Data Scientis Interview - Assignment

- Sample Data : E-commerce database of user purchases
- Abdul Kadir Samta

1. Exploratory Data Analysis - An approach to analyzing data sets to summarize their main characteristics.

In [1]:
```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#import dataset
df = pd.read_csv('customer_data.csv',encoding= 'unicode_escape')
df.head()
```

Out[2]:

	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 [3]:
```
df.shape
```

Out[3]: (541909, 8)

In [4]:
```python
#Display the number of attributes available in 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 [5]:
```python
# Validate and convert data types
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
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  datetime64[ns]
 5   UnitPrice    541909 non-null  float64       
 6   CustomerID   406829 non-null  float64       
 7   Country      541909 non-null  object        
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

In [6]:
```
df.describe()
```

Out[6]:

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

In [7]:
```
# Unique values in each column
print(df.nunique())
```

```
InvoiceNo      25900
StockCode       4070
Description     4223
Quantity         722
InvoiceDate    23260
UnitPrice       1630
CustomerID      4372
Country           38
dtype: int64
```

In [8]:
```python
# Check for missing values
print("Missing Values:")
print(df.isnull().sum())
```

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

In [9]:
```python
# Drop rows with missing CustomerID (assuming CustomerID is crucial for customer segmentation)
df = df.dropna(subset=['CustomerID'])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406829 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   InvoiceNo    406829 non-null  object        
 1   StockCode    406829 non-null  object        
 2   Description  406829 non-null  object        
 3   Quantity     406829 non-null  int64         
 4   InvoiceDate  406829 non-null  datetime64[ns]
 5   UnitPrice    406829 non-null  float64       
 6   CustomerID   406829 non-null  float64       
 7   Country      406829 non-null  object        
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 27.9+ MB
```

In [10]:
```python
df.shape
```

Out[10]:
```
(406829, 8)
```

In [11]:
```
#Find the duplicates
df.duplicated().sum()
```

Out[11]:
```
5225
```

In [12]:
```
#Drop duplicate rows, if any
df = df.drop_duplicates()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 401604 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    401604 non-null  object
 1   StockCode    401604 non-null  object
 2   Description  401604 non-null  object
 3   Quantity     401604 non-null  int64
 4   InvoiceDate  401604 non-null  datetime64[ns]
 5   UnitPrice    401604 non-null  float64
 6   CustomerID   401604 non-null  float64
 7   Country      401604 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 27.6+ MB
```

In [13]:
```
df['CustomerID'] = df['CustomerID'].astype(str)
```

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 401604 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   InvoiceNo    401604 non-null  object        
 1   StockCode    401604 non-null  object        
 2   Description  401604 non-null  object        
 3   Quantity     401604 non-null  int64         
 4   InvoiceDate  401604 non-null  datetime64[ns]
 5   UnitPrice    401604 non-null  float64       
 6   CustomerID   401604 non-null  object        
 7   Country      401604 non-null  object        
dtypes: datetime64[ns](1), float64(1), int64(1), object(5)
memory usage: 27.6+ MB
```

In [15]:
```
temp = df[['CustomerID', 'InvoiceNo', 'Country']].groupby(['CustomerID', 'InvoiceNo', 'Country']).count()
temp = temp.reset_index(drop = False)
countries = temp['Country'].value_counts()
```

In [16]:
```python
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

data = dict(type='choropleth',
locations = countries.index,
locationmode = 'country names', z = countries,
text = countries.index, colorbar = {'title':'Order nb.'},
colorscale=[[0, 'rgb(224,255,255)'],
            [0.01, 'rgb(166,206,227)'], [0.02, 'rgb(31,120,180)'],
            [0.03, 'rgb(178,223,138)'], [0.05, 'rgb(51,160,44)'],
            [0.10, 'rgb(251,154,153)'], [0.20, 'rgb(255,255,0)'],
            [1, 'rgb(227,26,28)']],
reversescale = False)
#_______________________
layout = dict(title='Number of orders per country',
geo = dict(showframe = True, projection={'type':'mercator'}))
#______________
choromap = go.Figure(data = [data], layout = layout)
iplot(choromap, validate=False)
```

Number of orders per country

In [17]:
```
pd.DataFrame([{'products': len(df['StockCode'].value_counts()),
               'transactions': len(df['InvoiceNo'].value_counts()),
               'customers': len(df['CustomerID'].value_counts()),
              }], columns = ['products', 'transactions', 'customers'], index = ['quantity'])
```

Out[17]:

	products	transactions	customers
quantity	3684	22190	4372

In [18]:
```
print ("Unique user : 4372")
print ("Unique product : 3684")
print ("Total number of transactions : 22190")
```

