Customer segmentation using RFM Analysis

IE6400 – FOUNDATIONS OF DATA ANALYTICS

Project Report 2

Group Number 14

Arun Solaiappan Valliappan (00201943)

Nidhi Mallikarjun (002051677)

Preeti Purnimaa Kannan (002374503)

Introduction:

In this project, we delve into the vast world of customer behavior in the hopes of discovering valuable insights that can transform marketing and customer engagement strategies. Our focal point is the rigorous application of the RFM (Recency, Frequency, Monetary) analysis method – an established framework empowering businesses to discern and categorize customers based on their recent purchasing patterns, transaction frequency, and monetary contributions.

Objective:

Our goal is to run RFM analysis on the dataset to generate a Customer Segmentation model. Recency, Frequency, and Monetary (RFM) are three key pillars that show various aspects of customer behaviour. Recency is concerned with how recently a customer made a purchase, Frequency is concerned with the pattern of purchases, and Monetary is concerned with the monetary value of these transactions. We intend to create distinct customer segments by leveraging RFM scores. These segments will not only provide a thorough understanding of customer purchasing patterns, but will also serve as a starting point for customized marketing and customer retention strategies. The ultimate goal is to provide businesses with the knowledge they need to better engage customers and drive long-term growth in the ever-changing eCommerce landscape

Data Source:

The dataset, hosted by the UCI Machine Learning Repository, contains transactions from a UK-based online retail store, spanning from December 2010 to December 2011. It includes data on orders, products, customers, and transactions.

Data Source : https://www.kaggle.com/datasets/carrie1/ecommerce-data

Methodology:

Data Preprocessing: Initial data cleaning, handling missing values, and data transformation to prepare the dataset for analysis.

RFM Metric Calculation: Computation of Recency, Frequency, and Monetary values for each customer to understand purchasing behavior.

Customer Segmentation: Application of K-Means clustering to segment customers based on their RFM scores.

Segment Profiling and Marketing Recommendations: Analysis of each customer segment to identify unique characteristics and formulate targeted marketing strategies.

Tasks performed:

1) Data Overview

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

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

In [5]:
```
# 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
```

Data Preprocessing:

● Import and Cleaning:

Imported the dataset and executed essential data preprocessing steps,

encompassing data cleaning, handling missing values, and converting datatypes as necessary.

```
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]: np.int64(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]: np.int64(10587)
```

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

Out[14]: np.int64(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)
```

RFM Calculation:

- Recency, Frequency, Monetary Metrics: Calculated RFM metrics for each customer:

➢ Recency (R): Determined the days since the customer's last purchase.

➢ Frequency (F): Computed the total number of orders for each customer.

➢ Monetary (M): Summed up the total monetary value of a customer's purchases

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

RFM segmentation:

● Assigning RFM Scores:

Assigned RFM scores based on quartiles or custom-defined bins, facilitating the creation of a single RFM score for each customer.

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)
```

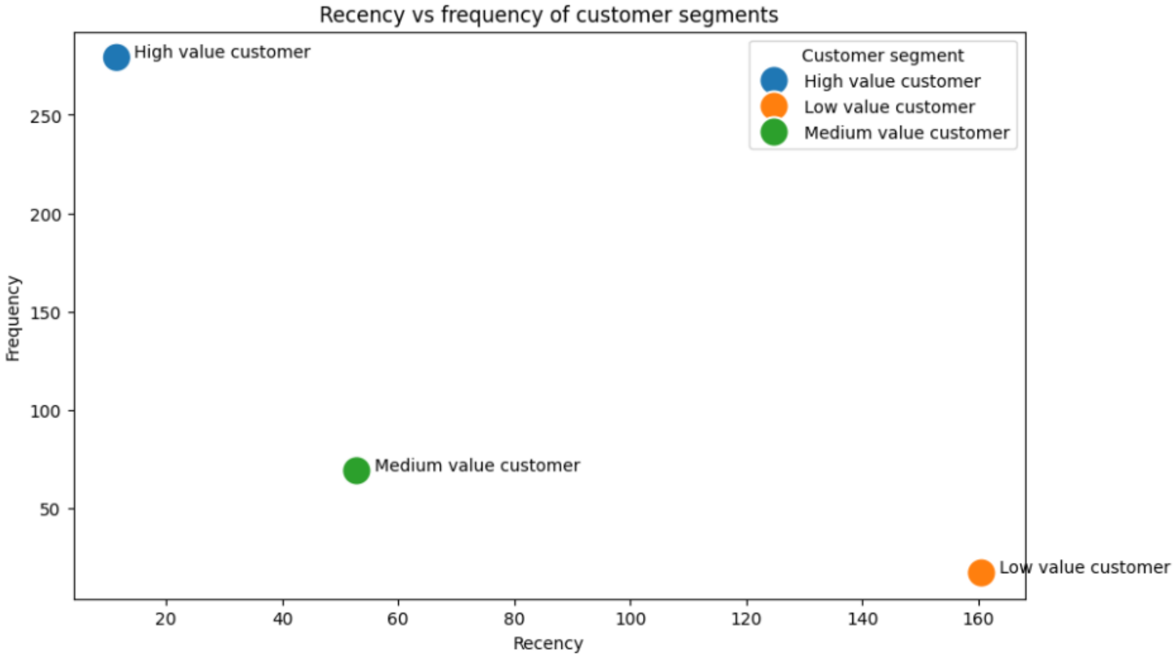
Out[56]:

	CustomerID	Recency	Frequency	Monetary	r_score	f_score	m_score	rfm_sum
1371	14250.0	6	110	1890.13	9	8	9	26
3577	17375.0	198	68	387.35	2	7	5	14
4181	18235.0	71	97	1607.58	4	8	9	21
1192	14001.0	45	30	1001.08	6	5	7	18
4022	18001.0	11	49	281.43	9	6	3	18
375	12831.0	261	8	189.55	2	2	2	6

Scatter Plot to visualize the customer segments.

- Low valued customers: low Frequecy 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()
```



Customer Segmentation:

● Utilizing Clustering Techniques:

Applied clustering techniques, such as K-Means clustering, to segment customers based on their RFM scores. Experimented with different cluster numbers to identify the optimal configuration for meaningful segmentation.

Segment Profiling:

● Analyzing Customer Segments:

Analyzed and profiled each customer segment, elucidating the characteristics of customers within each segment, including RFM scores and other relevant attributes.

Visualization:

● RFM Data Visualization:

Developed visualizations such as bar charts, scatter plots to illustrate the distribution of RFM scores and visualize the formed clusters.

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()
```

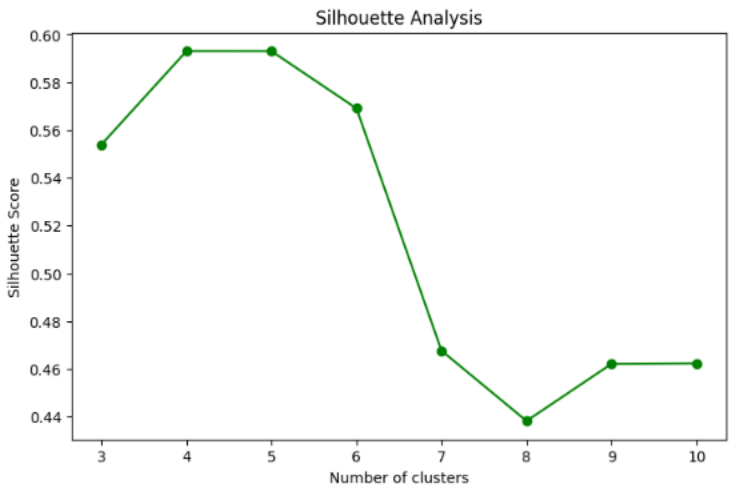
Out[62]:

