Customer Segmentation: Clustering & RFM Analysis

Importing the necessary Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
```

In [6]:
```
data = pd.read_csv(r'C:\Users\hp\Desktop\RFM data\data.csv', encoding='latin1')
data
```

Out[6]:

	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
...	...	...	...	...	...	...	...	...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/2011 12:50	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	12/9/2011 12:50	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/2011 12:50	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/2011 12:50	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/2011 12:50	4.95	12680.0	France

541909 rows × 8 columns

Exploratory Data Analysis (EDA)

In [7]:
```
data.head()
```

Out[7]:

	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

In [8]:
```
data.tail()
```

Out[8]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/2011 12:50	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	12/9/2011 12:50	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/2011 12:50	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/2011 12:50	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/2011 12:50	4.95	12680.0	France

In [9]:
```
data.isna().sum()
```

Out[9]:
```
InvoiceNo          0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID     135080
Country            0
dtype: int64
```

In [10]:
```
data.duplicated().sum()
```

Out[10]:
```
5268
```

In [11]:
```
data.drop_duplicates(inplace=True)
```

In [12]:
```
data.dropna(inplace=True)
```

In [13]:
```
data.describe()
```

Out[13]:

	Quantity	UnitPrice	CustomerID
count	401604.000000	401604.000000	401604.000000
mean	12.183273	3.474064	15281.160818
std	250.283037	69.764035	1714.006089
min	-80995.000000	0.000000	12346.000000
25%	2.000000	1.250000	13939.000000
50%	5.000000	1.950000	15145.000000
75%	12.000000	3.750000	16784.000000
max	80995.000000	38970.000000	18287.000000

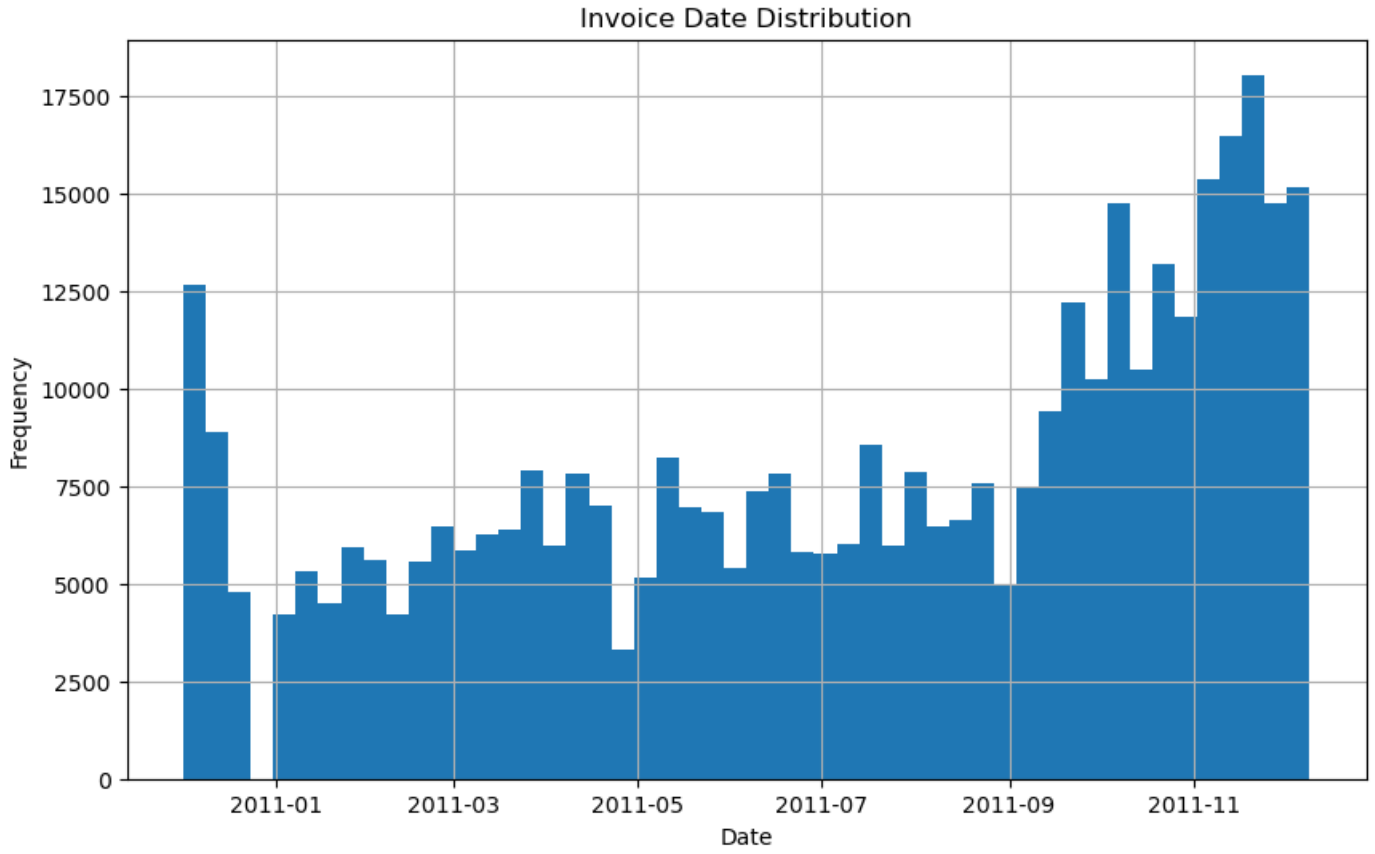
Visualization 1: Invoice Date distribution

Purpose: This visualization displays the distribution of invoice dates, showing the frequency of transactions over time.

Business Insight: It helps identify trends, seasonality, and any irregularities in transaction volume. Businesses can use this information to plan inventory management, marketing campaigns, and staffing levels according to peak sales periods.

In [15]:

