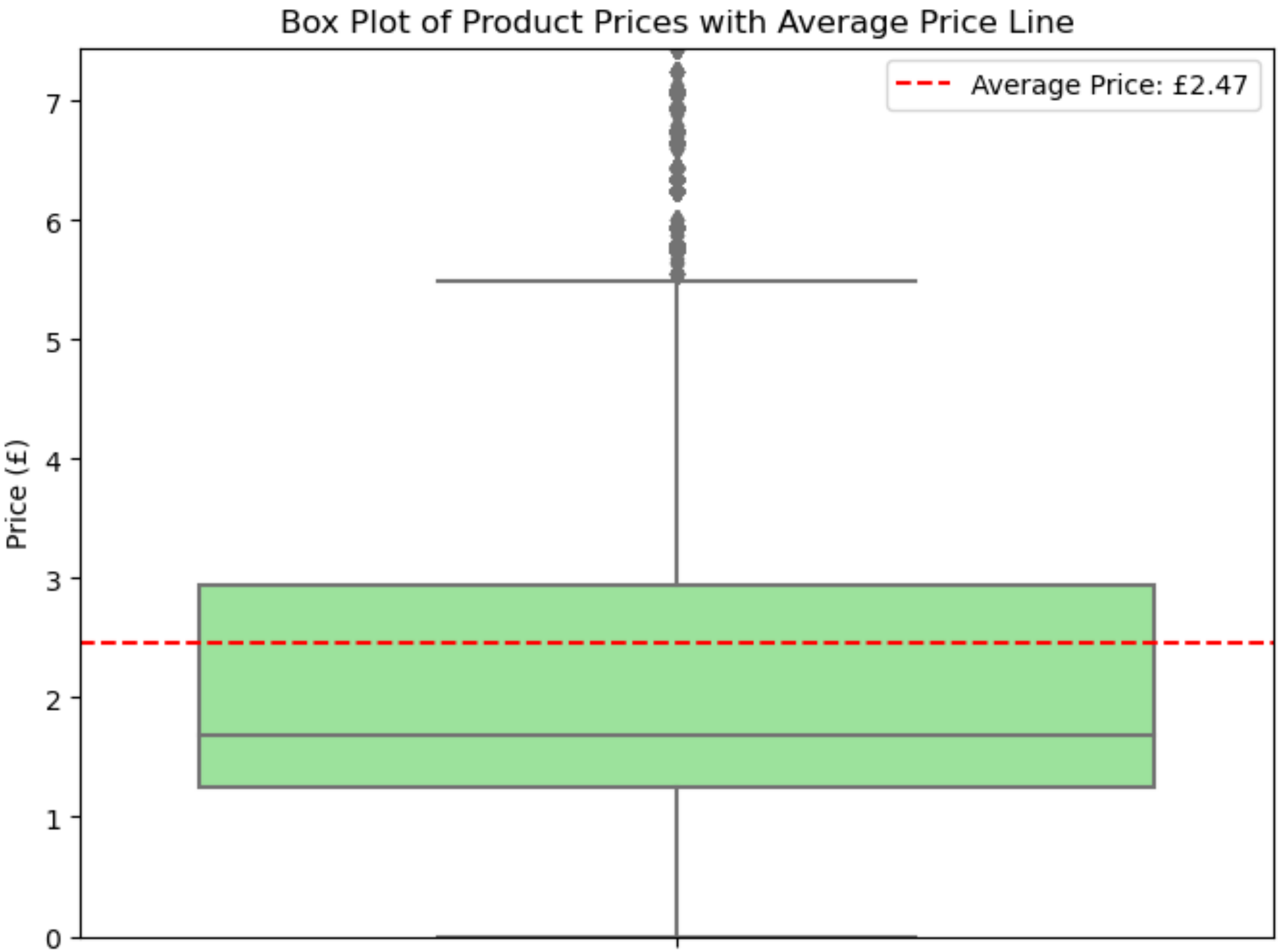
```
The Top 10 products are:

 Description
WHITE HANGING HEART T-LIGHT HOLDER    1909
JUMBO BAG RED RETROSPOT               1379
PARTY BUNTING                         1271
ASSORTED COLOUR BIRD ORNAMENT         1243
LUNCH BAG RED RETROSPOT               1227
SET OF 3 CAKE TINS PANTRY DESIGN      1123
LUNCH BAG  BLACK SKULL.               1040
SPOTTY BUNTING                         977
LUNCH BAG SPACEBOY DESIGN              968
PACK OF 72 RETROSPOT CAKE CASES        946
Name: count, dtype: int64

Average price of products in the dataset:

 2.465995333046914

The product that generates the highest revenue is:

 ('WHITE HANGING HEART T-LIGHT HOLDER', 51472.01)
```

In [33]:
```
# Creating a DataFrame for the top 10 most frequently purchased products
top_10_products_df = pd.DataFrame({'Product': top_10_products.index,
                                   'Frequency': top_10_products.values})
```

In [34]:
```
# Plotting bar chart to visualize top 10 most frequently purchased products
plt.figure(figsize=(12, 6))
sns.barplot(x='Frequency', y='Product', data=top_10_products_df, palette='coolwarm')
plt.title('Top 10 Most Frequently Purchased Products')
plt.xlabel('Frequency')
plt.ylabel('Product')
plt.show()
```



In [35]:
```
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['UnitPrice'], color='lightgreen')
plt.axhline(y=average_price, color='red', linestyle='--', label=f'Average Price: £{average_price:.2f}')
plt.title('Box Plot of Product Prices with Average Price Line')
plt.ylabel('Price (£)')
plt.ylim(0, df['UnitPrice'].quantile(0.95))  # Limiting y-axis to 95th percentile for better visibility
plt.legend()
plt.show()
```