	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

In [63]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
#from yellowbrick.cluster import silhouette_visualizer

clusters = [3,4,5,6,7,8,9,10]
Silhouette_score = []
#silhouette_score = silhouette_score(rfm_data)

for n_cluster in clusters:
    kmeans = KMeans(n_clusters=n_cluster,random_state=10)
    labels = kmeans.fit_predict(rfm_data)

    score = silhouette_score(rfm_data,labels)
    Silhouette_score.append(score)
    print(f'The Silhouette score for {n_cluster} cluster is: {score}')
    silhouette_val = silhouette_samples(rfm_data, labels)

# Plotting silhouette scores
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 5))
plt.plot(clusters, Silhouette_score, marker='o',linestyle='-',color='green')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Analysis')
plt.grid(False)
plt.show()
```

```
The Silhouette score for 3 cluster is: 0.5539352554820048
The Silhouette score for 4 cluster is: 0.5930336618519508
The Silhouette score for 5 cluster is: 0.5930065332250419
The Silhouette score for 6 cluster is: 0.5690377361529451
The Silhouette score for 7 cluster is: 0.4676275335029238
The Silhouette score for 8 cluster is: 0.4381991522219885
The Silhouette score for 9 cluster is: 0.46210482151841614
The Silhouette score for 10 cluster is: 0.4623138966463448
```

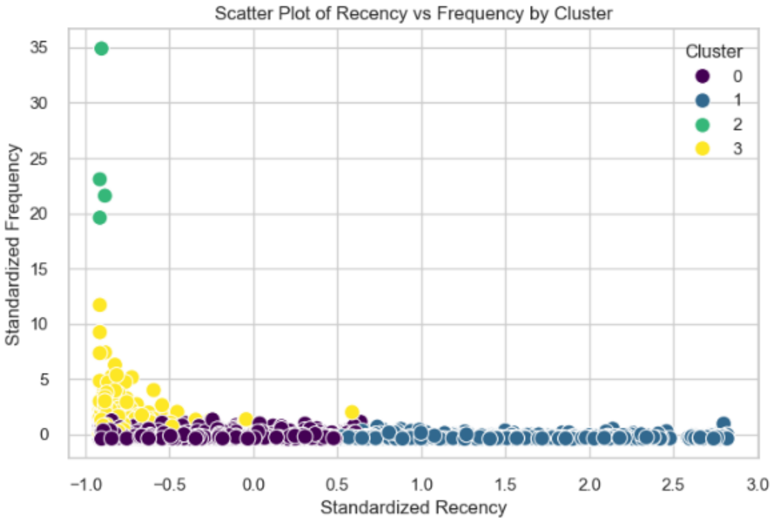


In [65]:
```python
from sklearn.cluster import KMeans

n_clusters = 4
kmeans = KMeans(n_clusters=n_clusters, random_state=10)
rfm_data['Cluster'] = kmeans.fit_predict(rfm_data)
rfm_data.head()
```

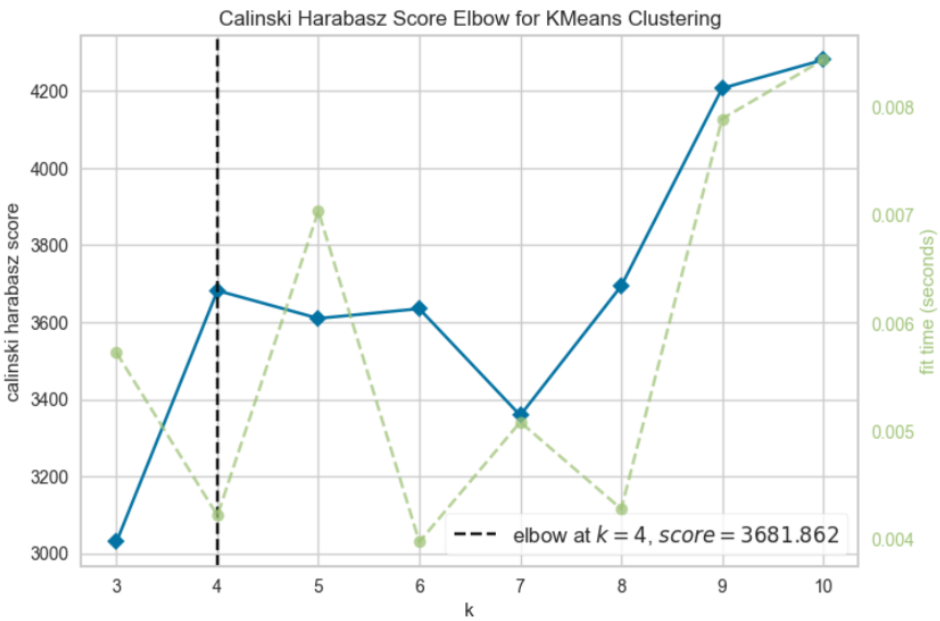
Out[65]:

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

In [66]:
```python
# scatter plot of clusters formed by K-Means clustering
import seaborn as sns
sns.set(style="whitegrid")

# Plot a scatter plot for Recency and Frequency using standardized scales
plt.figure(figsize=(8, 5))
sns.scatterplot(x='Recency', y='Frequency', hue='Cluster', data=rfm_data, palette='viridis', s=100)
plt.title('Scatter Plot of Recency vs Frequency by Cluster')
plt.xlabel('Standardized Recency')
plt.ylabel('Standardized Frequency')
plt.show()
```

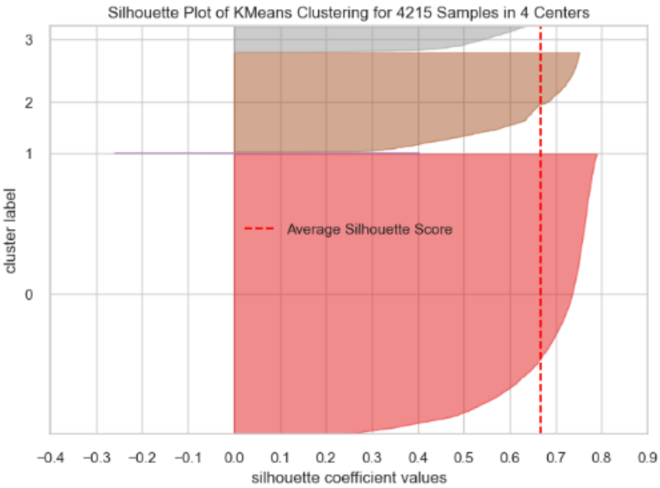


In [64]:
```python
# plot for elbow method (finding the optimal number of clusters)
from yellowbrick.cluster import InterclusterDistance, KElbowVisualizer, SilhouetteVisualizer