```
Unique user : 4372
Unique product : 3684
Total number of transactions : 22190
```

In [19]:
```
temp = df.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)['InvoiceDate'].count()
no_of_products = temp.rename(columns = {'InvoiceDate':'Number of products'})
no_of_products[:10].sort_values('CustomerID')
```

Out[19]:

	CustomerID	InvoiceNo	Number of products
0	12346.0	541431	1
1	12346.0	C541433	1
2	12347.0	537626	31
3	12347.0	542237	29
4	12347.0	549222	24
5	12347.0	556201	18
6	12347.0	562032	22
7	12347.0	573511	47
8	12347.0	581180	11
9	12348.0	539318	17

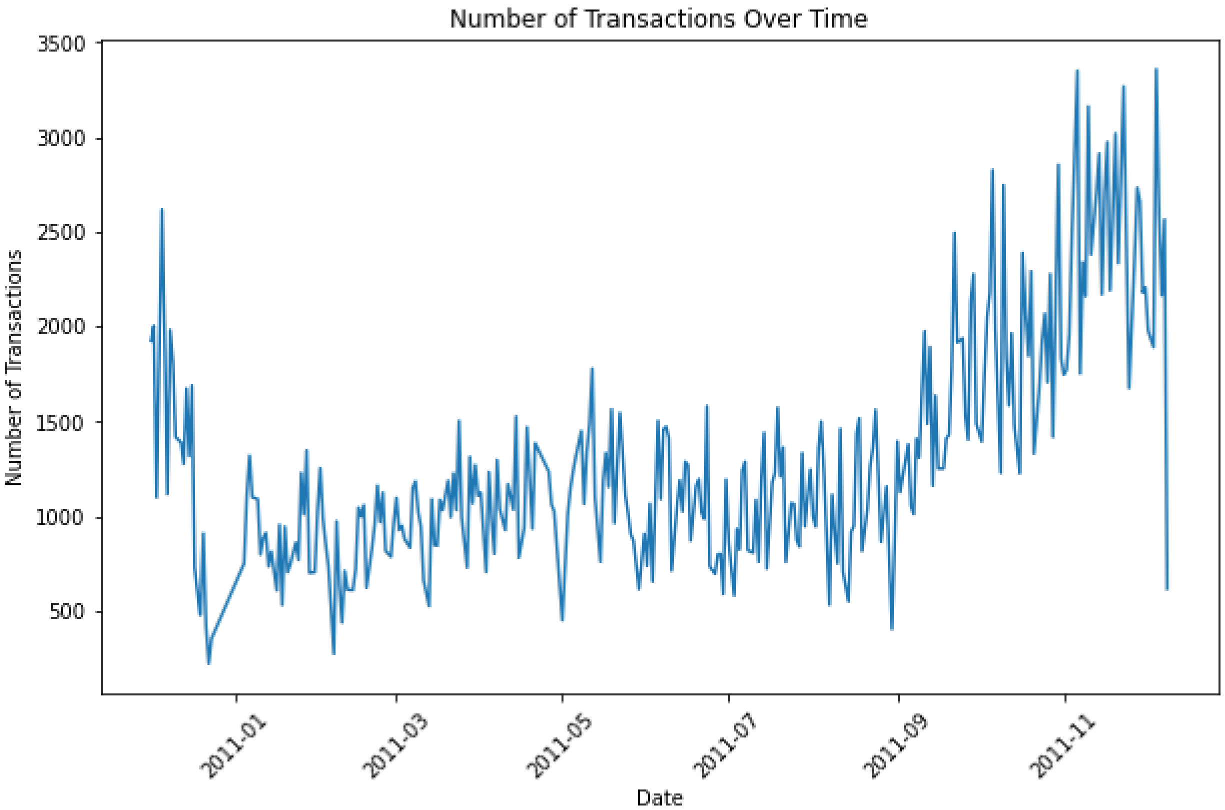
Summary

- Prefix C for the Invoice No that indicates transactions that have been canceled (Customer 12346)
- Customer who only came once and only purchased one product (Customer 12346)
- Frequent users that buy a large number of items at each order (Customer 12347)

In [38]:
```
#To gain more insights on the data
```

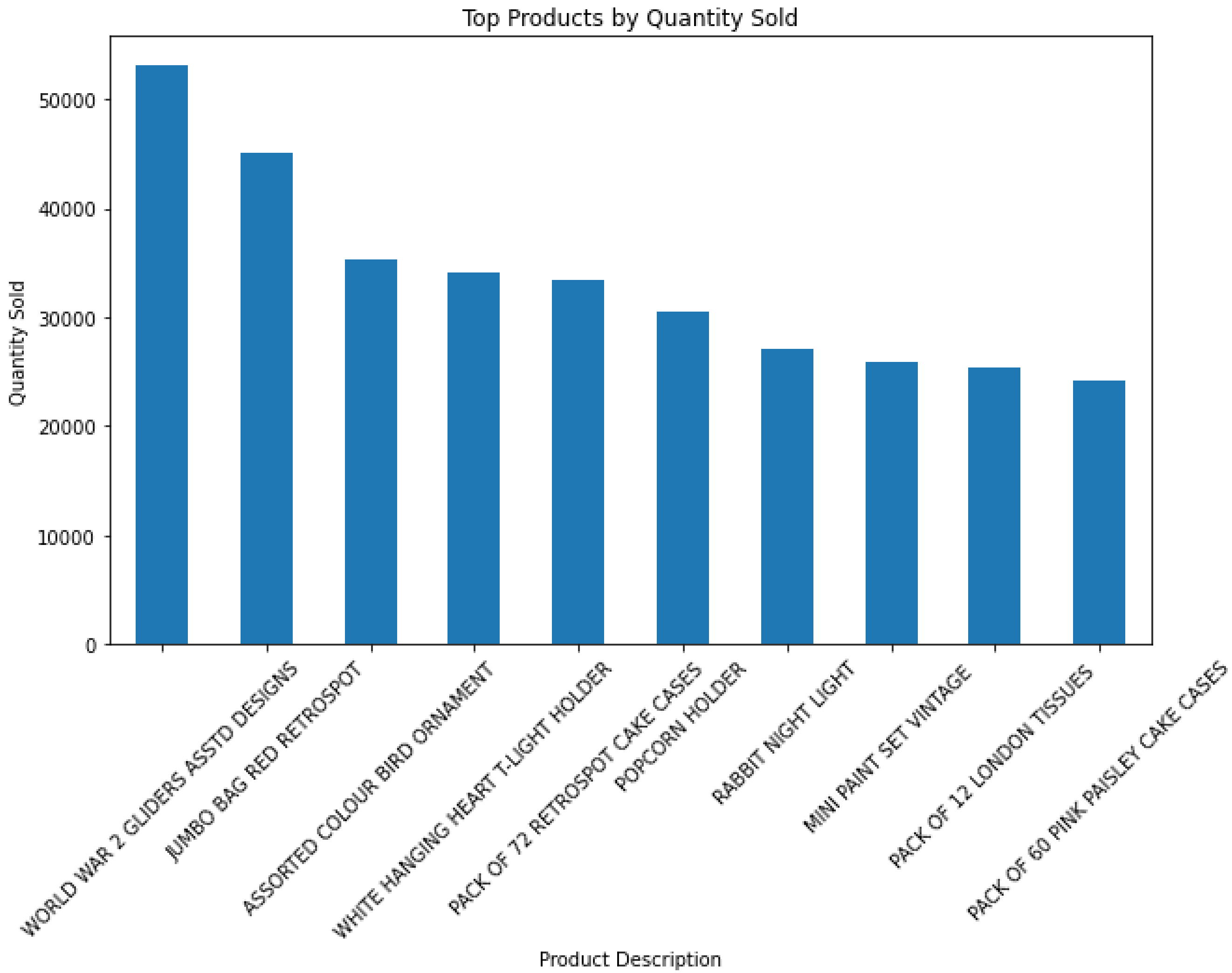
In [40]:

```
#Transaction Distribution Over Time:
# Convert the InvoiceDate column to datetime
df["InvoiceDate"] = pd.to_datetime(df["InvoiceDate"])

# Create a new column for just the date
df["Date"] = df["InvoiceDate"].dt.date

# Plot the number of transactions over time
plt.figure(figsize=(10, 6))
df.groupby("Date").size().plot()
plt.title("Number of Transactions Over Time")
plt.xlabel("Date")
plt.ylabel("Number of Transactions")
plt.xticks(rotation=45)
plt.show()
```



Based on the graph above, we can see that the highest number of transactions occured on November 2011 compare to the lowest number of transactions on March 2011. This data correlates with the total revenue.

In [41]:
```
# Group by product and calculate total quantity sold
product_quantity = df.groupby("Description")["Quantity"].sum().sort_values(ascending=False)[:10]

# Plot top products by quantity sold
plt.figure(figsize=(10, 6))
product_quantity.plot(kind="bar")
plt.title("Top Products by Quantity Sold")
plt.ylabel("Quantity Sold")
plt.xlabel("Product Description")
plt.xticks(rotation=45)
plt.show()
```

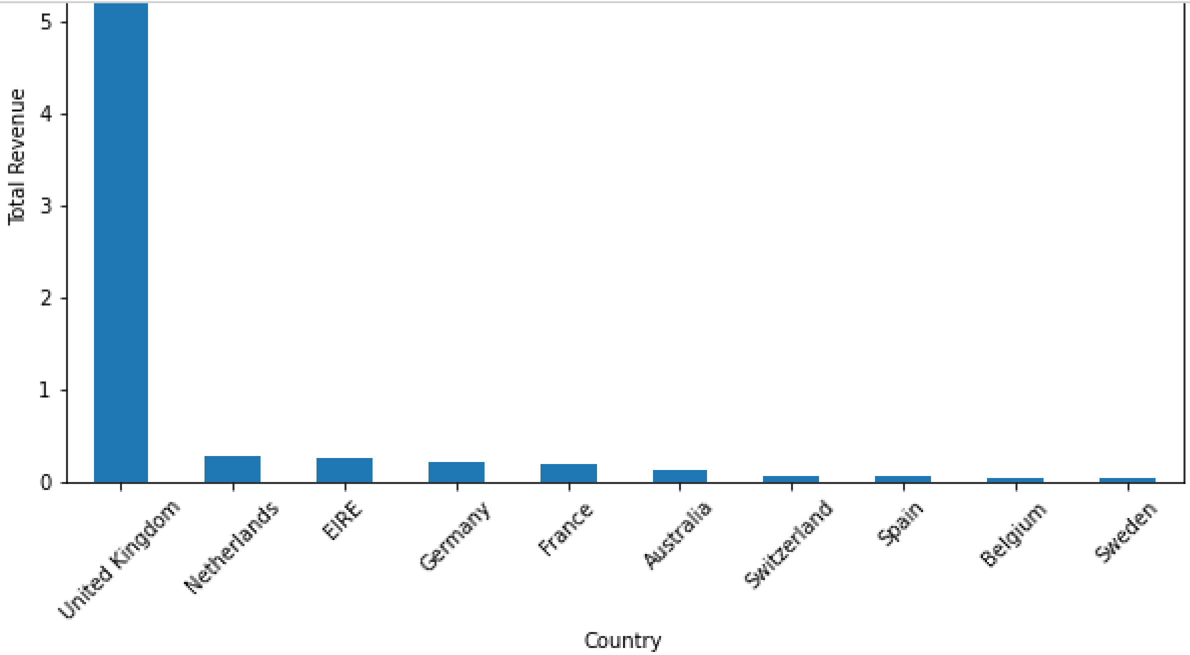


Based on this graph, we can the the popularity of the products. Thus, with data, we can recommend new promotional activities focusing on the product itself. We can reduce the marketing for the highest sales product as the product already popular and have a name on the market

In [20]:
```
#Product Analysis
#Top-selling products based on quantity
top_products_quantity = df.groupby(['StockCode', 'Description'])['Quantity'].sum().nlargest(5)
print("Top-selling products based on quantity:")
print(top_products_quantity)
print('\t')