```
plt.figure(figsize=(10, 6))
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
data['InvoiceDate'].hist(bins=50)
plt.title('Invoice Date Distribution')
plt.xlabel('Date')
plt.ylabel('Frequency')
plt.show()
```

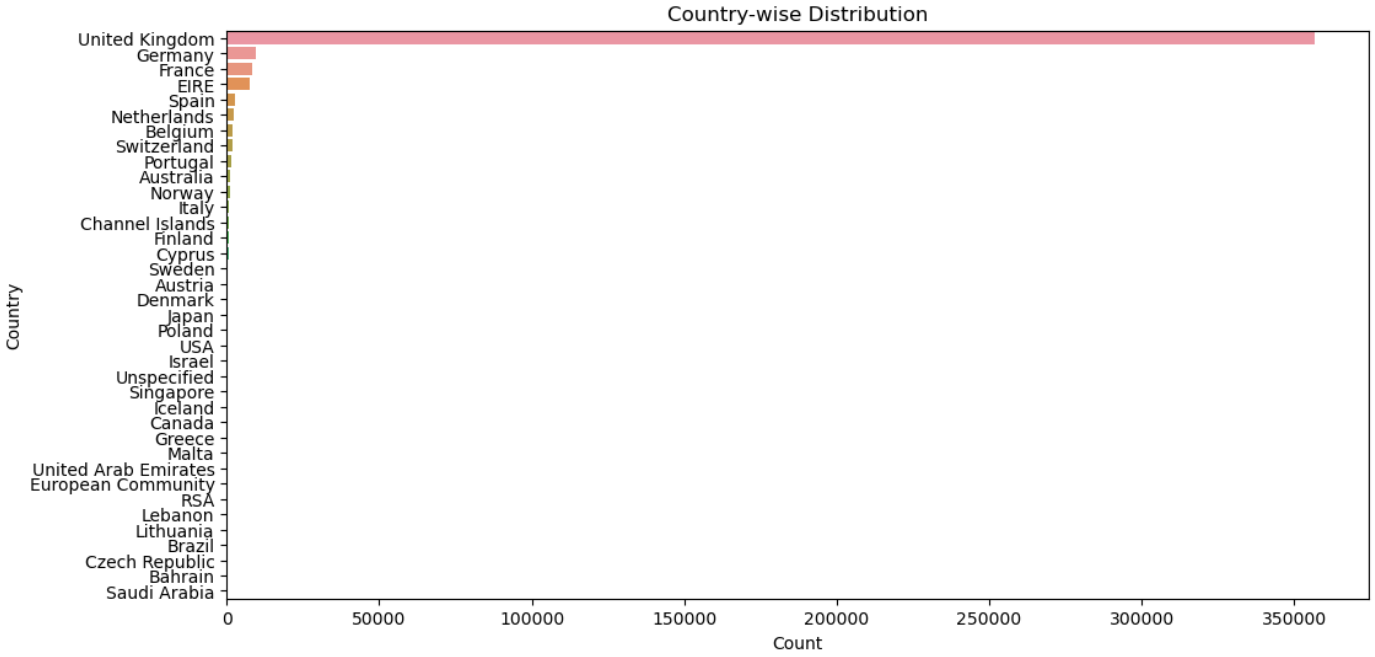


Visualization 2: Country-wise distribution

Purpose: This visualization presents the distribution of transactions across different countries.

Business Insight: It allows businesses to understand their customer base geographically and identify regions with high sales activity. This information can inform international expansion strategies, localization efforts, and targeted marketing campaigns tailored to specific regions.

In [16]:
```
plt.figure(figsize=(12, 6))
sns.countplot(y='Country', data=data, order=data['Country'].value_counts().index)
plt.title('Country-wise Distribution')
plt.xlabel('Count')
plt.ylabel('Country')
plt.show()
```

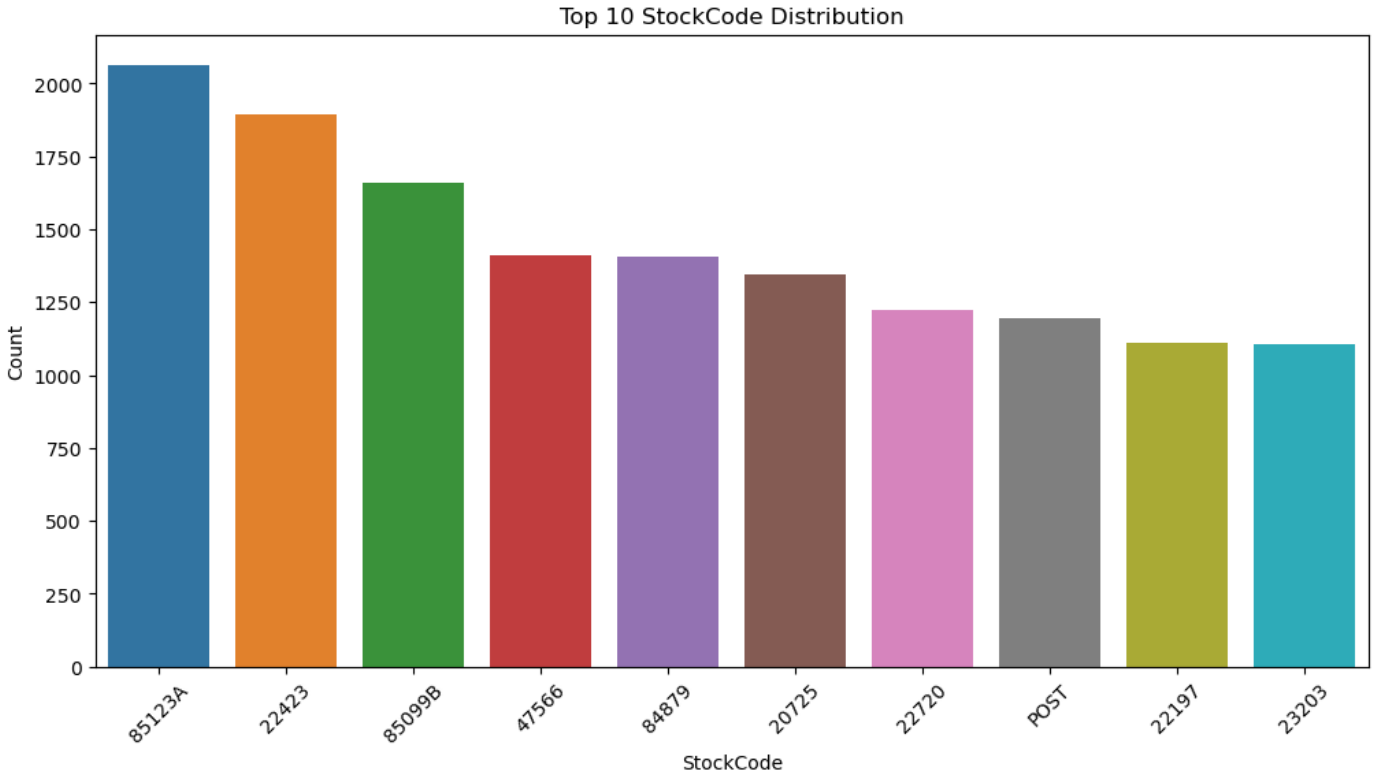


Visualization 3: StockCode distribution

Purpose: This visualization shows the distribution of the top 10 most frequently purchased stock codes.

Business Insight: Identifying the most popular products helps businesses understand customer preferences and demand patterns. It enables inventory management decisions, such as stock replenishment priorities and product bundling strategies, to optimize sales and customer satisfaction.

In [17]:
```
plt.figure(figsize=(12, 6))
sns.countplot(x='StockCode', data=data, order=data['StockCode'].value_counts().index[:10
plt.title('Top 10 StockCode Distribution')
plt.xlabel('StockCode')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```