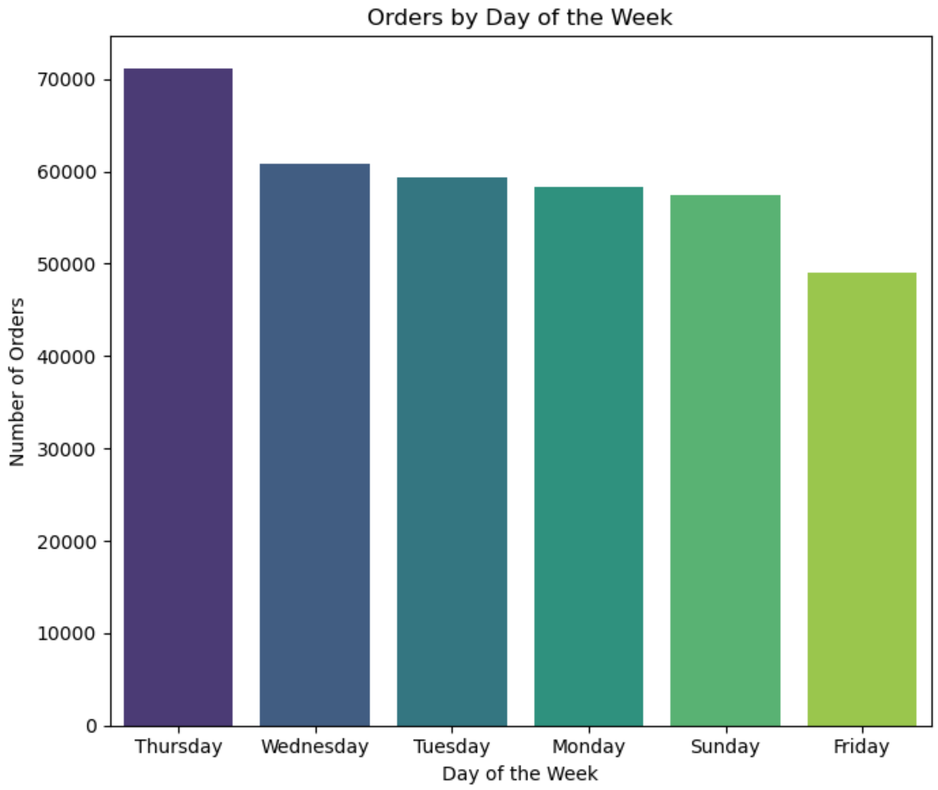
4) Time Analysis

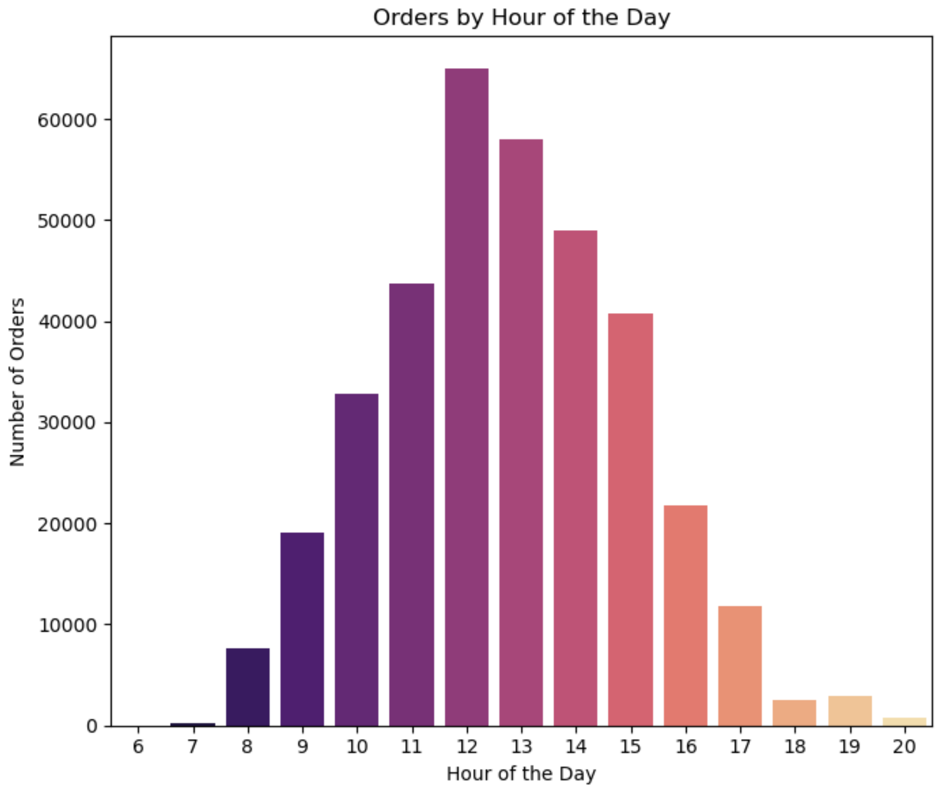
In [36]:

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

# Finding the most common day of the week for orders
most_common_day = df['DayOfWeek'].value_counts().idxmax()

# Finding the most common hour of the day for orders
most_common_hour = df['HourOfDay'].value_counts().idxmax()

print(f"Most Common Day: {most_common_day}")
print(f"Most Common Hour: {most_common_hour}")
```

```
Most Common Day: Thursday
Most Common Hour: 12
```

In [37]:
```
# Aggregating data for visualization
day_order_counts = df['DayOfWeek'].value_counts()
hour_order_counts = df['HourOfDay'].value_counts()

# Visualization
plt.figure(figsize=(14, 6))

# Day of the Week Orders
plt.subplot(1, 2, 1)
sns.barplot(x=day_order_counts.index, y=day_order_counts.values, palette='viridis')
plt.title('Orders by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Orders')

# Hour of the Day Orders
plt.subplot(1, 2, 2)
sns.barplot(x=hour_order_counts.index, y=hour_order_counts.values, palette='magma')
plt.title('Orders by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Orders')

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





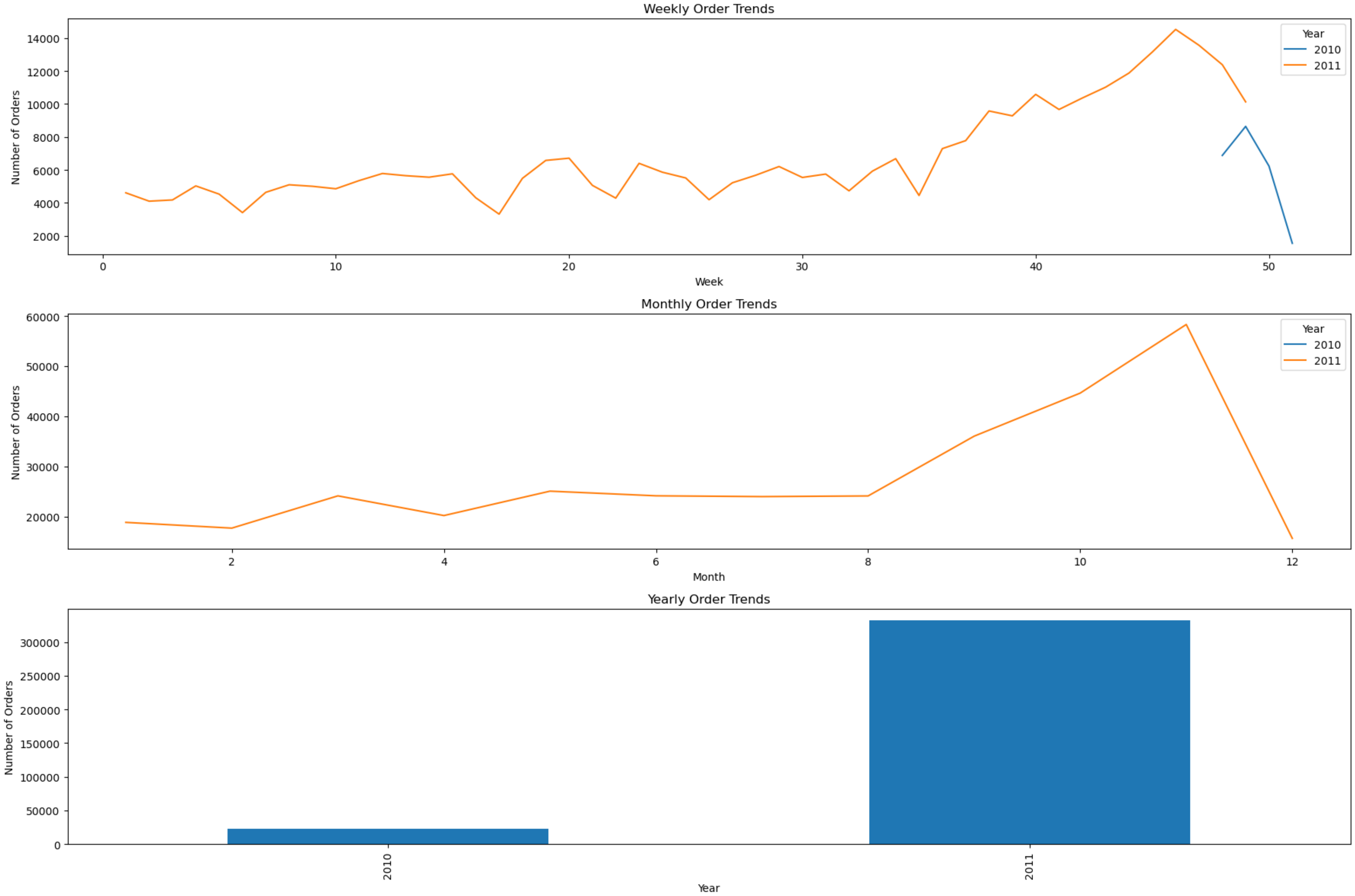
In [38]:

```
# Sorting the data by CustomerID and InvoiceDate
data_sorted = df.sort_values(['CustomerID', 'InvoiceDate'])

# Calculate the difference in InvoiceDate for each consecutive order by the same customer
data_sorted['NextInvoiceDate'] = data_sorted.groupby('CustomerID')['InvoiceDate'].shift(-1)
data_sorted['ProcessingTime'] = (data_sorted['NextInvoiceDate'] - data_sorted['InvoiceDate'])

# Exclude the last order of each customer as it does not have a next order to compare with
processing_times = data_sorted['ProcessingTime'].dropna()

# Calculate the average processing time across all orders
average_processing_time = processing_times.mean()

print(f"Average Order Processing Time: {average_processing_time}")
```

```
Average Order Processing Time: 1 days 13:16:02.473785613
```