model = KMeans()
visualizer = KElbowVisualizer(model, k=(3,11), metric='calinski_harabasz')
visualizer.fit(rfm_data)
visualizer.show();
```

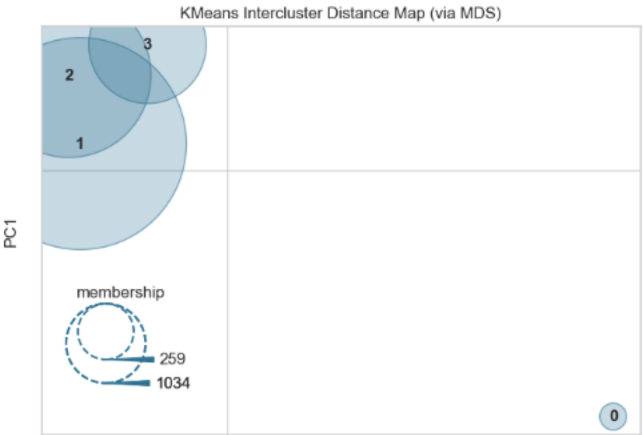


As we can see elbow forms around 4, According to elbow method we can conclude that Optimal value for k is 4

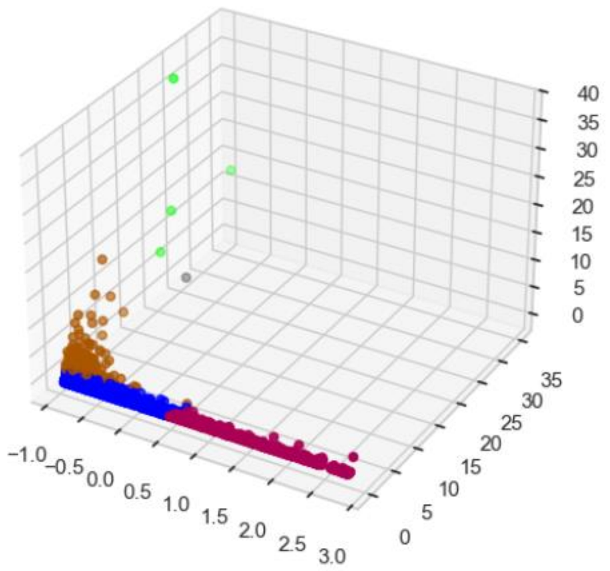
In [68]:
```
#  plot of clustering model
model = KMeans(4)
visualizer = SilhouetteVisualizer(model)
visualizer.fit(rfm_data)
visualizer.show();
```



In [69]:
```
# Plot to visualize intercluster distance
model = KMeans(4)
visualizer = InterclusterDistance(model)
visualizer.fit(rfm_data)
visualizer.show();
```



In [70]:
```
# 3D scatter plot visualize clusters
model = KMeans(4,random_state=42)
model.fit(rfm_data)
centers = model.cluster_centers_
predict = model.predict(rfm_data)
figure = plt.figure()
ax = figure.add_subplot(111,projection='3d')
ax.scatter(rfm_data['Recency'],rfm_data['Frequency'],rfm_data['Monetary'],cmap='brg',c=predict)
ax.scatter(centers[:,0],centers[:,1],c='black');
```



Marketing Recommendations:

→Tailored Marketing Strategies:

Based on the visualizations generated, that outline the distribution of customer segments, their RFM (Recency, Frequency, Monetary) scores, and the clustering of these segments, here are actionable marketing recommendations for each customer segment to improve customer retention and maximize revenue:

Normal (Largest Segment):

Strategy: Convert 'Normal' customers into more engaged segments through targeted communications and introductory offers.

Actionable Steps:

Implement a customer education campaign about the benefits of more frequent purchases, offer incentives for frequent shopping, and personalized product recommendations based on browsing behavior.

Loyal Customers:

Strategy: Maintain their high purchase frequency with rewards and recognition. Actionable Steps: Offer a loyalty program with points or discounts, exclusive access to sales, and engage them with brand storytelling to deepen their emotional connection to the brand.

Big Spenders:

Strategy: Encourage larger basket sizes and premium product purchases.

Actionable Steps: Upsell and cross-sell premium products or services, provide bundle offers, and create high-value tailored content that resonates with their willingness to spend more.

Lost Big Spenders:

Strategy: Re-engage to understand their disengagement and reintroduce them to the brand. Actionable Steps: Send out personalized re-engagement campaigns, survey to understand why they left, and offer a we-miss-you discount or gift with purchase to incentivize their return.

Almost Lost:

Strategy: Win back these customers before they lapse with urgency-triggering campaigns. Actionable Steps: Implement a win-back email series with time-sensitive offers, highlight product improvements or new arrivals, and offer a limited-time welcome back discount.

Best Customers:

Strategy: Foster exclusivity and premium experiences to retain their high-value status. Actionable Steps: Provide VIP service, exclusive events or content, early access to new products, and high-engagement experiences that exceed expectations.

Lost Minimal Spenders:

Strategy: Do not invest heavily but keep the door open for their return.Actionable Steps: Include them in mass-market campaigns, offer self-service tools for re-engagement, and maintain a brand presence in their lives without significant marketing spend. The bubble chart of RFM segments suggests varying levels of engagement across different RFM scores

Lastly, the RFM score distribution shows us the spread of customer engagement. We can leverage this information to focus our efforts on moving customers from lower RFM scores to higher ones through personalized marketing strategies that consider their unique behaviors and value to the company.

All strategies should be continuously monitored and optimized based on customer response and feedback to ensure the best return on investment and customer satisfaction

Summary of results:

1.Data Overview:

- The size of the dataset in terms of rows and columns are concluded to be as following:

The number of rows are: 541909

The number of columns are: 8

- Brief description of each column:

- InvoiceNo: A six-digit number storing the details of the transaction.

- StockCode: A code which defines the product which has been sold.

- Description: Product name

- Quantity: The quantities of each product per transaction

- InvoiceDate: Shows the time and day that each transaction was created.

- UnitPrice: Product price per unit.

- CustomerID: A unique number designated to each customer.

- Country: Name of the country where each customer resides.

- Time period covered by this dataset:

The dataset contains data starting from 12/1/2010 - 12/9/2011.

2. Customer Analysis:

- By calculating the number of unique customers using the CustomerID column we could conclude the number of unique customers present in the dataset to be 4215.

- By grouping the DataFrame by 'CustomerID' and calculating the number of unique invoices (orders) for each customer, the top 5 customers are identified as

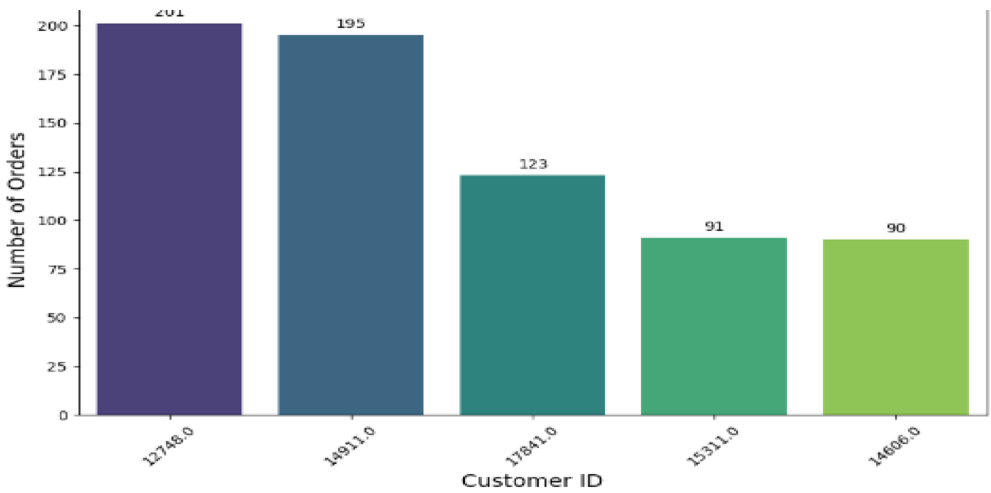
In [28]:
```
# Analyzing Customer Orders
orders_per_customer = df.groupby('CustomerID')['InvoiceNo'].nunique()