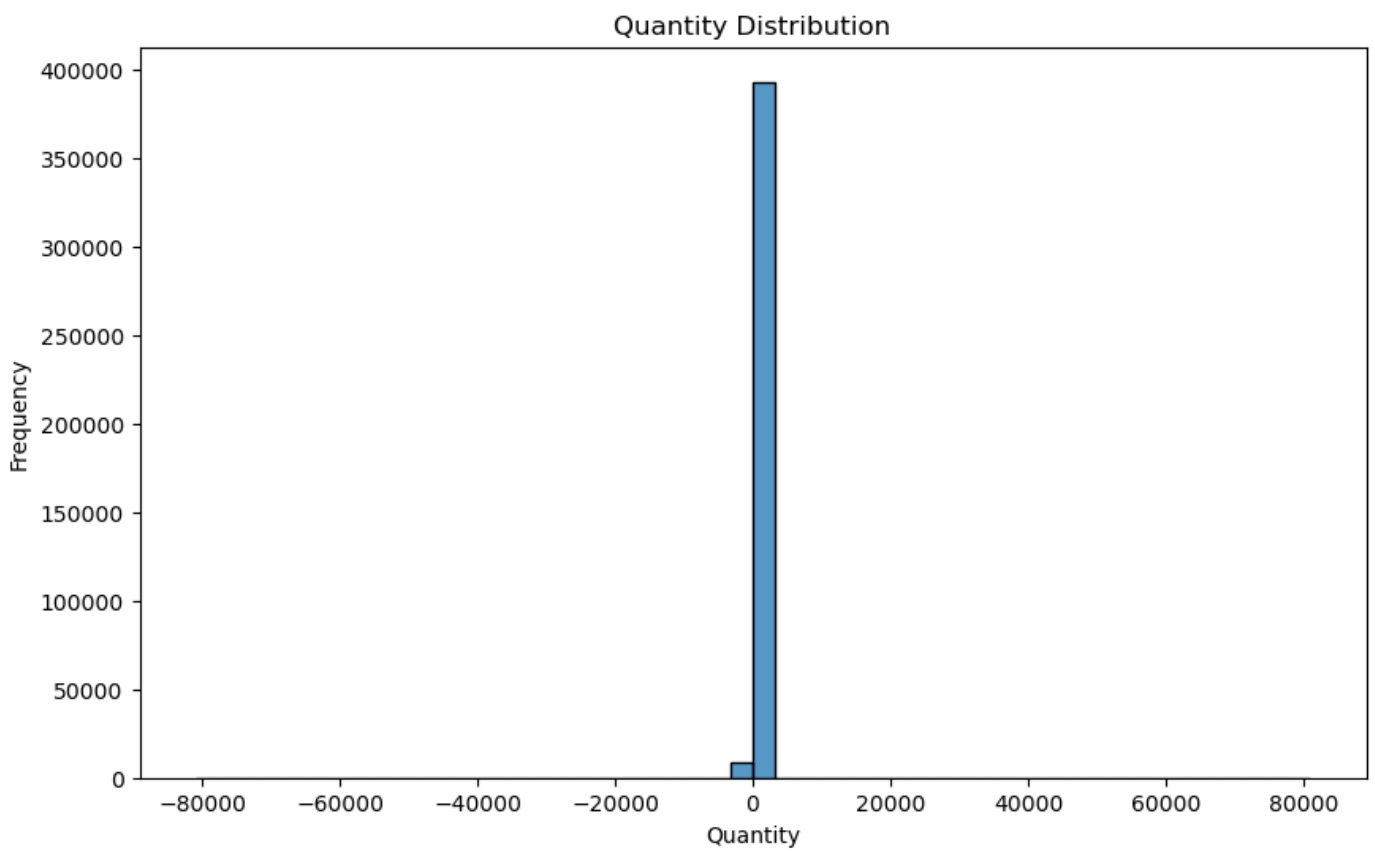


Visualization 4: Quantity distribution

Purpose: This visualization illustrates the distribution of product quantities purchased in each transaction.

Business Insight: Analyzing quantity distribution helps businesses understand typical purchase sizes and customer buying behaviors. It informs decisions related to pricing strategies, product packaging, and promotions aimed at encouraging larger purchases or upselling complementary products.

In [18]:
```
plt.figure(figsize=(10, 6))
sns.histplot(data['Quantity'], bins=50)
plt.title('Quantity Distribution')
plt.xlabel('Quantity')
plt.ylabel('Frequency')
plt.show()
```

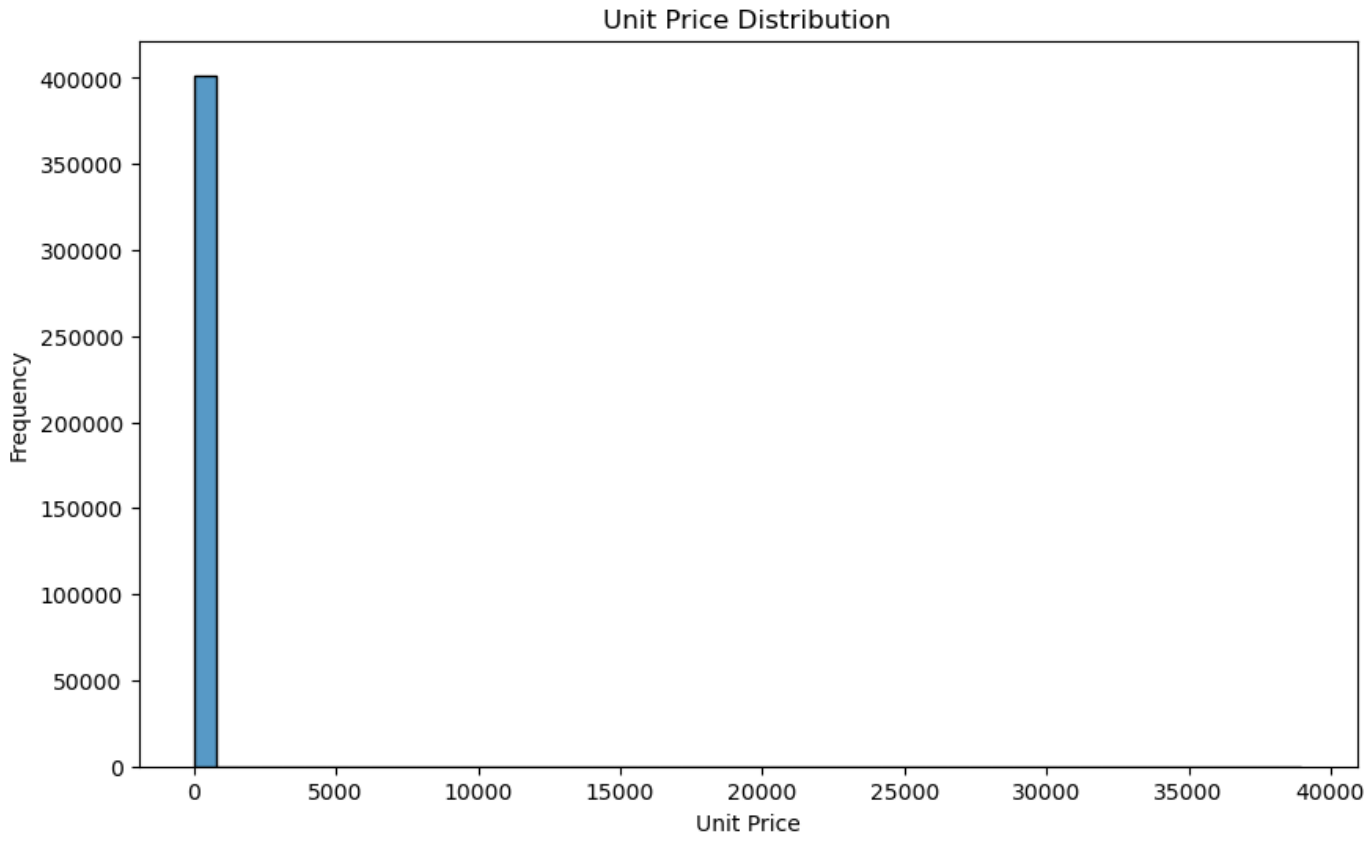


Visualization 5: UnitPrice distribution

Purpose: This visualization displays the distribution of unit prices for products sold.

Business Insight: Examining unit price distribution helps businesses assess pricing strategies and product affordability for customers. It can reveal pricing outliers, price sensitivity levels, and opportunities for adjusting pricing tiers or introducing promotional discounts to maximize sales and profitability.

In [19]:
```
plt.figure(figsize=(10, 6))
sns.histplot(data['UnitPrice'], bins=50)
plt.title('Unit Price Distribution')
plt.xlabel('Unit Price')
plt.ylabel('Frequency')
plt.show()
```