In [39]:

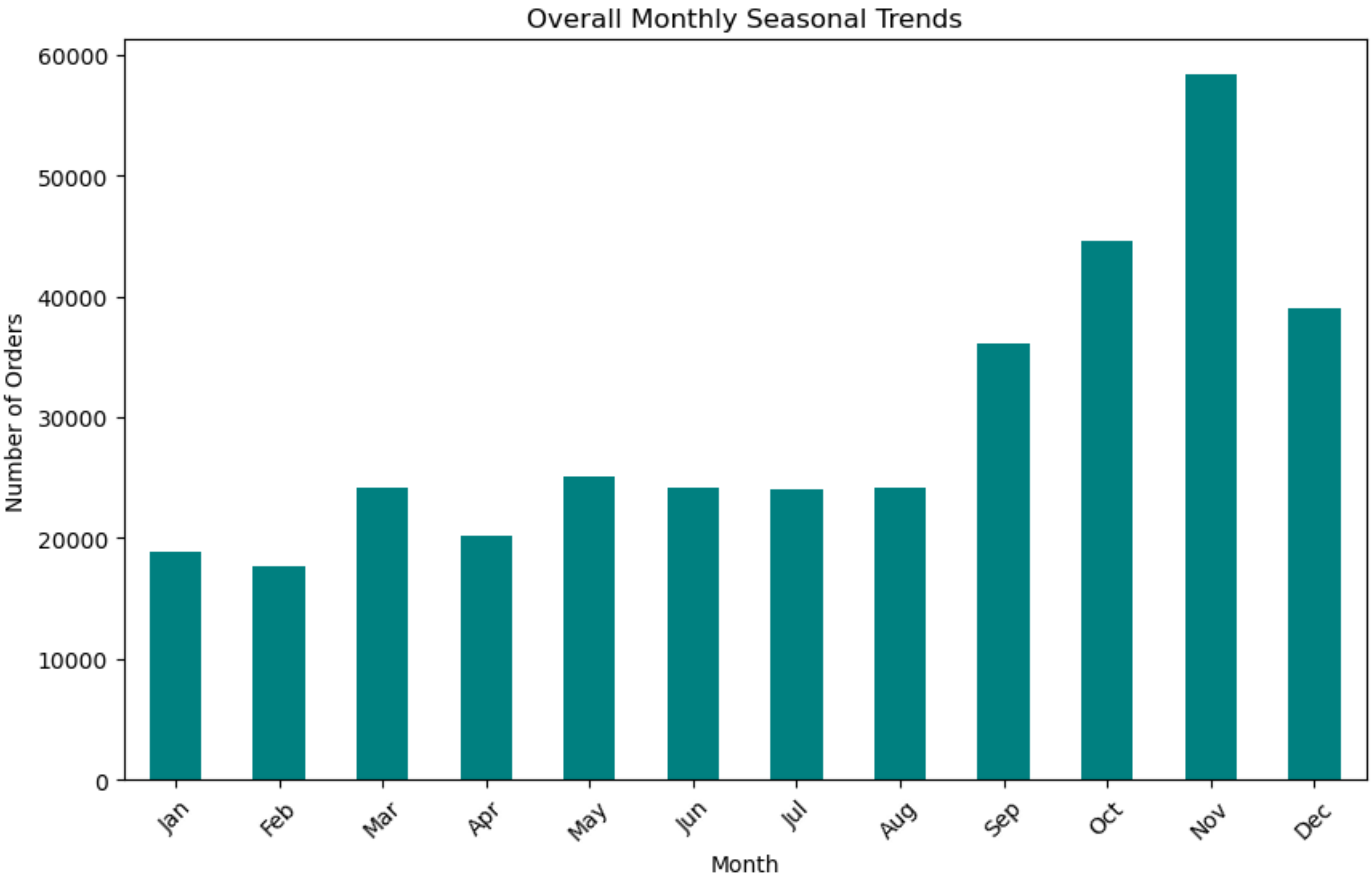
```python
# Ensure InvoiceDate is in datetime format
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

# Extracting week, month, and year
df['Week'] = df['InvoiceDate'].dt.isocalendar().week
df['Month'] = df['InvoiceDate'].dt.month
df['Year'] = df['InvoiceDate'].dt.year

# Aggregating data for weekly, monthly, and yearly trends
weekly_trends = df.groupby(['Year', 'Week']).size()
monthly_trends = df.groupby(['Year', 'Month']).size()
yearly_trends = df.groupby('Year').size()

# Visualization
plt.figure(figsize=(18, 12))

# Weekly Trends
plt.subplot(3, 1, 1)
weekly_trends.unstack(level=0).plot(ax=plt.gca())
plt.title('Weekly Order Trends')
plt.xlabel('Week')
plt.ylabel('Number of Orders')

# Monthly Trends
plt.subplot(3, 1, 2)
monthly_trends.unstack(level=0).plot(ax=plt.gca())
plt.title('Monthly Order Trends')
plt.xlabel('Month')
plt.ylabel('Number of Orders')

# Yearly Trends
plt.subplot(3, 1, 3)
yearly_trends.plot(kind='bar', ax=plt.gca())
plt.title('Yearly Order Trends')
plt.xlabel('Year')
plt.ylabel('Number of Orders')

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



Weekly Order Trends
Number of Orders
Week
Year
2010
2011
Monthly Order Trends
Number of Orders
Month
Year
2010
2011
Yearly Order Trends
Number of Orders
Year
2010
2011

In [40]:

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

# Extracting month for seasonal analysis (ignoring the year)
df['Month'] = df['InvoiceDate'].dt.month

# Aggregating data for monthly trends over the entire timeframe
overall_monthly_trends = df.groupby('Month').size()

# Visualization
plt.figure(figsize=(10, 6))
overall_monthly_trends.plot(kind='bar', color='teal')
plt.title('Overall Monthly Seasonal Trends')
plt.xlabel('Month')
plt.ylabel('Number of Orders')
plt.xticks(ticks=range(0, 12), labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov
plt.show()
```