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})
```

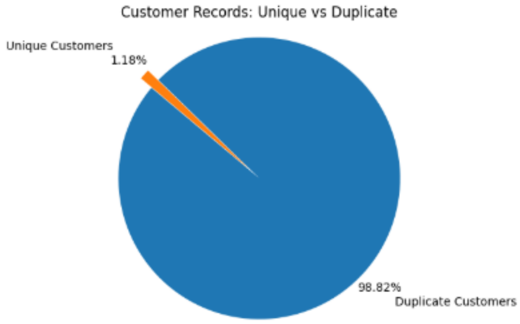
In [30]:
```
top_5_customers_df
```

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 [26]:
```python
# 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()
```



-This bar chart shows the unique and duplicate customer records which is 1.18 and 98.82 percent respectively.

3. Product Analysis:

- Counting the occurrences of each product description has resulted in identifying the top 10 products to be:

3) Product Analysis

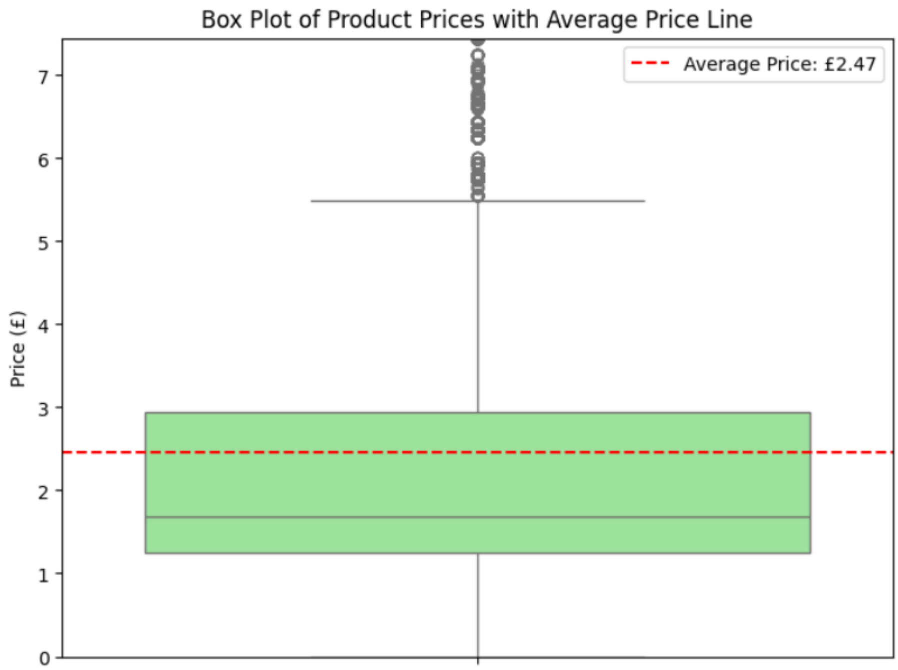
In [32]:
```python
# Top 10 most frequently purchased products
top_10_products = df['Description'].value_counts().head(10)

# 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', np.float64(51472.01))
```

- the below bar chart is the representation of the top 10 most frequently purchased products.

The most and least purchased product is white hanging heart t-light holder and pack of retrospot and cake cases respectively.

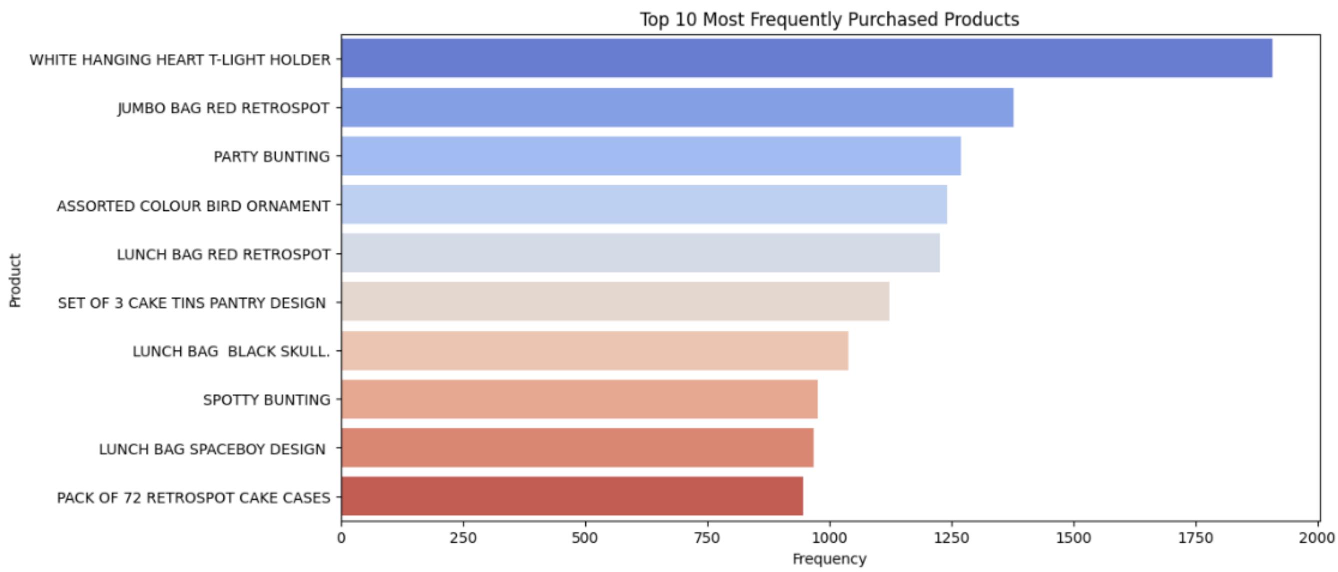
```python
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['UnitPrice'], color='lightgreen')
plt.axhline(y=average_price, color='red', linestyle='--', label=f'Average Price: £{average_price:.2f}')
plt.title('Box Plot of Product Prices with Average Price Line')
plt.ylabel('Price (£)')
plt.ylim(0, df['UnitPrice'].quantile(0.95))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



-Further we calculated the total price of all products in the DataFrame and the total number of products in the DataFrame and used the result to find the average price of a product to be 2.4659.