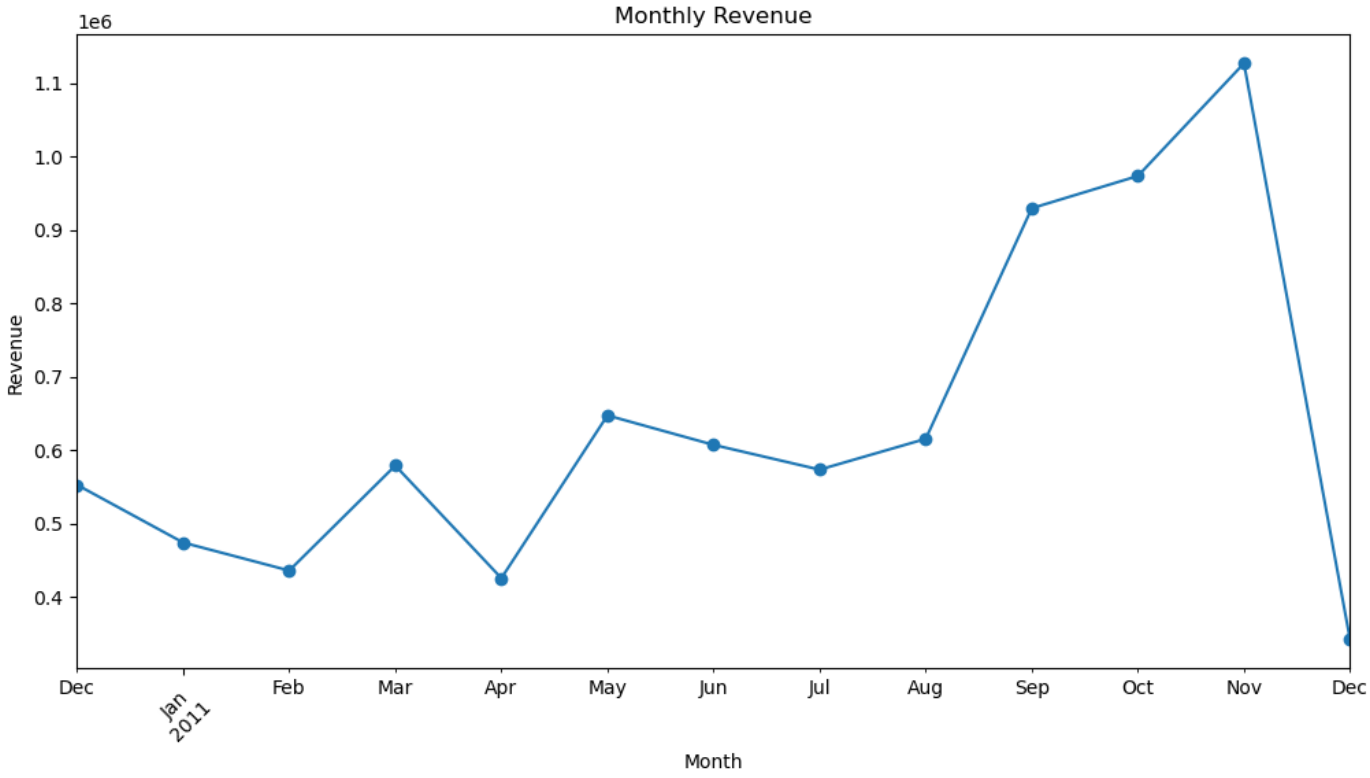


Visualization 6: Revenue Over Time

Purpose: This visualization tracks monthly revenue over time, providing insights into sales trends and identifying seasonal patterns or anomalies.

Business Insight: By observing fluctuations in revenue, businesses can plan marketing campaigns, promotions, or inventory management strategies accordingly to capitalize on peak periods and address any downturns effectively.

In [20]:
```
plt.figure(figsize=(12, 6))
data['Revenue'] = data['Quantity'] * data['UnitPrice']
monthly_revenue = data.groupby(data['InvoiceDate'].dt.to_period('M'))['Revenue'].sum()
monthly_revenue.plot(marker='o')
plt.title('Monthly Revenue')
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.xticks(rotation=45)
plt.show()
```

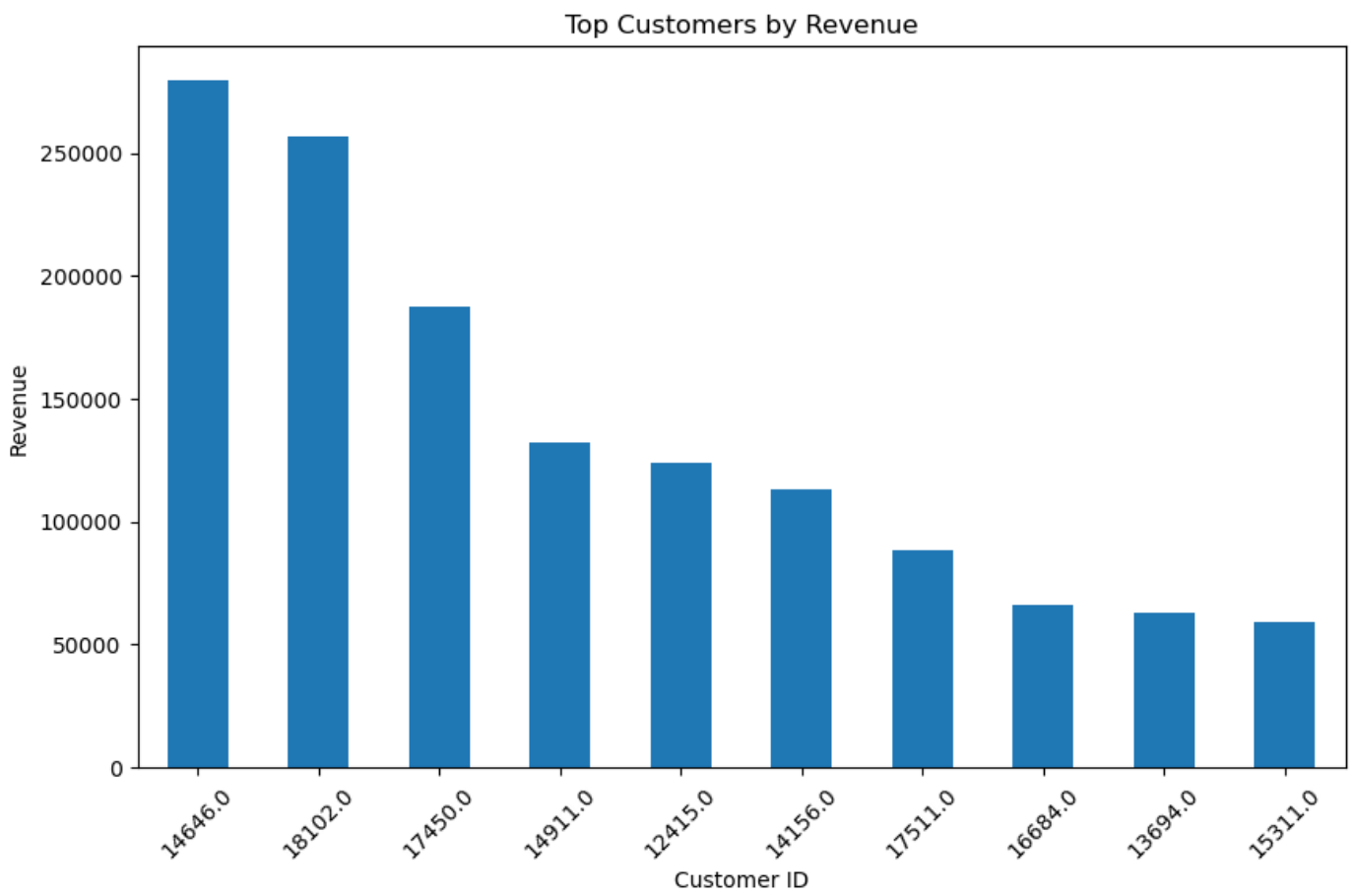


Visualization 7: Top Customers by Revenue

Purpose: This visualization identifies the top revenue-generating customers, helping businesses understand the importance of key clients and their contribution to overall sales.

Business Insight: Recognizing high-value customers allows businesses to tailor personalized marketing strategies, offer loyalty rewards, or provide exceptional customer service to maintain and strengthen these valuable relationships.