5. Geographical Analysis

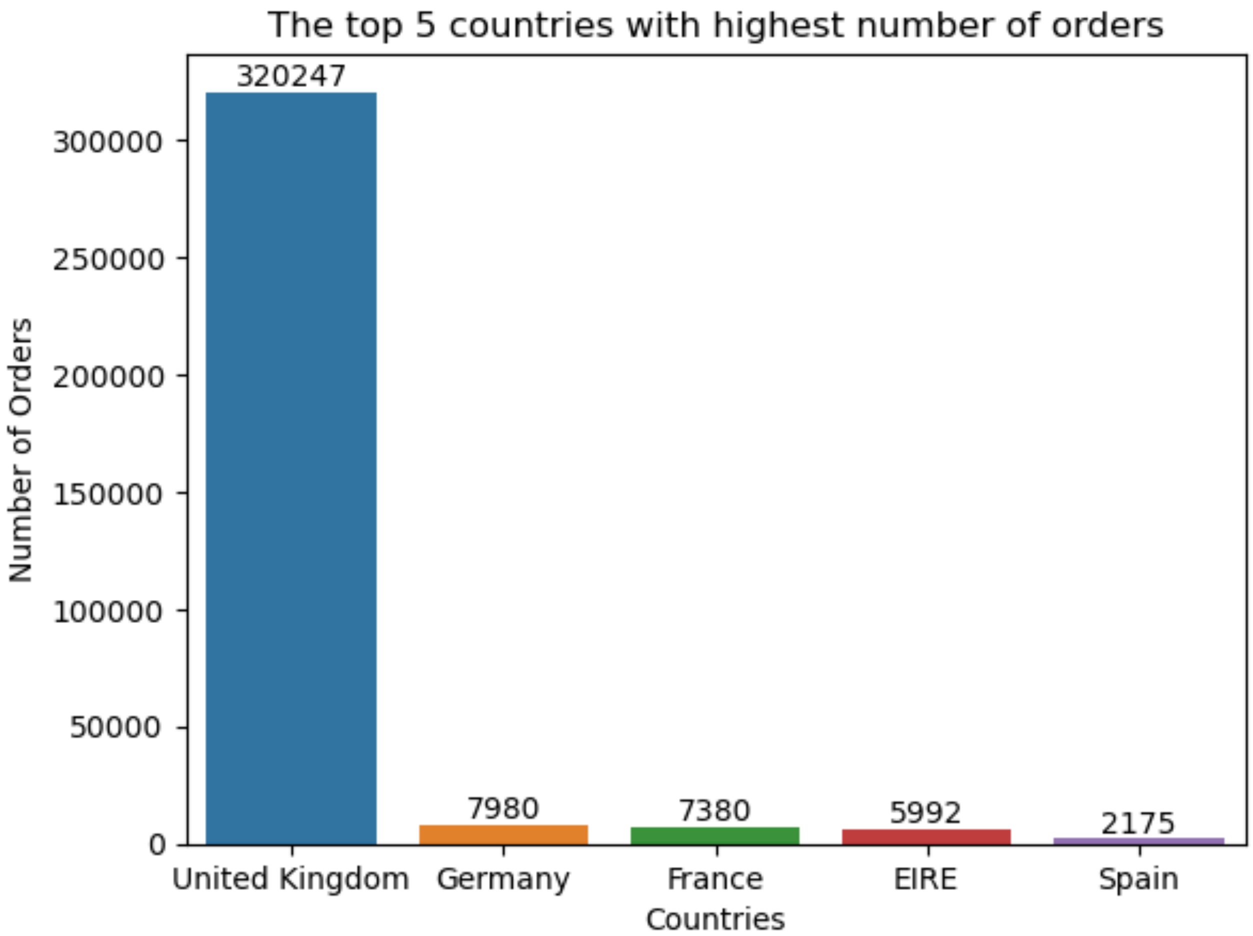
The top 5 countries with the highest number of orders

In [41]:
```python
# the top 5 countries with the highest number of orders
country_orders = df.groupby('Country')['CustomerID'].count().reset_index()
country_orders.rename(columns={'CustomerID':'Number of orders'},inplace=True)
top_5_cnt = country_orders.sort_values(by='Number of orders',ascending=False)[:5]
top_5_cnt
```

Out[41]:

	Country	Number of orders
35	United Kingdom	320247
14	Germany	7980
13	France	7380
10	EIRE	5992
30	Spain	2175

In [42]:
```python
# Visualization of top 5 countries with highest number of orders
sns.barplot(data=top_5_cnt,x='Country',y='Number of orders')
for index, value in enumerate(top_5_cnt['Number of orders']):
    plt.text(index, value, str(value), ha='center', va='bottom')

plt.title('The top 5 countries with highest number of orders')
plt.xlabel('Countries')
plt.ylabel('Number of Orders')
plt.grid(False)
plt.show();
```



Correlation between the country of the customer and the average order value

In [43]:

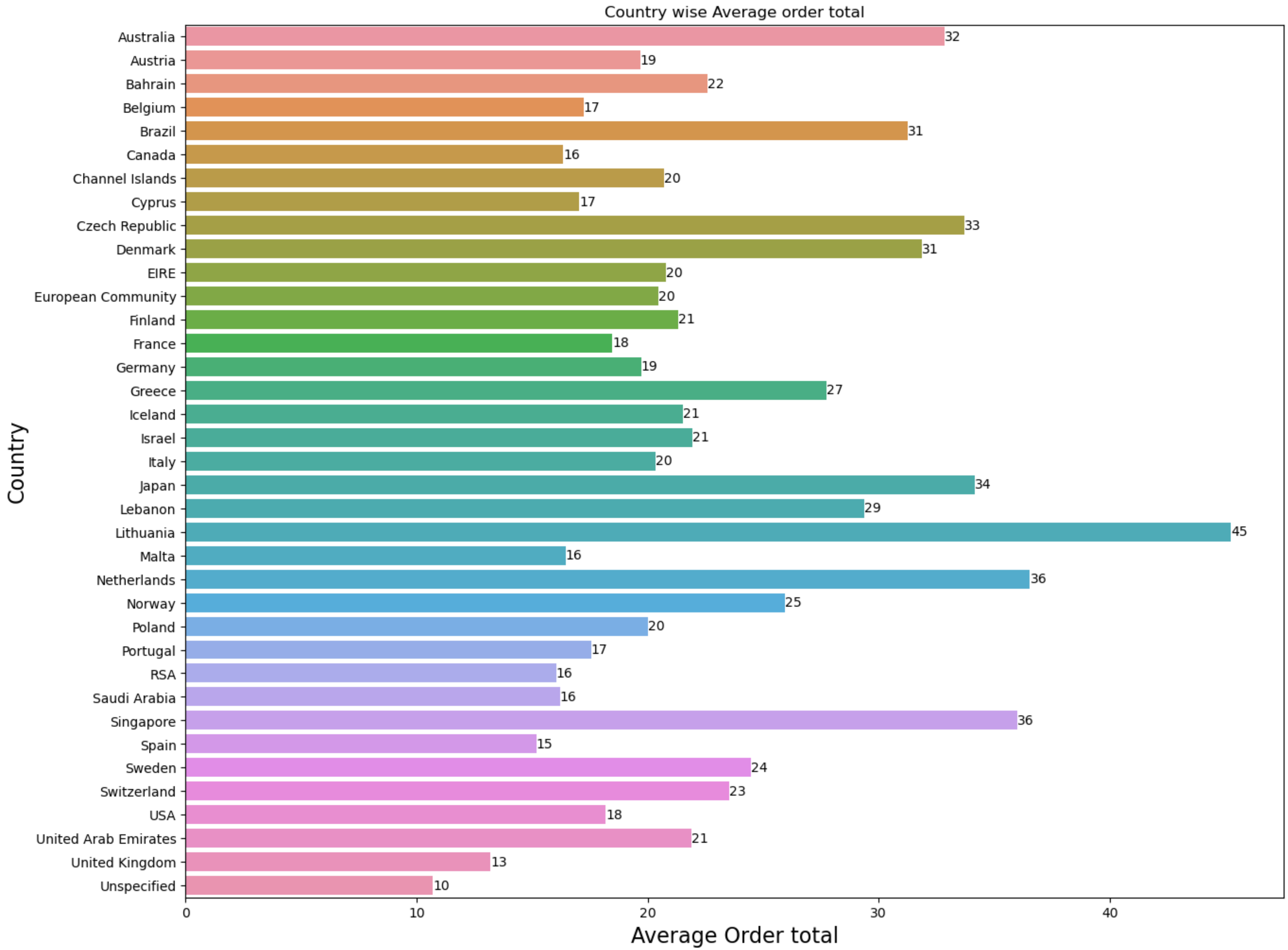
```
# the country of the customer and the average order value
df['Total_Amount'] = df['Quantity']*df['UnitPrice']

cust_od = df.groupby('Country')['Total_Amount'].mean().reset_index()
cust_od.rename(columns={'Total_Amount':'Average order val'},inplace=True)
cust_od.head()
```