- The product that generates the highest revenue is: white hanging heart t-light holder'(51472.01)

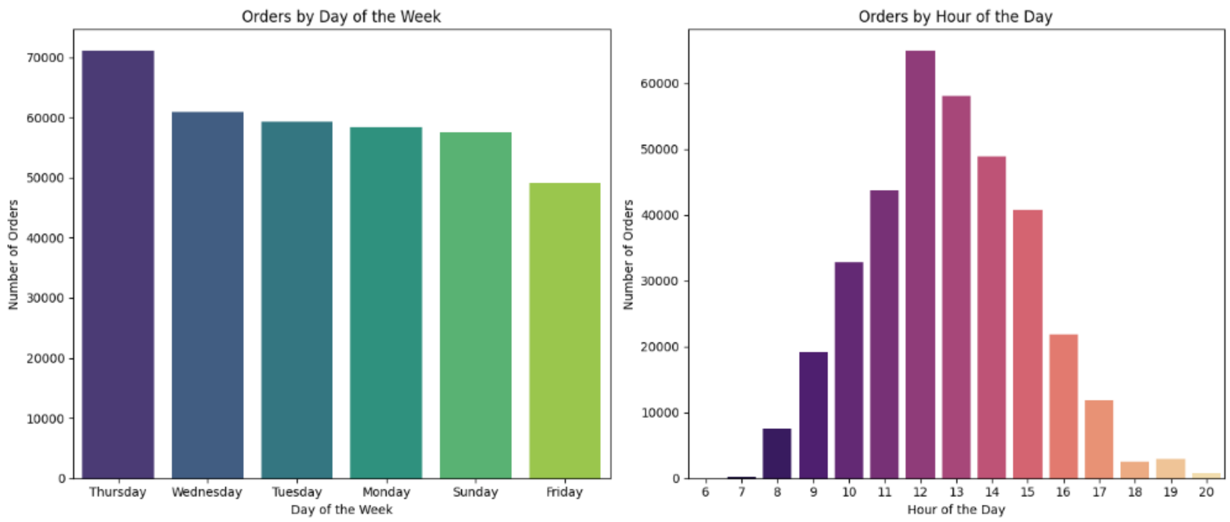
```python
# Plotting bar chart to visualize top 10 most frequently purchased products
plt.figure(figsize=(12, 6))
sns.barplot(x='Frequency', y='Product', data=top_10_products_df, palette='coolwarm')
plt.title('Top 10 Most Frequently Purchased Products')
plt.xlabel('Frequency')
plt.ylabel('Product')
plt.show()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



4. Time Analysis:

- By grouping the DataFrame by 'Day of Week' and 'Hour of Day', and identifying the day and hour with the highest number of orders, we were able to conclude that the day of the week with the highest order count is Thursday and the hour of the day with the highest order count is 12.

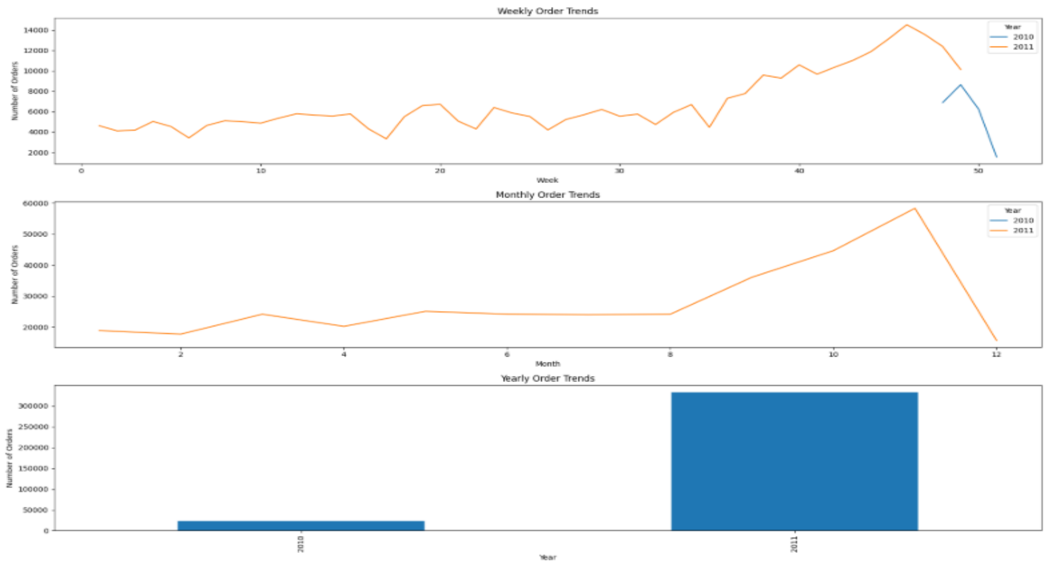
In [37]:
```
# Aggregating data for visualization
day_order_counts = df['DayOfWeek'].value_counts()
hour_order_counts = df['HourOfDay'].value_counts()

# 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()
```



- By creating a line plot to visualize the monthly ,weekly and yearly order counts, we were able to identify the presence of trends in the dataset.

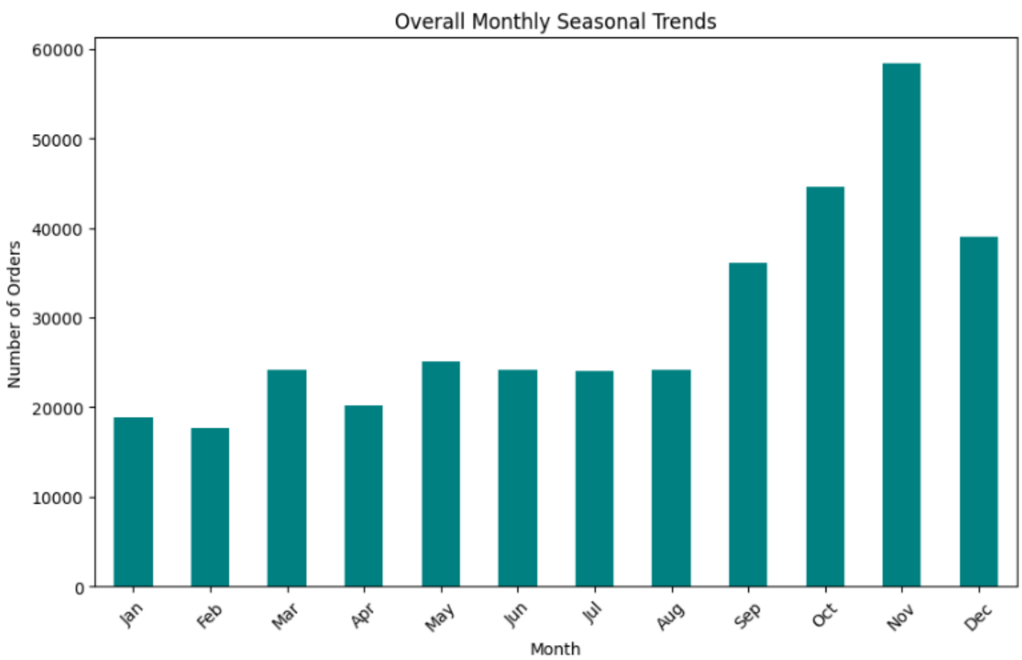
- Using the graph below we can conclude that there is fluctuation in the dataset as we can see that the number of orders is as low as 20000 in the starting months 2 and 4 while it hits a peak of above 60000 between n the months 10 and 12.