In [21]:
```
top_customers = data.groupby('CustomerID')['Revenue'].sum().nlargest(10)
plt.figure(figsize=(10, 6))
top_customers.plot(kind='bar')
plt.title('Top Customers by Revenue')
plt.xlabel('Customer ID')
plt.ylabel('Revenue')
plt.xticks(rotation=45)
plt.show()
```

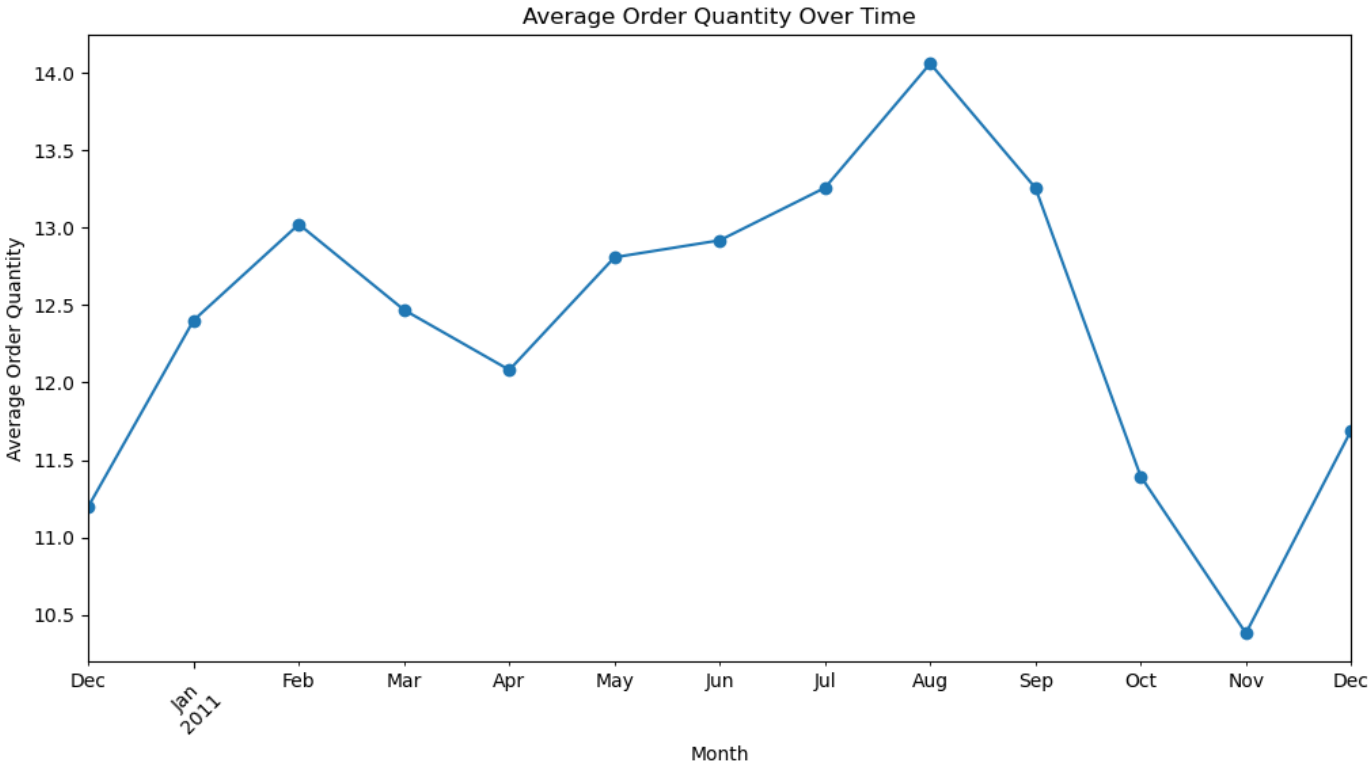


Visualization 8: Average Order Quantity Over Time

Purpose: This visualization tracks the average order quantity per month, revealing purchasing trends and variations in customer buying behavior.

Business Insight: Monitoring changes in average order quantity over time enables businesses to adjust inventory levels, forecast demand more accurately, and optimize pricing strategies to encourage larger purchases during slower periods.

In [22]:
```
plt.figure(figsize=(12, 6))
avg_order_quantity = data.groupby(data['InvoiceDate'].dt.to_period('M'))['Quantity'].mea
avg_order_quantity.plot(marker='o')
plt.title('Average Order Quantity Over Time')
plt.xlabel('Month')
plt.ylabel('Average Order Quantity')
plt.xticks(rotation=45)
plt.show()
```



Calculate RFM metrics

In [23]:
```
# recency
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
current_date = data['InvoiceDate'].max()
recency_data = data.groupby('CustomerID')['InvoiceDate'].max().reset_index()
recency_data['Recency'] = (current_date - recency_data['InvoiceDate']).dt.days
print(recency_data[['CustomerID','Recency']])
```

```
      CustomerID  Recency
0        12346.0      325
1        12347.0        1
2        12348.0       74
3        12349.0       18
4        12350.0      309
...          ...      ...
4367     18280.0      277
4368     18281.0      180
4369     18282.0        7
4370     18283.0        3
4371     18287.0       42

[4372 rows x 2 columns]
```

In [24]:
```
# frequency
frequency_data = data.groupby('CustomerID')['InvoiceNo'].nunique().reset_index()
frequency_data.columns = ['CustomerID','Frequency']
print(frequency_data)
```

```
      CustomerID  Frequency
0        12346.0          2
1        12347.0          7
2        12348.0          4
3        12349.0          1
4        12350.0          1
...          ...        ...
4367     18280.0          1
4368     18281.0          1
4369     18282.0          3
4370     18283.0         16
4371     18287.0          3

[4372 rows x 2 columns]
```

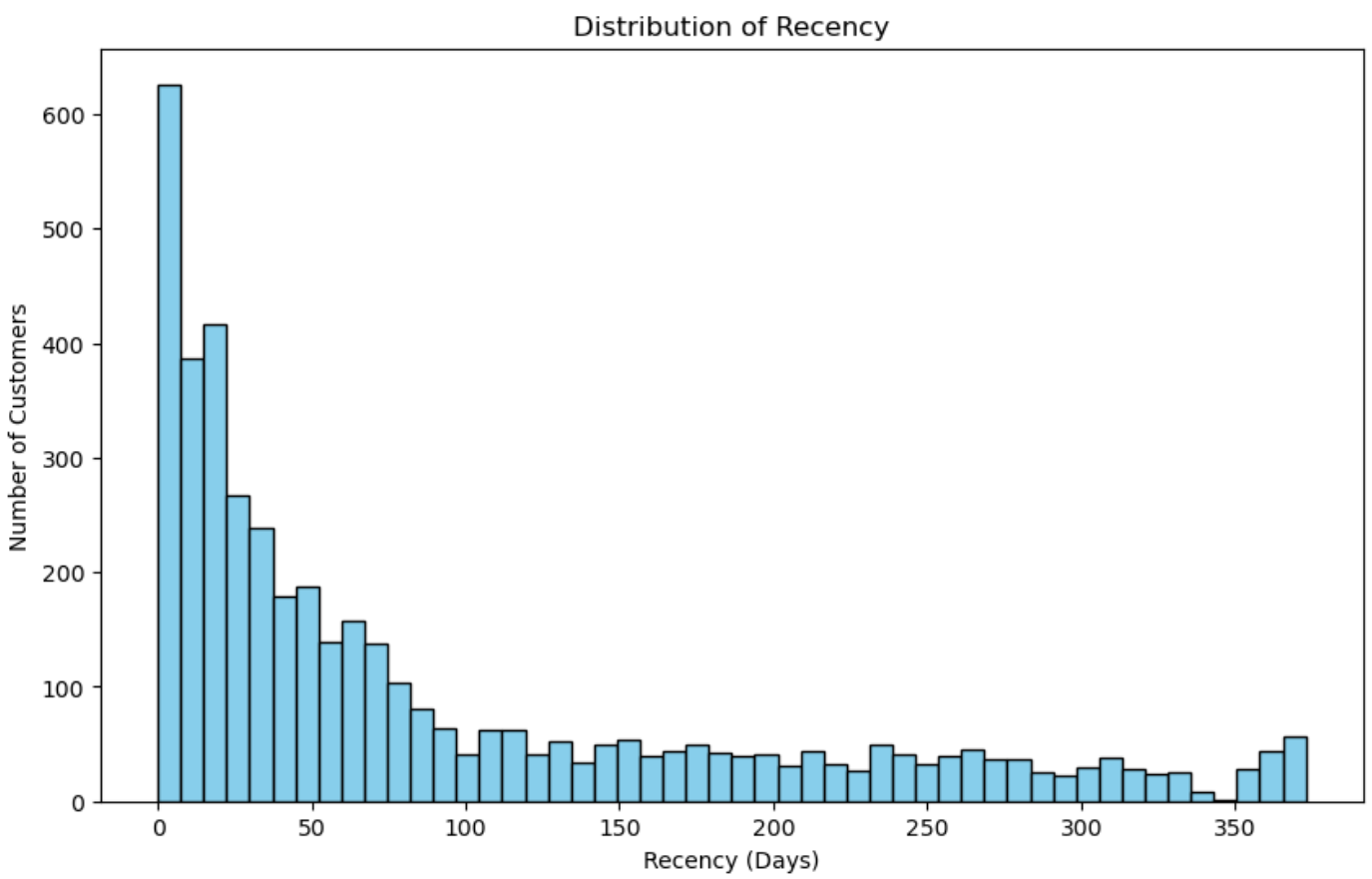
In [25]:
```
# monetary
data['TotalPrice'] = data['Quantity']*data['UnitPrice']
```