# Top-selling products based on revenue (Revenue = Quantity * UnitPrice)
df['Revenue'] = df['Quantity'] * df['UnitPrice']
top_products_revenue = df.groupby(['StockCode', 'Description'])['Revenue'].sum().nlargest(5)
print("Top-selling products based on revenue:")
print("Revenue = Quantity * UnitPrice")
print(top_products_revenue)
```

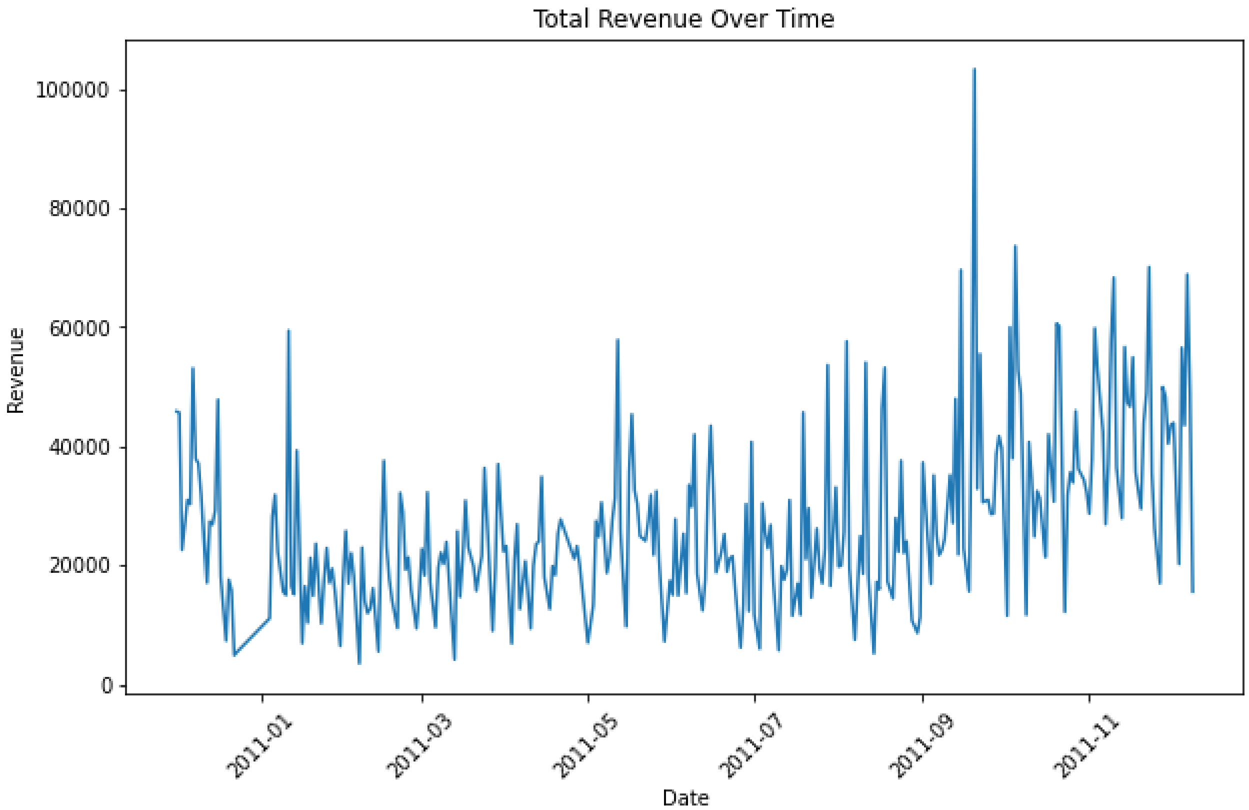
```
Top-selling products based on quantity:
StockCode  Description
84077      WORLD WAR 2 GLIDERS ASSTD DESIGNS     53119
85099B     JUMBO BAG RED RETROSPOT               44963
84879      ASSORTED COLOUR BIRD ORNAMENT         35215
85123A     WHITE HANGING HEART T-LIGHT HOLDER    34128
21212      PACK OF 72 RETROSPOT CAKE CASES       33386
Name: Quantity, dtype: int64

Top-selling products based on revenue:
Revenue = Quantity * UnitPrice
StockCode  Description
22423      REGENCY CAKESTAND 3 TIER              132567.70
85123A     WHITE HANGING HEART T-LIGHT HOLDER     93767.80
85099B     JUMBO BAG RED RETROSPOT                83056.52
47566      PARTY BUNTING                          67628.43
POST       POSTAGE                                66710.24
Name: Revenue, dtype: float64
```

In [43]:
```python
# Group by country and calculate total purchase amount
country_purchase = df.groupby("Country")["Revenue"].sum().sort_values(ascending=False)[:10]