Out[43]:

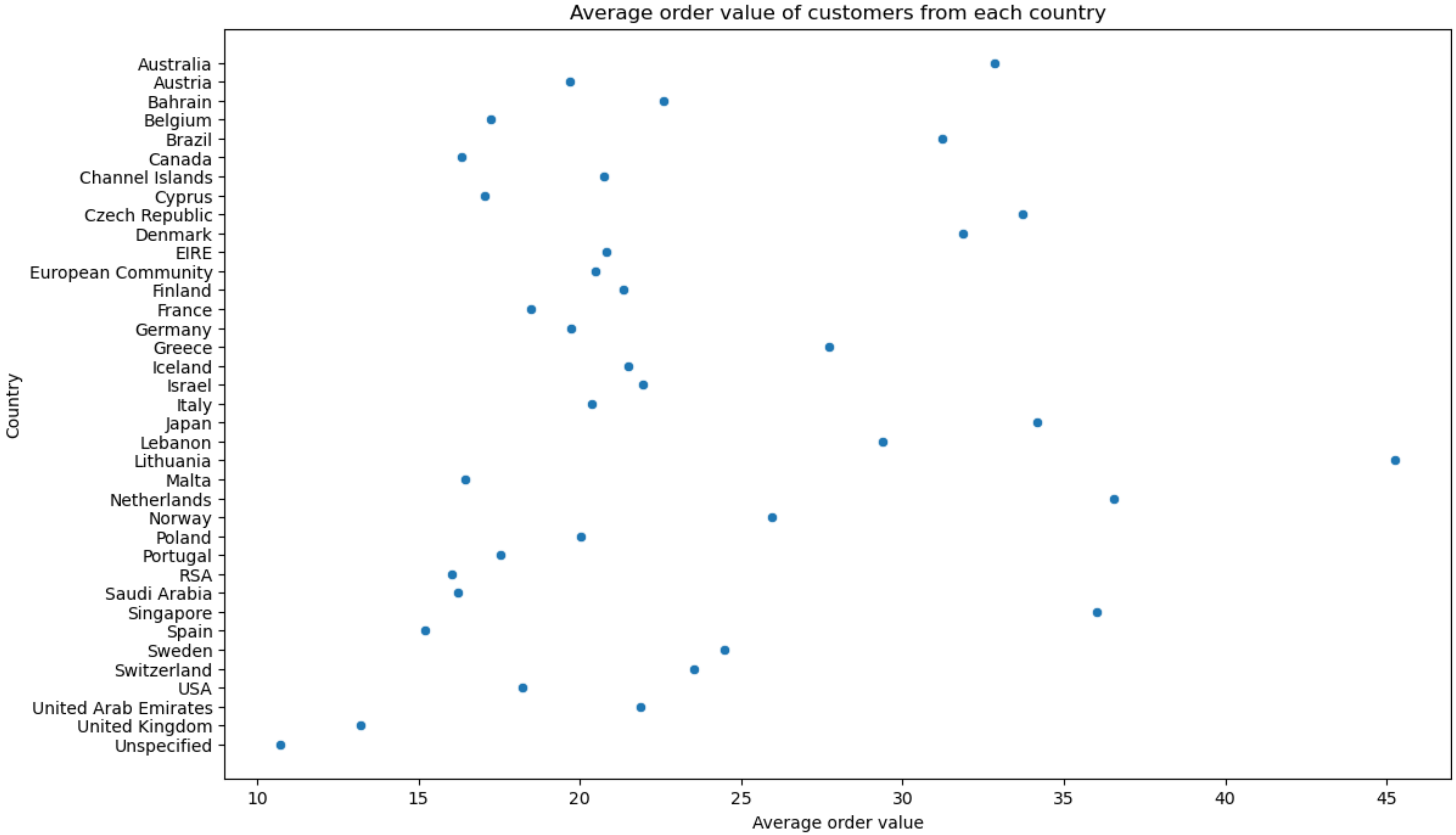
	Country	Average order val
0	Australia	32.864146
1	Austria	19.671917
2	Bahrain	22.572727
3	Belgium	17.233345
4	Brazil	31.238710

In [44]:

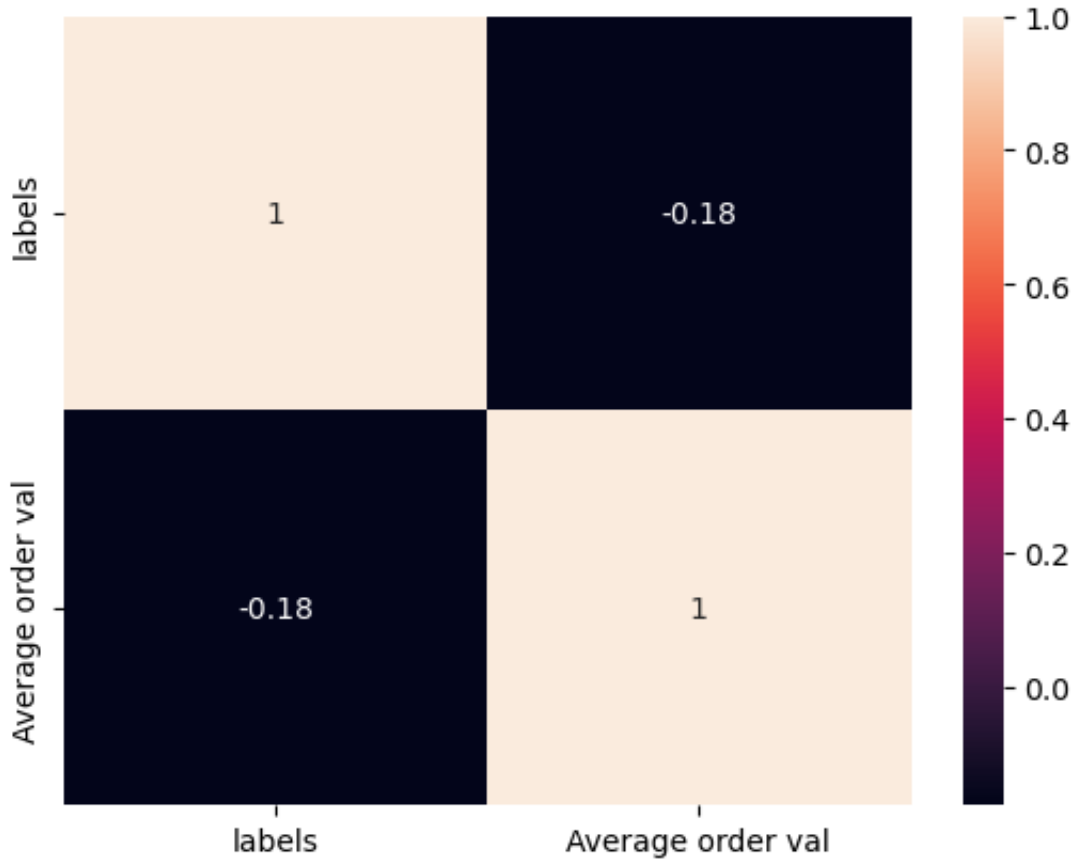
```
# bar chart of country wise average order value
plt.figure(figsize=(15,12))
cust_plt = sns.barplot(data=cust_od, x='Average order val',y='Country',orient='h')
for bar in cust_plt.patches:
    plt.text(
     bar.get_width(),
     bar.get_y() + bar.get_height() / 2,
     f'{int(bar.get_width())}',
     va='center')

plt.title('Country wise Average order total')
plt.xlabel('Average Order total', fontsize = 16)
plt.ylabel('Country', fontsize = 16)
plt.grid(False)
plt.show();
```