In [26]:
```
monetary_data = data.groupby('CustomerID')['TotalPrice'].sum().reset_index()
monetary_data.columns = ['CustomerID','Monetary']
print(monetary_data)
```

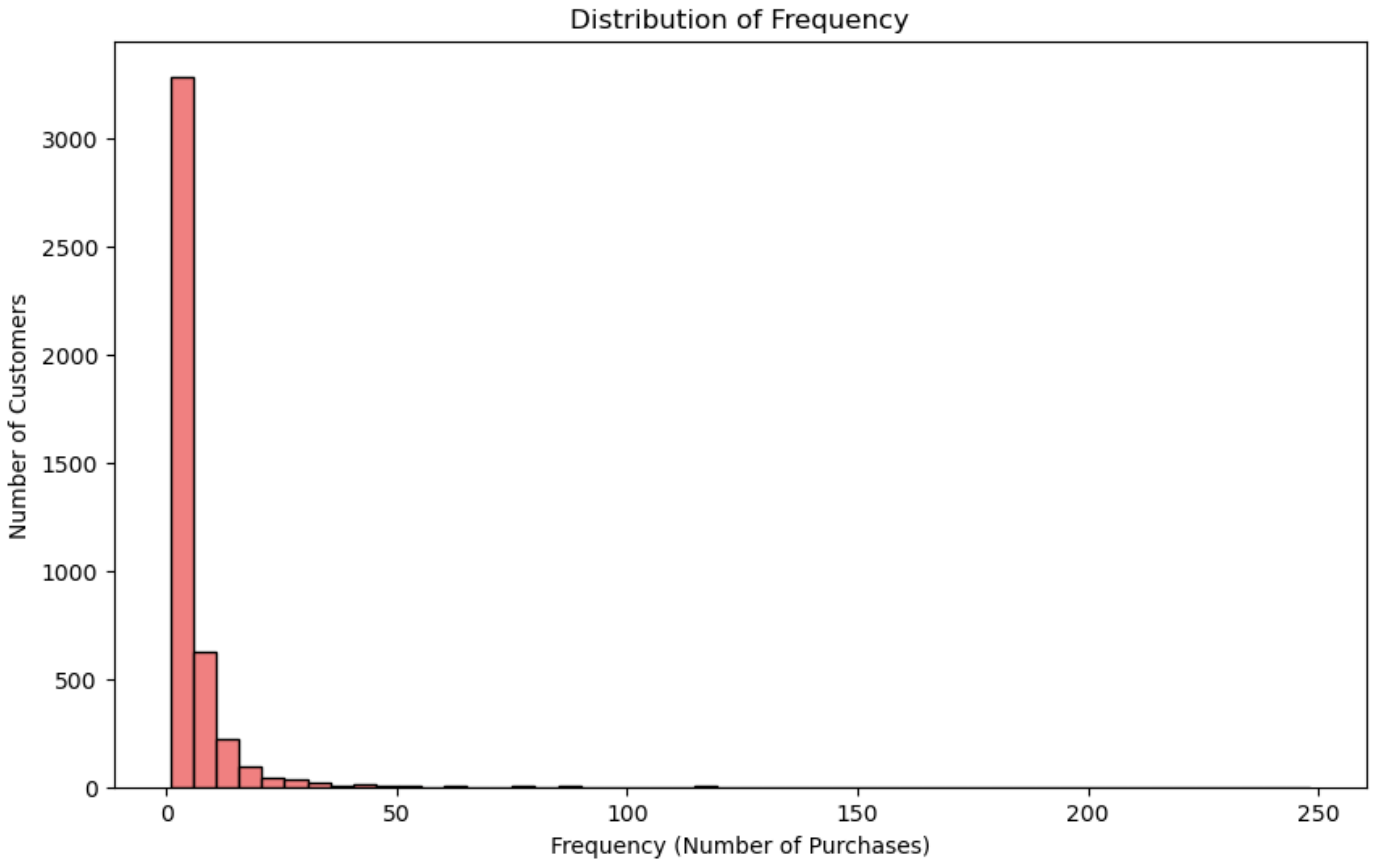
```
      CustomerID  Monetary
0        12346.0      0.00
1        12347.0   4310.00
2        12348.0   1797.24
3        12349.0   1757.55
4        12350.0    334.40
...          ...       ...
4367     18280.0    180.60
4368     18281.0     80.82
4369     18282.0    176.60
4370     18283.0   2045.53
4371     18287.0   1837.28

[4372 rows x 2 columns]
```

Visualization 9: RFM Analysis

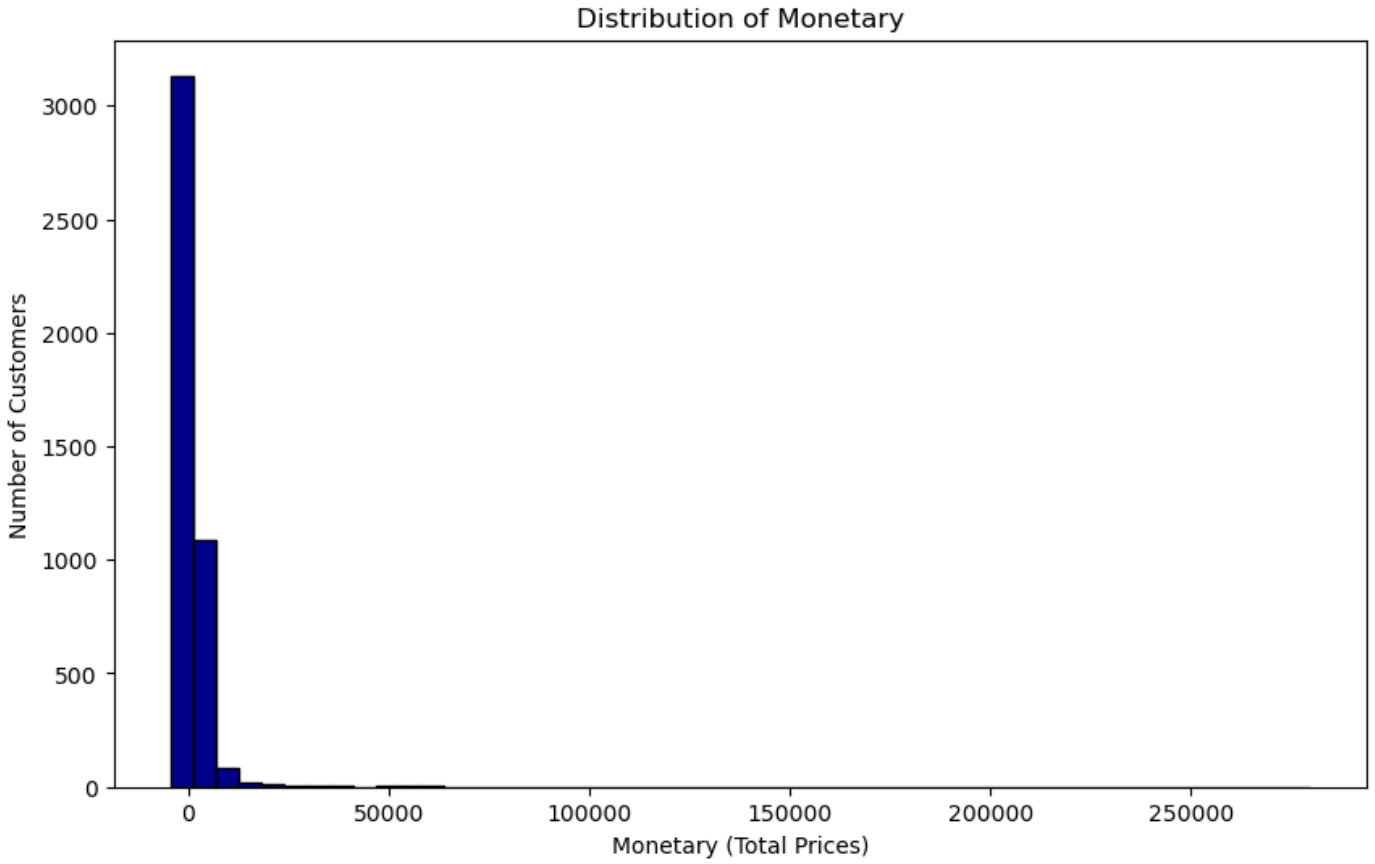
In [27]:
```
plt.figure(figsize=(10,6))
plt.hist(recency_data['Recency'],bins=50, color='skyblue',edgecolor='black')
plt.title('Distribution of Recency')
plt.xlabel('Recency (Days)')
plt.ylabel('Number of Customers')
plt.show()
```