```
# 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()
```



In [40]:

```
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', 'Dec'], rota
plt.show()
```



5. Geographical Analysis:

By grouping the DataFrame by 'Country' and calculate the total quantity of orders for each country we identified the top 5 countries with the highest number of orders:

5. Geographical Analysis

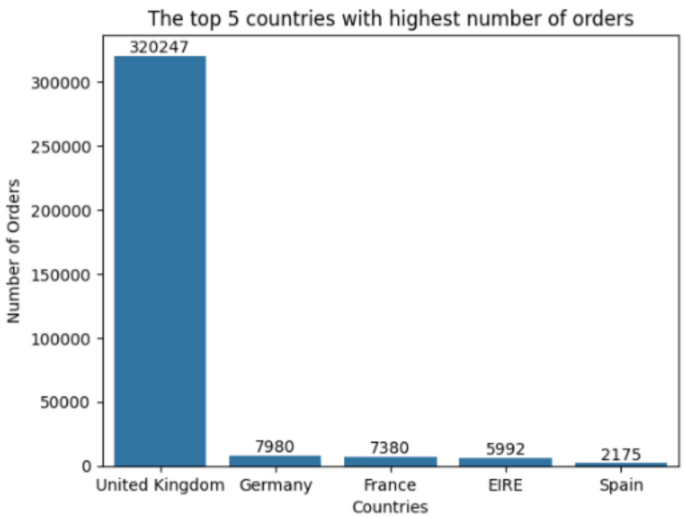
The top 5 countries with the highest number of orders

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

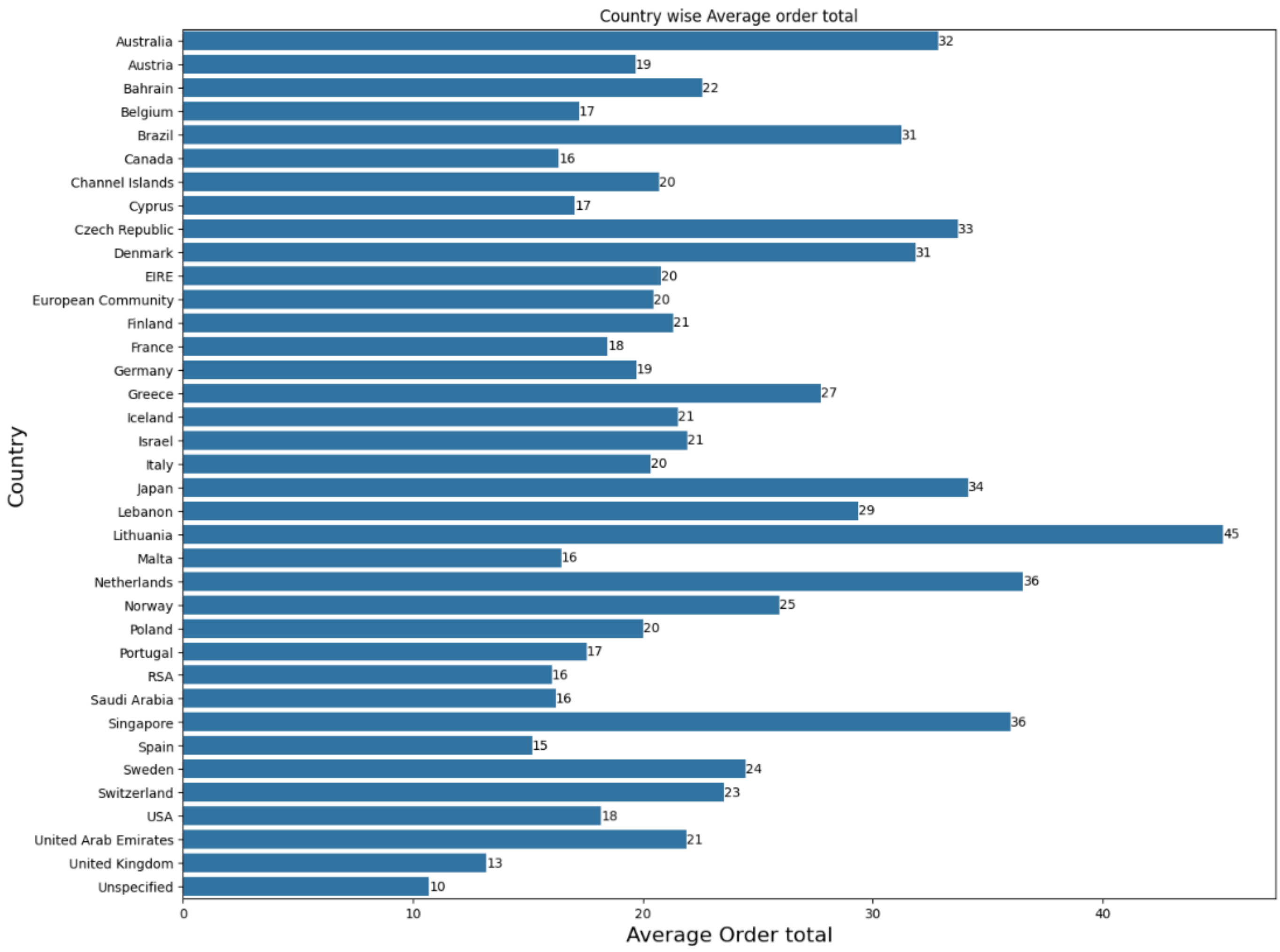
In [42]:
```python
# Visualization of top 5 countries with highest number of orders
sns.barplot(data=top_5_cnt,x='Country',y='Number of orders')
for index, value in enumerate(top_5_cnt['Number of orders']):
    plt.text(index, value, str(value), ha='center', va='bottom')

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



-The bar chart shows the average order total by country wise. Lithuania has the highest order average order total which is 45.

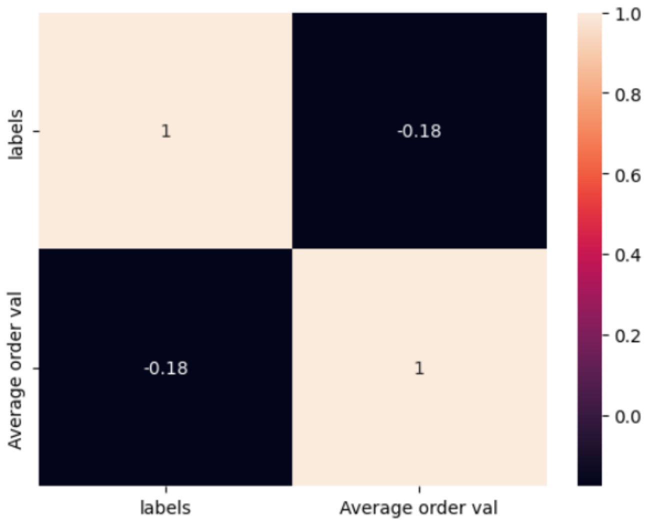
- The heatmap to visualise correlation between the country of the customer and the average order value.