Country wise Average order total
Australia
Austria
Bahrain
Belgium
Brazil
Canada
Channel Islands
Cyprus
Czech Republic
Denmark
EIRE
European Community
Finland
France
Germany
Greece
Iceland
Israel
Italy
Japan
Lebanon
Lithuania
Malta
Netherlands
Norway
Poland
Portugal
RSA
Saudi Arabia
Singapore
Spain
Sweden
Switzerland
USA
United Arab Emirates
United Kingdom
Unspecified
32
19
22
17
31
16
20
17
33
31
20
20
21
18
19
27
21
21
20
34
29
45
16
36
25
20
17
16
16
36
15
24
23
18
21
13
10
Country
0
10
20
30
40
Average Order total

In [45]:
```
# scatter chart of country wise average order value
plt.figure(figsize=(13,8))
sns.scatterplot(data=cust_od,x='Average order val',y='Country')
plt.xlabel('Average order value')
plt.ylabel('Country')
plt.title('Average order value of customers from each country')
plt.grid(False)
plt.show();
```



In [46]:
```
# heatmap to visualise correlation between the country of the customer and the average order value
from pandas import factorize

labels, categories = factorize(cust_od["Country"])
cust_od["labels"] = labels
corr_matrix = cust_od[['labels','Average order val']].astype(float).corr()
corr_matrix
sns.heatmap(corr_matrix,annot=True)
plt.show();
```



We can conclude from the bar chart, scatter plot and heatmap generated that there is minimal to no correlation (-0.1) between the country of the customer and the average order value

6. Customer Behavior

- How long, on average, do customers remain active (between their first and last purchase)?

In [47]:
```
# Group by customer ID and calculate the duration of customer activity
cust_duration = df.groupby('CustomerID')['InvoiceDate'].apply(lambda x:(x.max()-x.min()).days).reset_index()
cust_duration.rename(columns={'InvoiceDate':'Active_duration'},inplace=True)

# average duration of customer activity
avg_cust_duration = round(cust_duration['Active_duration'].mean())

print(f'The Average active Duration of Customer: {avg_cust_duration} days')
```

```
The Average active Duration of Customer: 129 days
```

Conclusion: On Average, The Customers remain active for 130 days

- Are there any customer segments based on their purchase behavior?

Analyzing customer segments

1) Checking the customers based on their activity duration

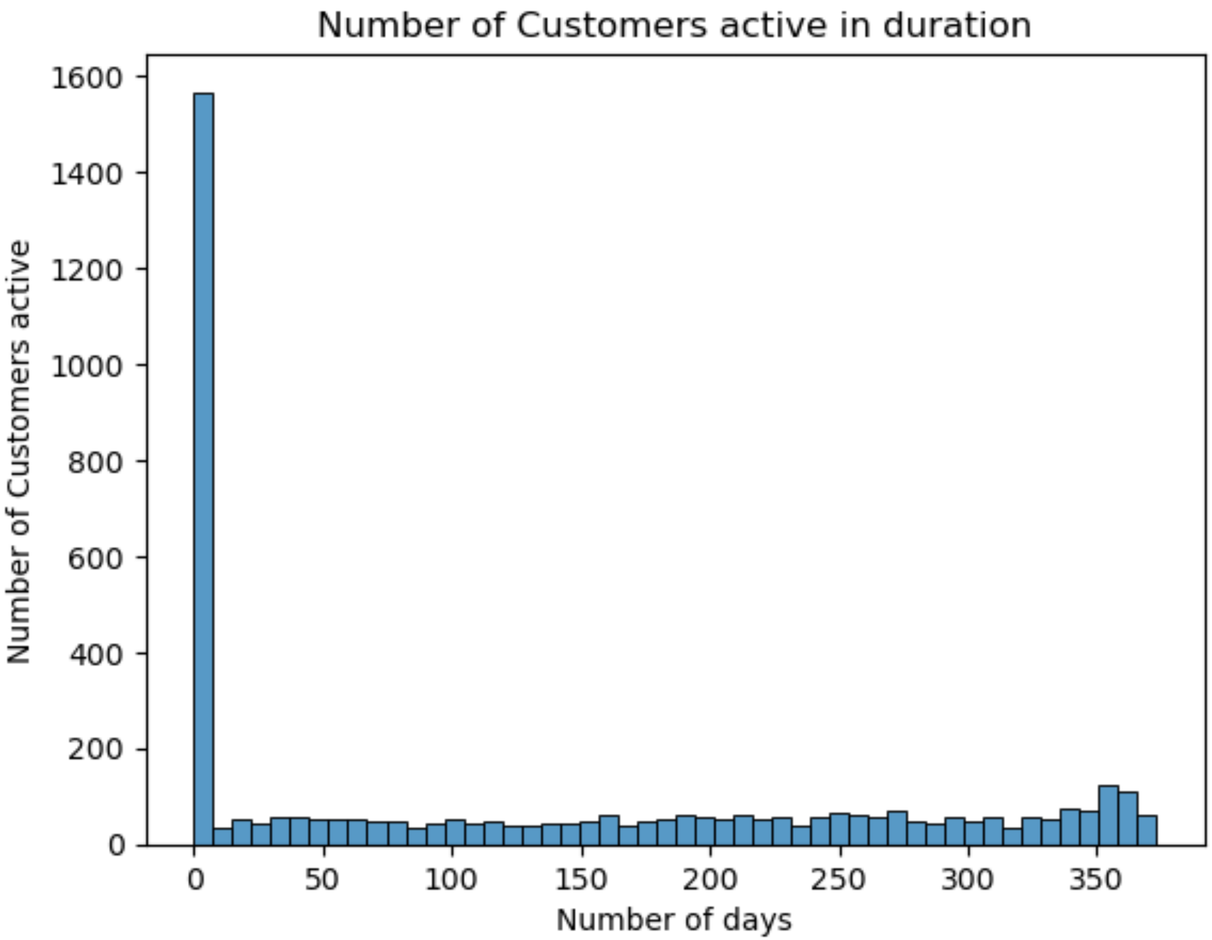
In [48]:
```
cust_duration.sample(6)
```

Out[48]:

	CustomerID	Active_duration
3475	17223.0	58
4061	18064.0	0
444	12935.0	317
3240	16878.0	0
786	13426.0	358
1037	13781.0	0

In [49]:

```
# histogram of active duration of customers
sns.histplot(cust_duration['Active_duration'],bins=50)
plt.xlabel('Number of days')
plt.ylabel('Number of Customers active')
plt.title('Number of Customers active in duration')
plt.grid(False)
plt.show();
```