In [28]:
```
plt.figure(figsize=(10,6))
plt.hist(frequency_data['Frequency'],bins=50, color='lightcoral',edgecolor='black')
plt.title('Distribution of Frequency')
plt.xlabel('Frequency (Number of Purchases)')
plt.ylabel('Number of Customers')
plt.show()
```



In [29]:
```
plt.figure(figsize=(10,6))
plt.hist(monetary_data['Monetary'],bins=50, color='darkblue',edgecolor='black')
plt.title('Distribution of Monetary')
plt.xlabel('Monetary (Total Prices)')
plt.ylabel('Number of Customers')
plt.show()
```



Perform K-means clustering with the optimal number of clusters

In [30]:
```
rfm_data = pd.merge(recency_data,frequency_data,on='CustomerID')
rfm_data = pd.merge(rfm_data, monetary_data,on='CustomerID')
X = rfm_data[['Recency','Frequency','Monetary']]
num_clusters = 3
Kmeans = KMeans(n_clusters=num_clusters,random_state=42)
rfm_data['Cluster'] = Kmeans.fit_predict(X)
print(rfm_data[['CustomerID','Recency','Frequency','Monetary','Cluster']])
```

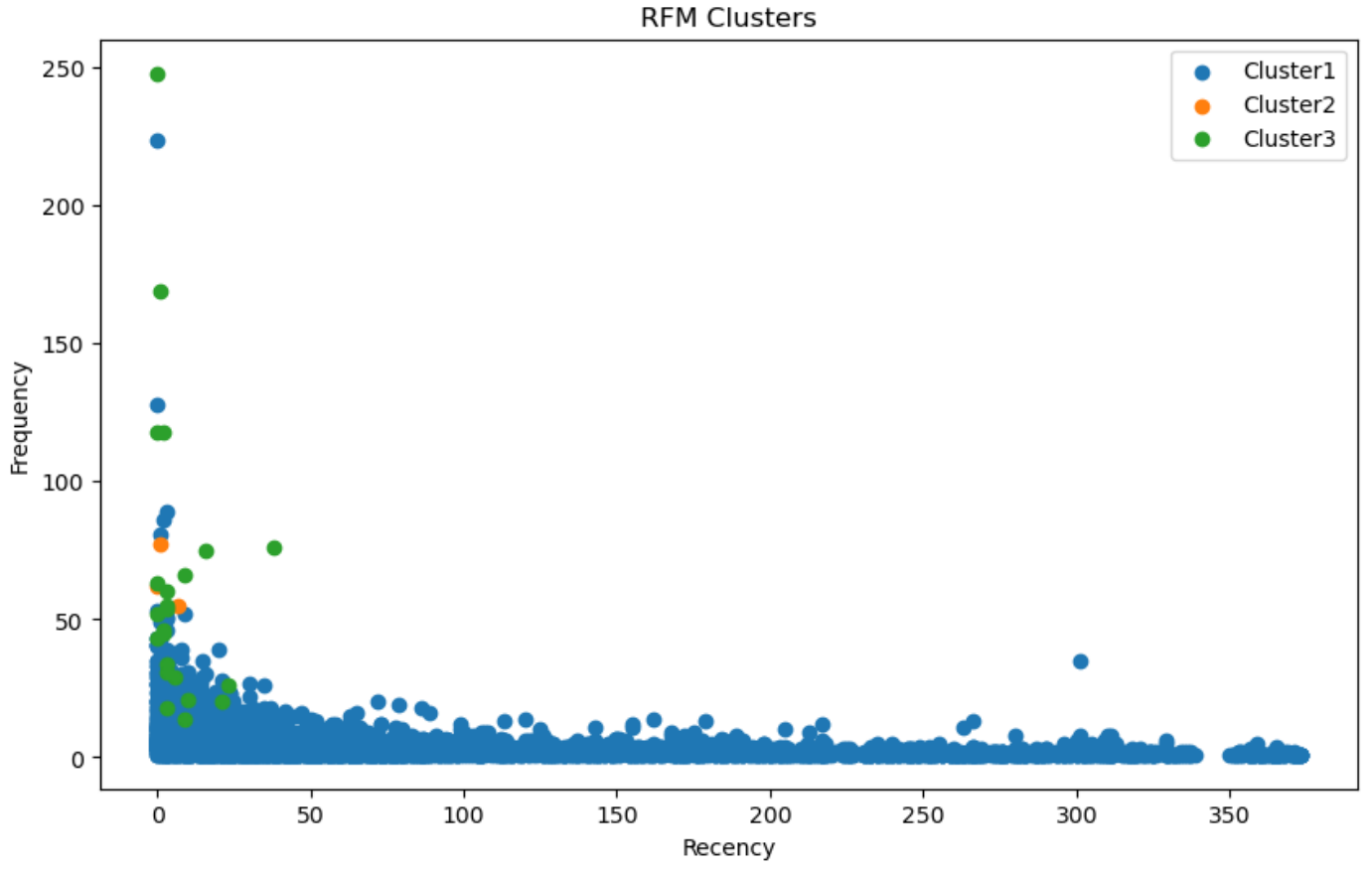
```
C:\Users\hp\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning:
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_
init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

	CustomerID	Recency	Frequency	Monetary	Cluster
0	12346.0	325	2	0.00	0
1	12347.0	1	7	4310.00	0
2	12348.0	74	4	1797.24	0
3	12349.0	18	1	1757.55	0
4	12350.0	309	1	334.40	0
...	...	...	...	...	...
4367	18280.0	277	1	180.60	0
4368	18281.0	180	1	80.82	0
4369	18282.0	7	3	176.60	0
4370	18283.0	3	16	2045.53	0
4371	18287.0	42	3	1837.28	0

[4372 rows x 5 columns]

Visualization 10: RFM Clusters

In [31]:
```
plt.figure(figsize=(10,6))
for cluster in range(num_clusters):
  cluster_data = rfm_data[rfm_data['Cluster']==cluster]
  plt.scatter(cluster_data['Recency'],cluster_data['Frequency'],label=f'Cluster{cluster+
plt.title('RFM Clusters')
plt.xlabel('Recency')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



Calculating R, F, M ranks:

Purpose: This section calculates ranks for the Recency, Frequency, and Monetary values in the RFM dataset.