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



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.Payment Analysis:

Unfortunately, there isn't enough data available to provide insights into questions related to payment analysis. The specific details about the most common payment methods used by customers and any potential relationship between payment methods and order amounts are not available for analysis. As a result, without adequate information on payment-related data, it's challenging to draw meaningful conclusions or explanations regarding these aspects of customer behaviour.

7. Customer Behavior:

- we were able to derive that the average duration of customer activity is 129.

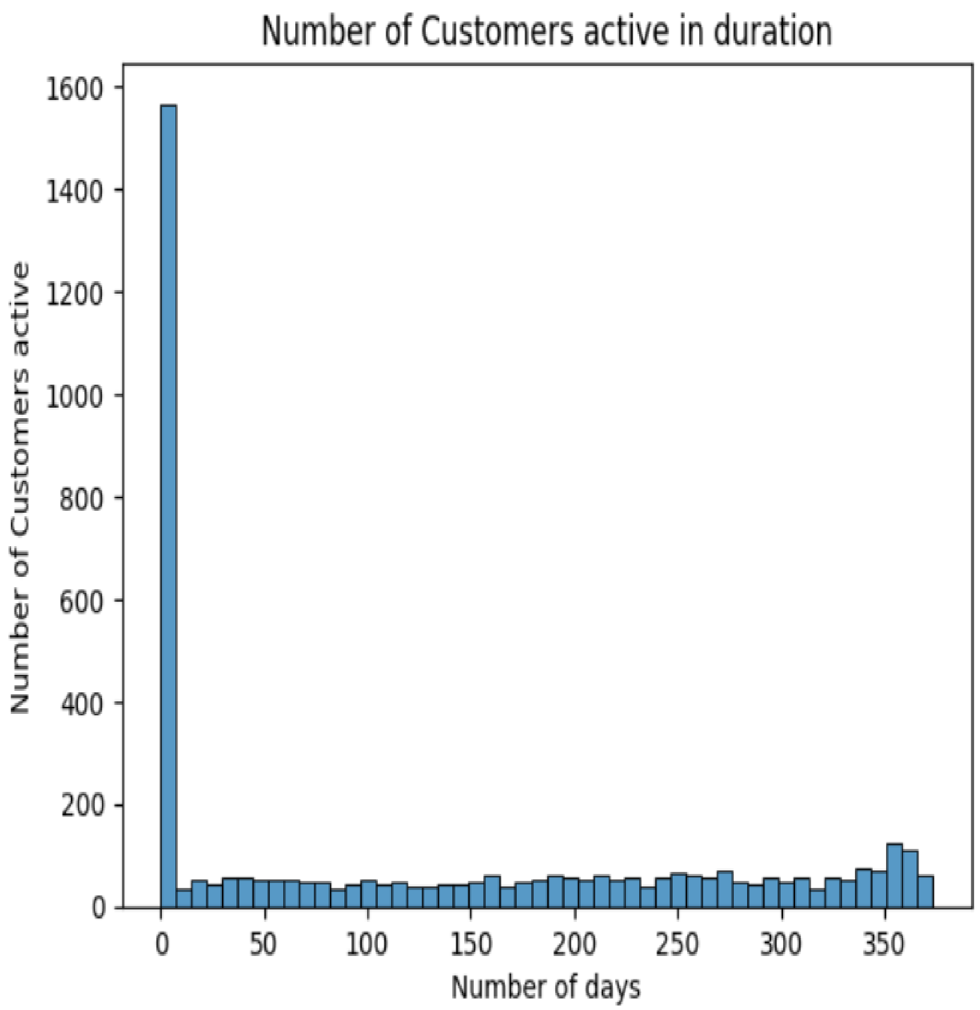
7. Customer Behavior

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

```
# 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')
```

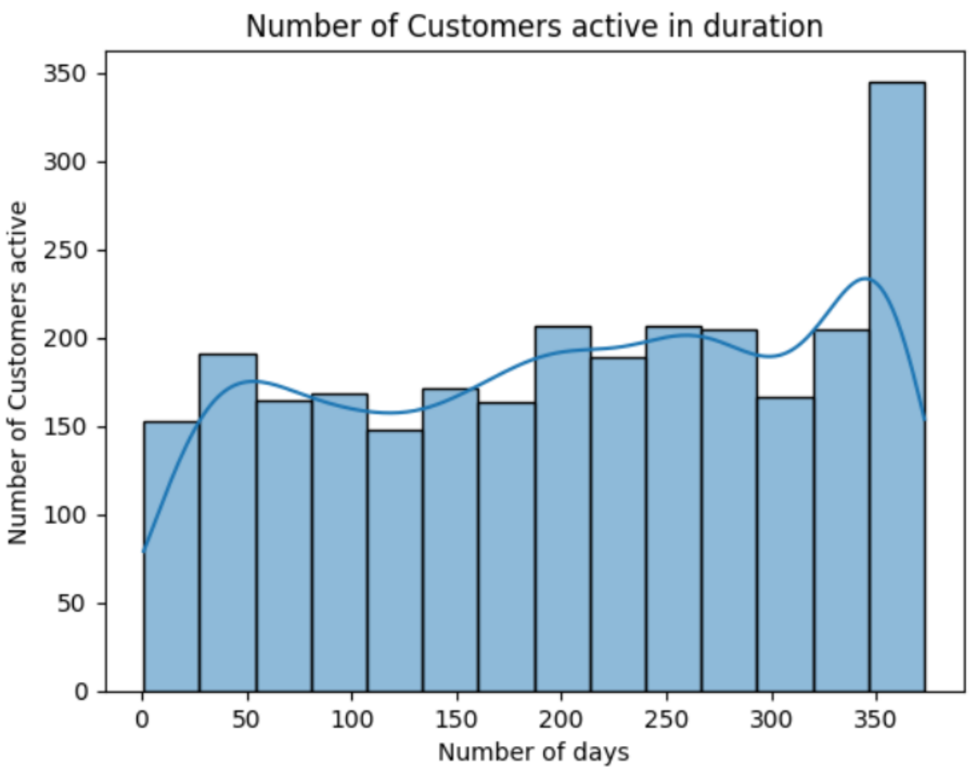
```
The Average active Duration of Customer: 129 days
```

```python
# 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

```python
# 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

The chart shows a slight rise in the number of customers active as the duration increases, with a notable peak around the 350-day mark.

A significant portion of customers remains consistent activity throughout the range.

There seems to be a retention trend where a sizable number of customers maintain activity for longer durations.

8. Returns and Refunds

- The percentage of orders that have experienced returns or refunds is: 14.81%
- High Return Counts:

Home Decor has the highest number of overall purchases (68,553) and returns (1,612). This suggests that while this category is popular, it might have quality or expectation mismatch issues leading to a higher return count.

2. Low Return Counts

Crafts and Hobbies has a low return count (133), even though it has 9,682 purchases. This indicates that customers are generally satisfied with this category, and the nature of the products aligns well with customer expectations.

Toys and Games also has a relatively low return count (54) with 3,865 purchases, showing strong customer satisfaction or low chances of product defects.