# Plot purchase amount by country
plt.figure(figsize=(10, 6))
country_purchase.plot(kind="bar")
plt.title("Revenue by Country")
plt.ylabel("Total Revenue")
plt.xlabel("Country")
plt.xticks(rotation=45)
plt.show()
```



Based on the graph, we can know the distribution of revenue based on the country. Thus, we can recommend on special activities on certain country in order to boost up the sales/revenue.

In [47]:
```python
# Create a new column for just the date
df["Date"] = df["InvoiceDate"].dt.date

# Group by date and calculate total purchase amount
daily_revenue = df.groupby("Date")["Revenue"].sum()

# Plot total purchase amount over time
plt.figure(figsize=(10, 6))
daily_revenue.plot()
plt.title("Total Revenue Over Time")
plt.xlabel("Date")
plt.ylabel("Revenue")
plt.xticks(rotation=45)
plt.show()
```



Based on the graph, we can know that the highest revenue are coming from September 2011 and lowest are during Mar'2011. Thus, we can recommend on promotional activities according to the month.

In [21]:
```
df.corr()
```
Out[21]:

	Quantity	UnitPrice	Revenue
Quantity	1.000000	-0.001243	0.916130
UnitPrice	-0.001243	1.000000	-0.129311
Revenue	0.916130	-0.129311	1.000000

In [22]:
```
sns.heatmap(df.corr(), annot=df.corr())
plt.show()
```


2. Be able to classify customers into segments and anticipates the purchases that will be made by a new customer, during the following year and this, from its first purchase by assigning them appropriate cluster/segment

In [23]:
```
# Customer Segmentation and Purchase Prediction:
# To classify customers into segments, use KMeans clustering method.
# Recency as the time since the last purchase
# Frequency as the number of purchases
# Monetary as the total spent.
```

In [24]:
```python
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

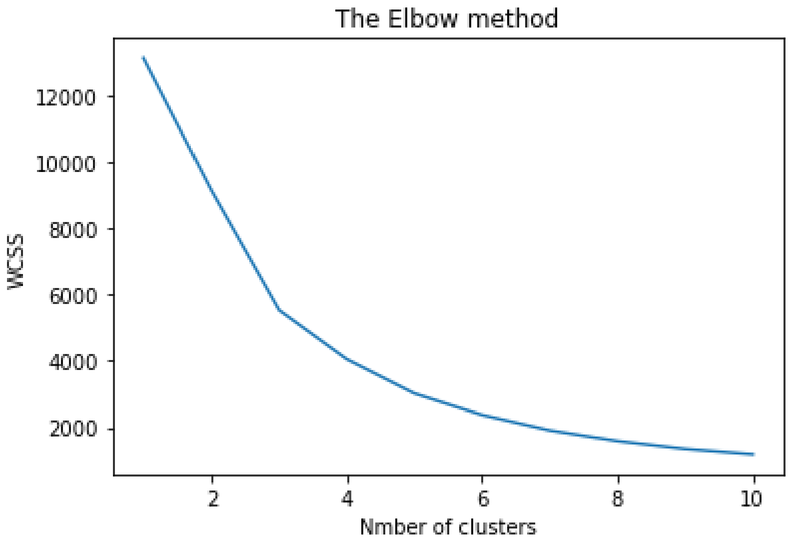
In [25]:
```python
# Calculate RFM values
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
df['Revenue'] = df['Quantity'] * df['UnitPrice']
snapshot_date = df['InvoiceDate'].max() + pd.Timedelta(days=1)
rfm_df = df.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (snapshot_date - x.max()).days,
    'InvoiceNo': 'nunique',
    'Revenue': 'sum'
})
rfm_df.rename(columns={
    'InvoiceDate': 'Recency',
    'InvoiceNo': 'Frequency',
    'Revenue': 'Monetary'
}, inplace=True)
```

In [26]:
```python
# Standardize the data
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm_df)
rfm_scaled
```