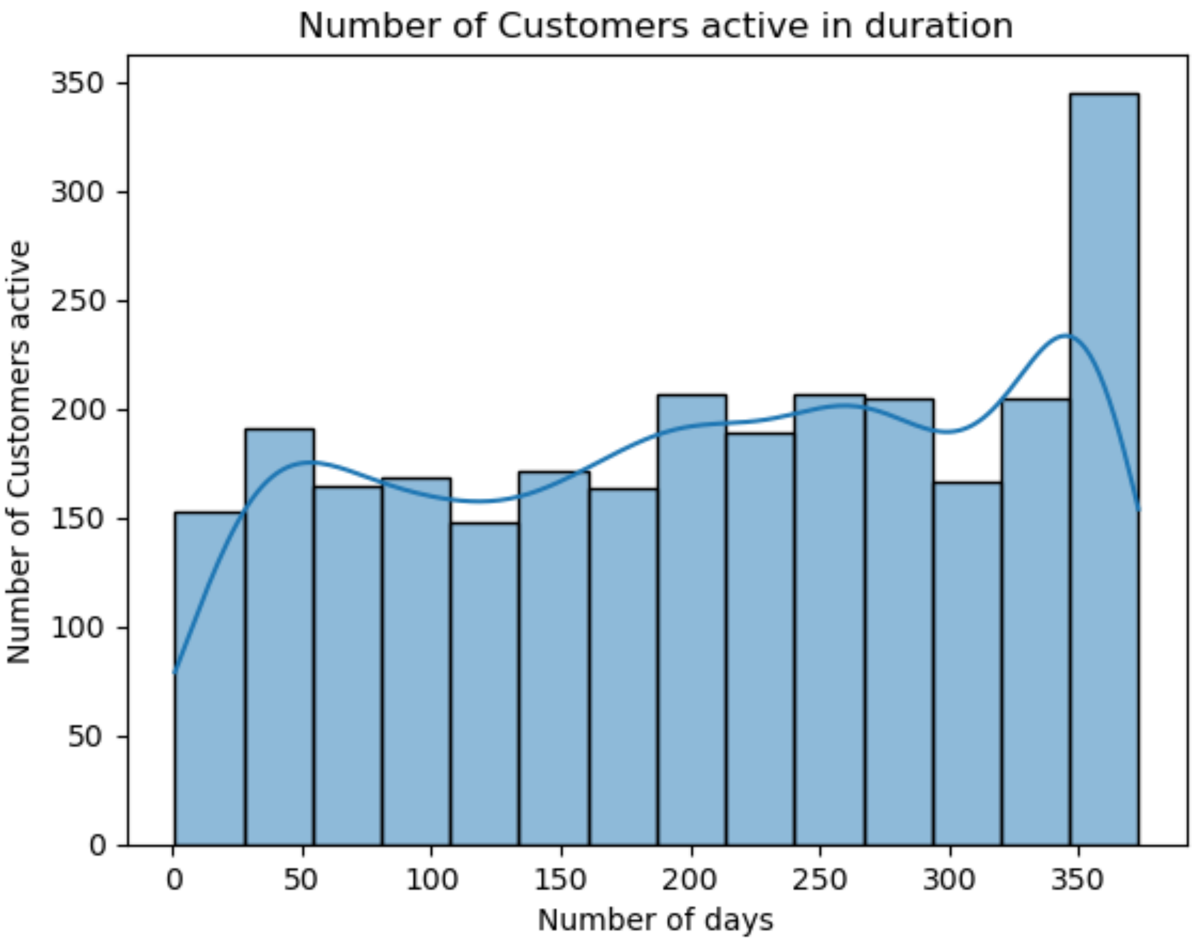
Data seems to be an skewed, Removing the customers who only has bought once that means active duration is 0

In [50]:
```
# for better understanding of data removing the customers who has bought only once
one_time_cust = cust_duration[cust_duration['Active_duration']==0]
mask = cust_duration['Active_duration']==0
cust_duration_1 = cust_duration[~mask]
```

In [51]:
```
print(one_time_cust.shape)
print(cust_duration_1.shape)
```

```
(1533, 2)
(2682, 2)
```

In [52]:
```
# histogram of active duration of customers (removing 1 time customers)
sns.histplot(cust_duration_1['Active_duration'],bins=14,kde=True)
plt.xlabel('Number of days')
plt.ylabel('Number of Customers active')
plt.title('Number of Customers active in duration')
plt.grid(False)
plt.show();
```



Data seems to be distributed fairly now

2) Customer segmentation based on the RFM score. Ranking the customers based on the RFM score calculated

Calculating the Recency, Frequency, Monetary for all customers

In [53]:
```
# calculate RFM matrix
curr_date = df['InvoiceDate'].max()

# Recency of each customer
recency = df.groupby('CustomerID')['InvoiceDate'].max().reset_index()
recency['Recency'] = (curr_date - recency['InvoiceDate']).dt.days

# frequency of each customer
frequency = df.groupby('CustomerID')['InvoiceDate'].count().reset_index()
frequency.columns = ['CustomerID', 'Frequency']

# Monetary value of each customer
monetary = df.groupby('CustomerID')['Total_Amount'].sum().reset_index()
monetary.columns = ['CustomerID', 'Monetary']
```

In [54]:
```
# Merging of dataframes to get RFM matrix for each customers
rfm = pd.merge(recency[['CustomerID','Recency']],frequency,on='CustomerID')
rfm = pd.merge(rfm,monetary,on='CustomerID')
rfm.head()
```

Out[54]:

	CustomerID	Recency	Frequency	Monetary
0	12347.0	1	174	3743.43
1	12348.0	248	6	90.20
2	12349.0	18	66	1287.15
3	12350.0	309	16	294.40
4	12352.0	35	71	1232.44

Assigning rank to each customer based on the RFM score

In [55]:
```
# assigning rank to each customer based on the RFM score
r, f, m = range(10, 0, -1), range(1,11), range(1,11)

rfm['r_score'] = pd.qcut(rfm['Recency'], q=10, labels=r).astype(int)
rfm['f_score'] = pd.qcut(rfm['Frequency'], q=10, labels=f).astype(int)
rfm['m_score'] = pd.qcut(rfm['Monetary'], q=10, labels=m).astype(int)

rfm['rfm_sum'] = rfm['r_score'] + rfm['f_score'] + rfm['m_score']
```

In [56]:
```
cust_rating = rfm.sort_values(by='rfm_sum',ascending=False)
cust_rating.sample(6)
```

Out[56]:

	CustomerID	Recency	Frequency	Monetary	r_score	f_score	m_score	rfm_sum
1136	13926.0	22	9	166.00	8	2	2	12
2170	15368.0	20	3	167.70	8	1	2	11
2447	15757.0	65	43	688.00	5	6	6	17
407	12876.0	57	63	1192.62	5	7	8	20
2126	15307.0	95	8	135.10	4	2	2	8
1284	14130.0	318	64	425.16	1	7	5	13

Assigning label to each customer, based on the RFM score calculated.