3. Mid-Range Categories

Kitchen and Dining and Stationery and Office show moderate purchase and return rates. While the return counts are not alarming, they can indicate minor issues with product quality or customer satisfaction that can be addressed with better descriptions or packaging.

```
df_heatmap = pd.read_csv('data.csv',encoding='unicode_escape')
has_returns = df_heatmap[df_heatmap['InvoiceNo'].str.startswith('C')].shape[0] > 0

if has_returns:
    total_orders = df_heatmap['InvoiceNo'].nunique()
    returned_orders = df_heatmap[df_heatmap['InvoiceNo'].str.startswith('C')]['InvoiceNo'].nunique()

    percentage_returns = (returned_orders / total_orders) * 100

    print(f"The percentage of orders that have experienced returns or refunds is: {percentage_returns:.2f}%")
else:
    print("No returns or refunds found in the dataset.")
```

```
The percentage of orders that have experienced returns or refunds is: 14.81%
```

```
returned_orders1 = df_heatmap[df_heatmap['InvoiceNo'].str.startswith('C')]['InvoiceNo']
returned_orders1
```

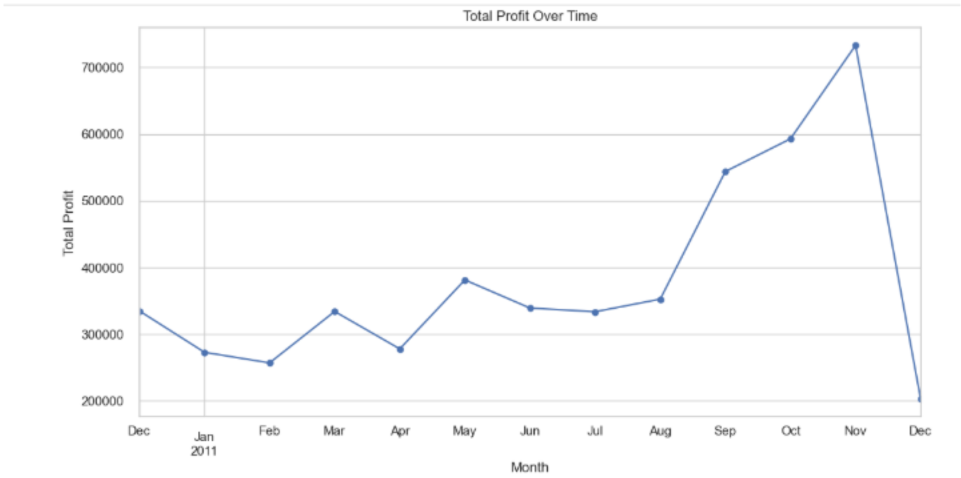
```
141       C536379
154       C536383
235       C536391
236       C536391
237       C536391
           ...
540449    C581490
541541    C581499
541715    C581568
541716    C581569
541717    C581569
Name: InvoiceNo, Length: 9288, dtype: object
```

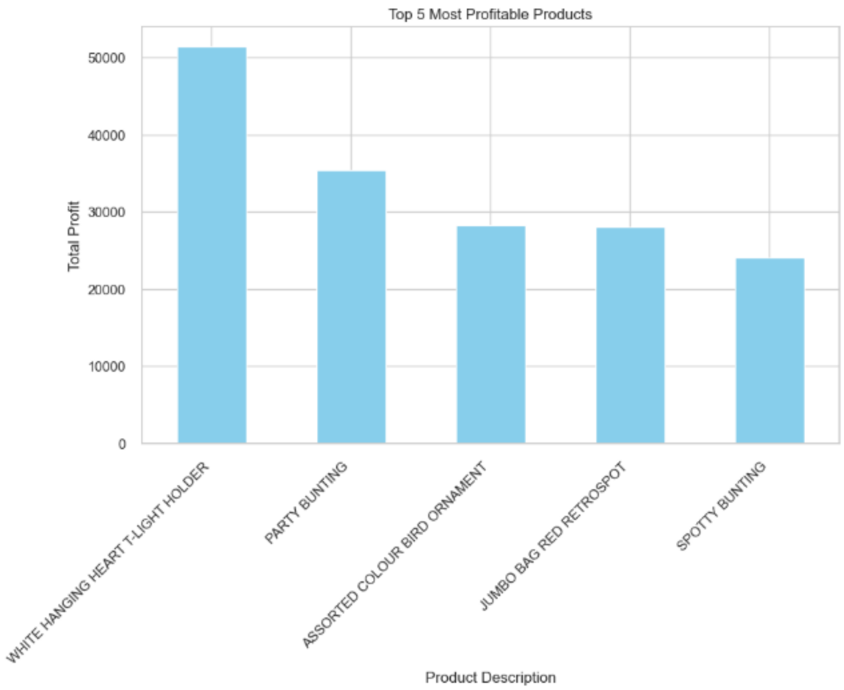
9) Profitability Analysis

since we do not have sufficient data to calculate the total profit earned by the company in dataset's time period, we have calculated total revenue generated in this time period.

```
# Calculate the total profit generated by the company during the dataset's time period
# since
df['Profit'] = df['Quantity'] * df['UnitPrice']
total_profit = df['Profit'].sum()

print(f"The total profit generated by the company is: {total_profit:.2f}")

# top 5 products with the highest profit margins?
top_profitable_products = df.groupby('Description')['Profit'].sum().sort_values(ascending=False).head(5)

print("Top 5 most profitable products:")
print(top_profitable_products)
```

```
The total profit generated by the company is: 4956120.20
Top 5 most profitable products:
Description
WHITE HANGING HEART T-LIGHT HOLDER    51472.01
PARTY BUNTING                         35457.75
ASSORTED COLOUR BIRD ORNAMENT         28255.11
JUMBO BAG RED RETROSPOT               28077.10
SPOTTY BUNTING                        24088.75
Name: Profit, dtype: float64
```





→Overall Trend from the above visualizations are :

The Profit fluctuates from January to August, with relatively stable and moderate growth during this period.
November shows the highest profit for the year.
Profit drops sharply in December.

→Top 5 Most Profitable Products from the bar chart are:
White Hanging Heart T-Light Holder is the most profitable, generating over 50,000 in profit. It's a clear leader and likely a customer favourite.

Party Bunting, Assorted Colour Bird Ornament, Jumbo Bag Red Retrospot, and Spotty Bunting also contribute significantly to profit.

The variety of top products suggests that different categories contribute to profitability, making it important to maintain diversity.

10) Customer Satisfaction:

The dataset lacks a dedicated column for customer feedback, limiting insights into satisfaction and preferences. This absence hinders a comprehensive analysis of customer sentiments and their potential impact on business strategies.

Key Findings:

1. Identification of key customer segments with distinct purchasing behaviors.
2. Insights into the correlation between customer segments and their contribution to revenue.
3. Development of tailored marketing strategies for different segments to enhance customer engagement and profitability.

Limitations and Future Work:

Discusses the limitations encountered, such as the absence of customer feedback data, and outlines future directions for incorporating additional datasets, predictive modeling, and dynamic RFM segmentation.

Conclusion:

1. The project demonstrates the effectiveness of RFM analysis in understanding customer behavior and guiding targeted marketing efforts.
2. The findings provide a basis for enhancing customer satisfaction and driving business growth.