Out[26]:
```
array([[ 2.32202285, -0.32936215, -0.23041952],
       [-0.89373323,  0.20610242,  0.29405454],
       [-0.1691956 , -0.11517632, -0.01171748],
       ...,
       [-0.83418219, -0.22226923, -0.20892947],
       [-0.87388289,  1.16993863,  0.01849636],
       [-0.48680114, -0.22226923, -0.00684511]])
```

In [27]:

```
wcss = []
for i in range (1,11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(rfm_scaled)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.title('The Elbow method')
plt.xlabel('Nmber of clusters')
plt.ylabel('WCSS')
plt.show()

print('The value of k, cluster = 4')
```



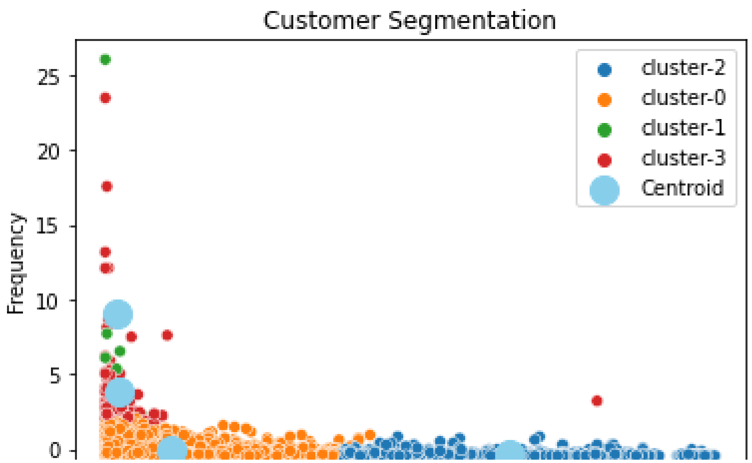

```
The value of k, cluster = 4
```

In [28]:

```
# Apply KMeans clustering to identify segments
kmeans = KMeans(n_clusters=4, random_state=42)
rfm_df['Cluster'] = kmeans.fit_predict(rfm_scaled)
```

In [29]:
```python
#visualize
sns.scatterplot(rfm_scaled[:,0], rfm_scaled[:,1], hue = ["cluster-{}".format(x) for x in rfm_df['Cluster']])
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1], s= 200, c= 'skyblue', label = 'Centroid')
plt.xlabel('Recency')
plt.ylabel('Frequency')
plt.title('Customer Segmentation')
plt.legend()
plt.show()
```

```
/Users/abdulkadir/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `
data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
```



In [30]:
```python
# Check the cluster centers
print(rfm_df.groupby('Cluster').mean())
```

```
            Recency  Frequency      Monetary
Cluster
0         41.606500   4.802461    1472.653251
1          7.666667  89.000000  182108.075000
2        247.951242   1.805888     451.802991
3          9.181818  40.672727   18435.663364
```

In [31]:
```
score = silhouette_score(rfm_scaled, kmeans.labels_,metric='euclidean')
print('Silhouette score :',score)
print('Silhouette score :',round((score*100),2),'%')
```

```
Silhouette score : 0.6114526770024191
Silhouette score : 61.15 %
```

In [32]:
```
pd.Series(rfm_df['Cluster']).value_counts()
```

Out[32]:
```
0    3169
2    1087
3     110
1       6
Name: Cluster, dtype: int64
```

To anticipates the purchases that will be made by a new customer, during the following year and this, from its first purchase by

1. Process the new customer data and calculate the RFM values to obtain the cluster centers and assignments.
2. Use the model and new customer RFM to anticipate whether the new customer is likely to make purchase in the following year.

In [33]:
```
# Assume we have RFM values for a new customer
# Example new customer RFM - R = 60, F = 2, M = 150

new_customer_rfm = np.array([[60, 2, 150]])
new_customer_rfm_scaled = scaler.transform(new_customer_rfm)
predicted_segment = kmeans.predict(new_customer_rfm_scaled)
print(f"Predicted segment for new customer: Cluster {predicted_segment[0]}")
```

```
Predicted segment for new customer: Cluster 0

/Users/abdulkadir/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py:450: UserWarning:

X does not have valid feature names, but StandardScaler was fitted with feature names
```

Based on prediction and assumption above, we can predict that the new customer will purchases total amount of 1472 (referring to monetary value for cluster 0). We can replace the RFM value depending on new customer.