In [57]:
```
# function to assign label to each customer based on RFM score calculated
def assign_label(x):
    if x >= 25:
        return 'High value customer'
    elif x>=15 and x<25:
        return 'Medium value customer'
    else:
        return 'Low value customer'

cust_rating['Rating'] = cust_rating['rfm_sum'].apply(assign_label)
cust_rating.sample(6)
```

Out[57]:

	CustomerID	Recency	Frequency	Monetary	r_score	f_score	m_score	rfm_sum	Rating
2256	15494.0	20	116	1186.49	8	9	8	25	High value customer
843	13504.0	63	19	260.48	5	3	3	11	Low value customer
2877	16369.0	17	113	1468.99	8	9	8	25	High value customer
1374	14257.0	63	96	2467.50	5	8	9	22	Medium value customer
755	13375.0	88	33	607.18	4	5	6	15	Medium value customer
2148	15341.0	80	9	803.16	4	2	7	13	Low value customer

Analyzing customer segments we formed

In [58]:

```
# Displaying the the scores segment wise
ratings = cust_rating.groupby('Rating').agg({'CustomerID':'count',
                                             'Recency':'mean',
                                             'Frequency':'mean',
                                             'Monetary':'sum'}).reset_index()
ratings.rename(columns={'Rating':'Customer segment','CustomerID':'Total Customers'},inplace=True)
ratings
```

Out[58]:

	Customer segment	Total Customers	Recency	Frequency	Monetary
0	High value customer	753	11.492696	279.146082	2912699.210
1	Low value customer	1794	160.557414	17.123188	445507.182
2	Medium value customer	1668	52.863909	69.067746	1597913.812

Normalising the Total customers and Monetary columns to get better interpretation of each customer segment and their contribution

In [59]:

```
ratings['Customers (%)'] = (ratings['Total Customers']/ratings['Total Customers'].sum())*100
ratings['Monetary (%)'] = (ratings['Monetary']/ratings['Monetary'].sum())*100
ratings['Monetary'] = ratings['Monetary'] / ratings['Total Customers']
ratings.drop(columns='Total Customers',inplace=True)
ratings.round(2)
```

Out[59]:

	Customer segment	Recency	Frequency	Monetary	Customers (%)	Monetary (%)
0	High value customer	11.49	279.15	3868.13	17.86	58.77
1	Low value customer	160.56	17.12	248.33	42.56	8.99
2	Medium value customer	52.86	69.07	957.98	39.57	32.24

The high value customer segment contributes most in terms of Money spending.

- **High value customers -** The 17.84% Customers of the total customers spends 58.53 % of the total money. Each High valued customer on average spends 3857 $.

- **Medium value customers -** The 39.53% Customers of the total customers spends 32.46 % of the total money. Each Medium valued customer on average spends 965 $.

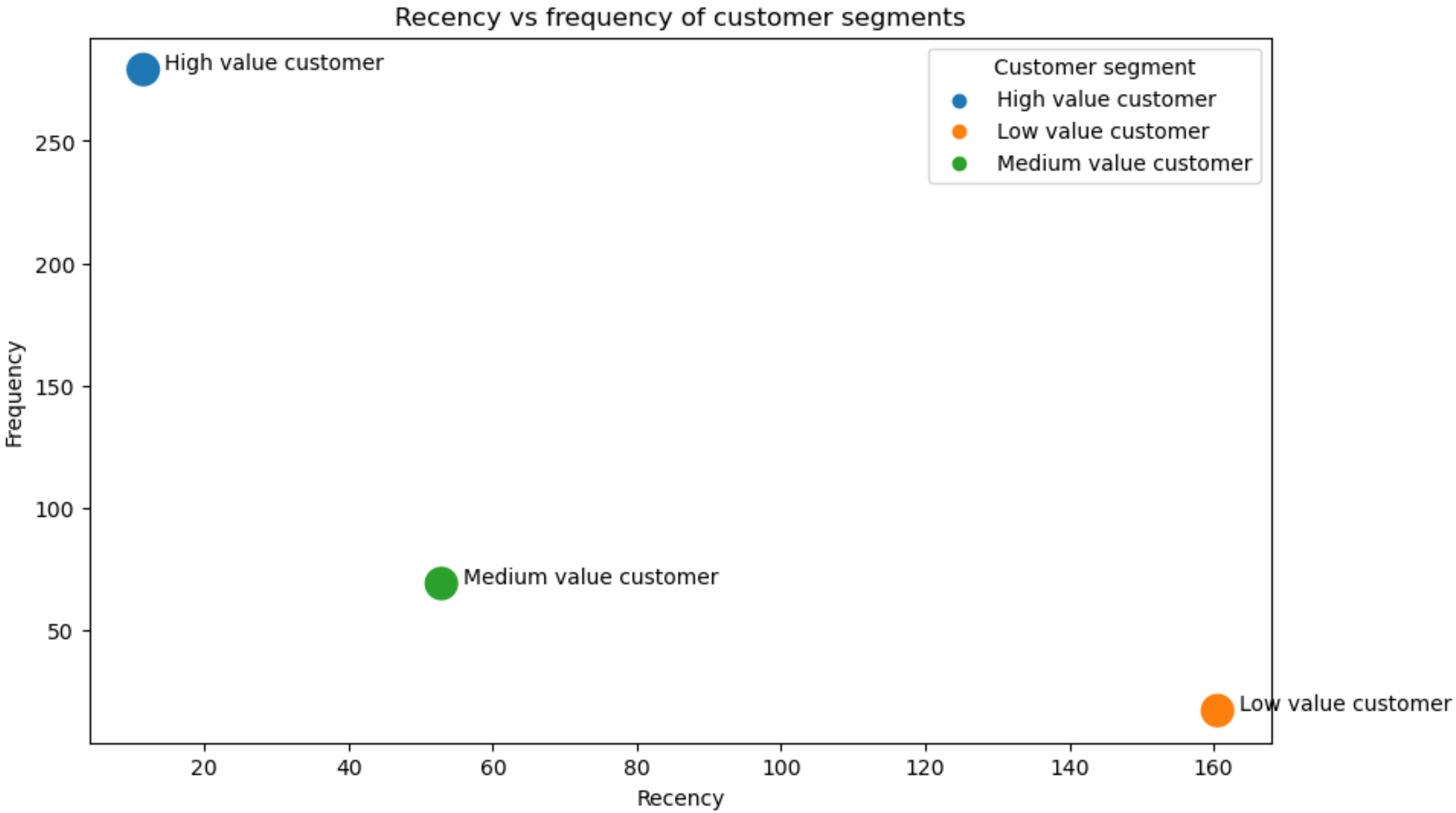
- **Low value customers -** The 42.63% Customers of the total customers spends 9.02 % of the total money. Each Low valued customer on average spends 248 $.

Scatter Plot to visualize the customer segments.

- Low valued customers: low Frequency and high Recency
- Medium valued customers: Medium Frequecy and Medium Recency
- High valued customers: High Frequecy and Low Recency

In [60]:

```
# Scatter Plot to visualize the customer segments
fig, ax = plt.subplots(figsize=(10,6))
plot = sns.scatterplot(x='Recency', y='Frequency', data=ratings, hue='Customer segment', s=300)

for i in range(len(ratings)):
     plot.text(ratings['Recency'][i]+3,
               ratings['Frequency'][i],
               ratings['Customer segment'][i])

ax.set_title('Recency vs frequency of customer segments')
plt.grid(False)
plt.show()
```



3) Applying Clustering method to analyze the customer segments

K-Means Clustering: It is a Non-parametric approach that groups the data points based on their similarity or closeness to each other and then forms K clusters from n observations.

Standardising the data:

In [61]:
```
# Standardising the data
from sklearn.preprocessing import StandardScaler

rfm = rfm[['Recency','Frequency','Monetary']]
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm)
```

In [62]:
```
rfm_data = pd.DataFrame(rfm_scaled,columns=['Recency','Frequency','Monetary'])
rfm_data.head()
```

Out[62]:

	Recency	Frequency	Monetary
0	-0.903680	0.426491	0.991960
1	1.567910	-0.373974	-0.419419
2	-0.733570	-0.088094	0.043007
3	2.178302	-0.326328	-0.340529
4	-0.563461	-0.064271	0.021871

Finding the optimal number of clusters using silhouette score