In [32]:
```
rfm_data['R_rank'] = rfm_data['Recency'].rank(ascending=False)
rfm_data['F_rank'] = rfm_data['Frequency'].rank(ascending=True)
rfm_data['M_rank'] = rfm_data['Monetary'].rank(ascending=True)
rfm_data['R_rank_norm'] = (rfm_data['R_rank']/rfm_data['R_rank'].max())*100
rfm_data['F_rank_norm'] = (rfm_data['F_rank']/rfm_data['F_rank'].max())*100
rfm_data['M_rank_norm'] = (rfm_data['M_rank']/rfm_data['M_rank'].max())*100
rfm_data.drop(columns=['R_rank','F_rank','M_rank'],inplace=True)
rfm_data
```

Out[32]:

	CustomerID	InvoiceDate	Recency	Frequency	Monetary	Cluster	R_rank_norm	F_rank_norm	M_rank_nc
0	12346.0	2011-01-18 10:17:00	325	2	0.00	0	3.865741	39.387008	1.063
1	12347.0	2011-12-07 15:52:00	1	7	4310.00	0	97.719907	81.427264	92.726
2	12348.0	2011-09-25 13:13:00	74	4	1797.24	0	38.182870	64.249771	77.721
3	12349.0	2011-11-21 09:51:00	18	1	1757.55	0	72.974537	15.027447	77.127
4	12350.0	2011-02-02 16:01:00	309	1	334.40	0	5.578704	15.027447	29.940
...	...	...	...	...	...	...	...	...	
4367	18280.0	2011-03-07 09:52:00	277	1	180.60	0	8.379630	15.027447	14.524
4368	18281.0	2011-06-12 10:53:00	180	1	80.82	0	19.988426	15.027447	3.636
4369	18282.0	2011-12-02 11:43:00	7	3	176.60	0	87.627315	54.334401	13.872
4370	18283.0	2011-12-06 12:02:00	3	16	2045.53	0	92.870370	94.842177	80.558
4371	18287.0	2011-10-28 09:29:00	42	3	1837.28	0	54.050926	54.334401	78.179

4372 rows × 9 columns

Calculating RFM Score:

Purpose: The code calculates an RFM score for each customer by combining three key metrics: Recency, Frequency, and Monetary value (RFM). The purpose of this score is to quantify the overall value of each customer based on their purchasing behavior

In [33]:
```
rfm_data['RFM_Score'] = (0.15*rfm_data['R_rank_norm']+0.28*rfm_data['F_rank_norm']+0.57*
rfm_data = rfm_data.round(2)
rfm_data[['CustomerID','RFM_Score']]
```

Out[33]:

	CustomerID	RFM_Score
0	12346.0	0.61
1	12347.0	4.52
2	12348.0	3.40
3	12349.0	2.96
4	12350.0	1.11
...	...	...
4367	18280.0	0.69
4368	18281.0	0.46
4369	18282.0	1.81
4370	18283.0	4.32
4371	18287.0	3.39

4372 rows × 2 columns

Assigning Customer Segments Based on RFM Scores:

Purpose: The code assigns customer segments based on their RFM scores to categorize customers into different groups according to their value and engagement with the business. This segmentation helps tailor marketing strategies and allocate resources effectively.

In [34]:
```
rfm_data['Customer_segment'] = np.where(rfm_data['RFM_Score']>4.5,'Top Customers',
                                (np.where(rfm_data['RFM_Score']>4,'High value Customer',
                                (np.where(rfm_data['RFM_Score']>3,'Medium Value Customer'
                                 (np.where(rfm_data['RFM_Score']>1.5,'Low Value Customers
rfm_data[['CustomerID','RFM_Score','Customer_segment']]
```

Out[34]:

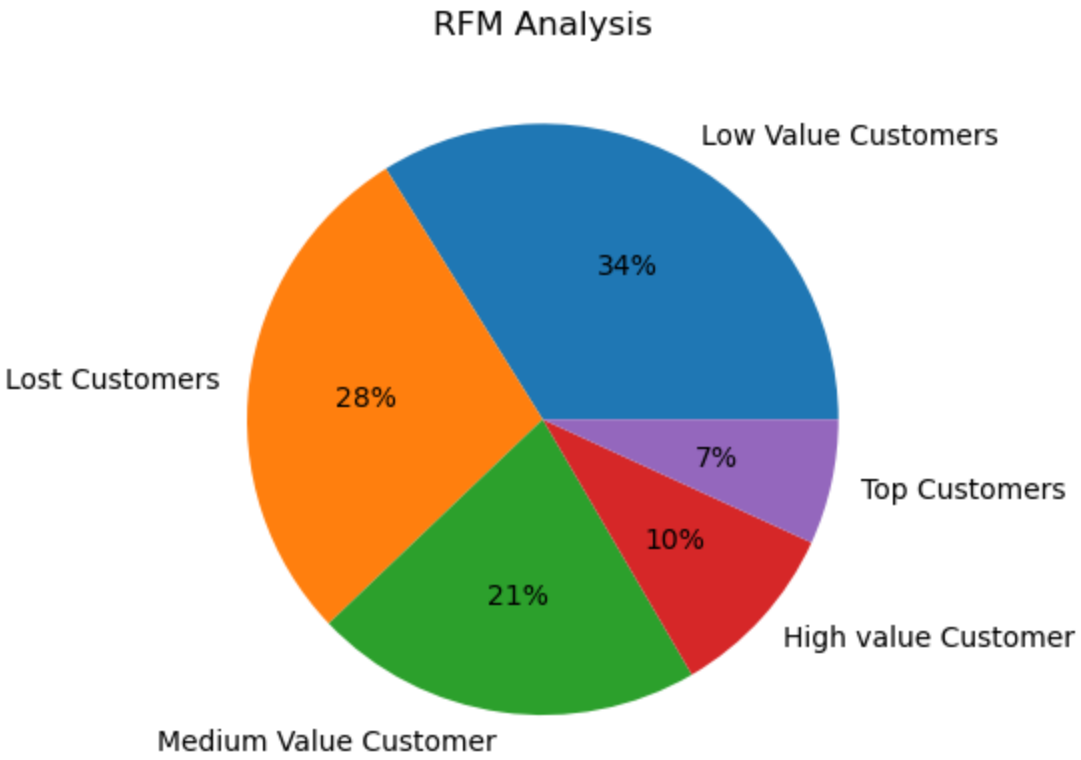
	CustomerID	RFM_Score	Customer_segment
0	12346.0	0.61	Lost Customers
1	12347.0	4.52	Top Customers
2	12348.0	3.40	Medium Value Customer
3	12349.0	2.96	Low Value Customers
4	12350.0	1.11	Lost Customers
...	...	...	...
4367	18280.0	0.69	Lost Customers
4368	18281.0	0.46	Lost Customers
4369	18282.0	1.81	Low Value Customers
4370	18283.0	4.32	High value Customer
4371	18287.0	3.39	Medium Value Customer

4372 rows × 3 columns

Creating a Pie Chart for Customer Segments:

Purpose: The code creates a pie chart to visually represent the distribution of customers across different segments based on RFM analysis. This visualization helps stakeholders quickly grasp the relative proportions of customers in each segment.

In [35]:
```
plt.pie(rfm_data.Customer_segment.value_counts(),labels=rfm_data.Customer_segment.value_
plt.title('RFM Analysis')
plt.show()
```



Conclusion

Our RFM analysis provides valuable insights into our customer base:

Low-Value Customers (34%):

These customers represent a significant portion of our base. We need to focus on engaging them more effectively through targeted marketing campaigns and incentives to increase their spending and loyalty.

High-Value Customers (10%):

Although a smaller group, they contribute a significant portion of our revenue. It's crucial to prioritize retention strategies to ensure we maintain their loyalty and maximize their lifetime value.

Lost Customers (28%):

This group presents an opportunity for re-engagement. Implementing strategies to win back their business, such as personalized communication and special offers, can help reduce churn and recover lost revenue.

Medium-Value Customers (21%):

While not as lucrative as high-value customers, they still make up a considerable portion of our base. We should focus on increasing their transaction frequency and value through incentives and loyalty programs.

Top Customers (7%):

These are our most valuable customers, driving a significant portion of our revenue. It's essential to continue providing them with personalized experiences and exclusive benefits to maintain their loyalty and advocacy.