3.Prefix "C" denotes a cancelled transaction,

- Based on the indicator, we observe that when an order is cancelled, there will be another transaction in the dataframe with the same attributes but with negative quantity. Locate these entries and check if there are correlation between orders indicating the same quantity (but positive), with the same description (**CustomerID**, **Description** and **UnitPrice**).
- State whether the hypothesis of "when an order has a negative quantity value, this would always means that the order has been canceled" is true or not.

In [34]:
```
#count no of transactions with prefix "C"
no_of_products['order_canceled'] = no_of_products['InvoiceNo'].apply(lambda x:int('C' in x))
display(no_of_products[:10])
#________________________________________________________________________________
n1 = no_of_products['order_canceled'].sum()
n2 = no_of_products.shape[0]
print('Number of orders canceled: {}/{} ({:.1f}%) '.format(n1, n2, n1/n2*100))
```

	CustomerID	InvoiceNo	Number of products	order_canceled
0	12346.0	541431	1	0
1	12346.0	C541433	1	1
2	12347.0	537626	31	0
3	12347.0	542237	29	0
4	12347.0	549222	24	0
5	12347.0	556201	18	0
6	12347.0	562032	22	0
7	12347.0	573511	47	0
8	12347.0	581180	11	0
9	12348.0	539318	17	0

```
Number of orders canceled: 3654/22190 (16.5%)
```

```
In [35]: display(df.sort_values('CustomerID')[:5])
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Revenue
61619	541431	23166	MEDIUM CERAMIC TOP STORAGE JAR	74215	2011-01-18 10:01:00	1.04	12346.0	United Kingdom	77183.6
61624	C541433	23166	MEDIUM CERAMIC TOP STORAGE JAR	-74215	2011-01-18 10:17:00	1.04	12346.0	United Kingdom	-77183.6
286623	562032	22375	AIRLINE BAG VINTAGE JET SET BROWN	4	2011-08-02 08:48:00	4.25	12347.0	Iceland	17.0
72260	542237	84991	60 TEATIME FAIRY CAKE CASES	24	2011-01-26 14:30:00	0.55	12347.0	Iceland	13.2
14943	537626	22772	PINK DRAWER KNOB ACRYLIC EDWARDIAN	12	2010-12-07 14:57:00	1.25	12347.0	Iceland	15.0

Based on data above, we can see that there are other transactions with mostly identical attributes (CustomerID, Description, Stock Code and UnitPrice) except for quantity and invoice date. However, need to confirm other data for similarities in pattern, thus to locate negative quantity and check whether there are positive quantity on another transactions.

In [36]:
```
#Exclude discount from the description
df_check = df[(df['Quantity'] < 0) & (df['Description'] != 'Discount')][['CustomerID','Quantity','StockCode','Descript

for index, col in  df_check.iterrows():
    if df[(df['CustomerID'] == col[0]) & (df['Quantity'] == -col[1]) & (df['Description'] == col[2])].shape[0] == 0:
        print(index, df_check.loc[index])
        print(15*'-'+'>'+' HYPOTHESIS NOT FULFILLED')
        break
```

```
154 CustomerID                              15311.0
Quantity                                         -1
StockCode                                    35004C
Description     SET OF 3 COLOURED  FLYING DUCKS
UnitPrice                                      4.65
Name: 154, dtype: object
---------------> HYPOTHESIS NOT FULFILLED
```

Based on test above, we can see that the hypothesis are not fulfilled which means cancellations do not necessarily corresponds to orders that have been made before.

In [37]:
```
df_check = df[df['Quantity'] < 0][['CustomerID','Quantity','StockCode','Description','UnitPrice']]
for index, col in  df_check.iterrows():
    if df[(df['CustomerID'] == col[0]) & (df['Quantity'] == -col[1]) & (df['Description'] == col[2])].shape[0] == 0:
        print(df_check.loc[index])
        print(15*'-'+'>'+' HYPOTHESIS NOT FULFILLED')
        break
```

```
CustomerID      14527.0
Quantity             -1
StockCode             D
Description     Discount
UnitPrice          27.5
Name: 141, dtype: object
---------------> HYPOTHESIS NOT FULFILLED
```

Based on test above, we can see the negative quantity also represent discounts (prefix 'D'). Thus the hypothesis "when an order has a negative quantity value, this would always means that the order has been canceled" is not true.