In [129]:
```
'''
DESCRIPTION:

Problem Statement:

1. It is a critical requirement for business to understand the value derived from a customer. RFM is a method used for analyzing customer value.
2. Customer segmentation is the practice of segregating the customer base into groups of individuals based on some common characteristics such as age,
gender, interests, and spending habits
3. Perform customer segmentation using RFM analysis. The resulting segments can be ordered from most valuable (highest recency, frequency, and value)
to least valuable (lowest recency, frequency, and value).
'''
```

Out[129]:
```
'\nDESCRIPTION:\n\nProblem Statement:\n\n1. It is a critical requirement for business to understand the value derived from a customer. RFM is a method used for analyzing customer value.\n2. Customer segmentation is the practice of segregating the customer base into groups of individuals based on some common characteristics such as age, \ngender, interests, and spending habits\n3. Perform customer segmentation using RFM analysis. The resulting segments can be ordered from most valuable (highest recency, frequency, and value) \nto least valuable (lowest recency, frequency, and value).\n'
```

In [130]:
```
'''
Project Task: Week 1
Data Cleaning:

1. Perform a preliminary data inspection and data cleaning.

a. Check for missing data and formulate an apt strategy to treat them.

b. Remove duplicate data records.

c. Perform descriptive analytics on the given data.
'''
```

Out[130]:
```
'\nProject Task: Week 1\nData Cleaning:\n\n1. Perform a preliminary data inspection and data cleaning.\n\na. Check for missing data and formulate an apt strategy to treat them.\n\nb. Remove duplicate data records.\n\nc. Perform descriptive analytics on the given data.\n'
```

In [131]:
```
# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
```

In [132]:
```
# Importing required libraries and dataset
import pandas as pd

train_url = "https://github.com/PraveenBandla/Data-Science-Projects/blob/master/Data-Science-Capstone-Projects/Project%203/Online%20Retail.xlsx?raw=true"
retail_train_df = pd.read_excel(train_url)
retail_train_df
```

Out[132]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
...	...	...	...	...	...	...	...	...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France

541909 rows × 8 columns

In [133]:
```
# Exploratory Data Analysis and Data Preprocessing
retail_train_df.info()

# Observations:
# 541909 rows x 8 columns
# Null values are present in the description and customer ID column
```

```
<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 [134]:
```
# Modifying customer ID type to int
#retail_train_df['CustomerID'] = retail_train_df['CustomerID'].astype('Int64')
#retail_train_df.info()
```

In [135]:
```
# Checking for duplicates
retail_train_df[retail_train_df.duplicated()]

# Observations:
# 5268 duplicates are present in the training dataset
```

Out[135]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
517	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	2010-12-01 11:45:00	1.25	17908.0	United Kingdom
527	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	2010-12-01 11:45:00	2.10	17908.0	United Kingdom
537	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	2010-12-01 11:45:00	2.95	17908.0	United Kingdom
539	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	2010-12-01 11:45:00	4.95	17908.0	United Kingdom
555	536412	22327	ROUND SNACK BOXES SET OF 4 SKULLS	1	2010-12-01 11:49:00	2.95	17920.0	United Kingdom
...	...	...	...	...	...	...	...	...
541675	581538	22068	BLACK PIRATE TREASURE CHEST	1	2011-12-09 11:34:00	0.39	14446.0	United Kingdom
541689	581538	23318	BOX OF 6 MINI VINTAGE CRACKERS	1	2011-12-09 11:34:00	2.49	14446.0	United Kingdom
541692	581538	22992	REVOLVER WOODEN RULER	1	2011-12-09 11:34:00	1.95	14446.0	United Kingdom
541699	581538	22694	WICKER STAR	1	2011-12-09 11:34:00	2.10	14446.0	United Kingdom
541701	581538	23343	JUMBO BAG VINTAGE CHRISTMAS	1	2011-12-09 11:34:00	2.08	14446.0	United Kingdom

5268 rows × 8 columns

In [136]:
```
# Removing duplicates
retail_train_df.drop_duplicates(inplace=True)
retail_train_df.shape
```

Out[136]:
```
(536641, 8)
```

In [137]:
```
# Checking for NULL Values
retail_train_df.isna().sum()

# Observations:
# Description has 1454(~0.27%) missing values whereas customer id has 135,027(~25%) missing values
# Since description column has less number of missing values, we can drop them from the dataset
# But, Customer ID has significant chunk of data missing => We have to impute it using one of the methods
# Customer ID is not a measure => we can't go for mean or median and if we replace it by mode, it will lead to biased dataset
# Instead of replacing missing values with one single value, we can impute it with multiple relative values using KNN or MICE imputation techniques
```

Out[137]:
```
InvoiceNo          0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID    135037
Country            0
dtype: int64
```

In [138]:
```
# Dropping records with null values in description column
retail_train_df = retail_train_df[retail_train_df['Description'].notna()]
retail_train_df.isna().sum()
```

Out[138]:
```
InvoiceNo          0
StockCode          0
Description        0
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID     133583
Country            0
dtype: int64
```

In [139]:
```
# Encoding categorical variables before KNN imputation
from sklearn.preprocessing import LabelEncoder

# Defining Encoder
le = LabelEncoder()

# Categorical columns
cat_col = retail_train_df.columns[retail_train_df.dtypes=="object"]

# Passing train data into a new dataframe
retail_train_enc_df = retail_train_df.copy()

# Fitting and transforming on the train data
for c in cat_col:
  retail_train_enc_df[c] = le.fit_transform(retail_train_enc_df[c].astype('str'))

# Converting date into days
from datetime import datetime

retail_train_enc_df['Days'] = (datetime.today()-retail_train_enc_df['InvoiceDate']).dt.days
retail_train_enc_df.drop('InvoiceDate',axis=1,inplace=True)
retail_train_enc_df
```

Out[139]:

	InvoiceNo	StockCode	Description	Quantity	UnitPrice	CustomerID	Country	Days
0	0	3438	3918	6	2.55	17850.0	36	3712
1	0	2739	3926	6	3.39	17850.0	36	3712
2	0	2975	913	8	2.75	17850.0	36	3712
3	0	2919	1910	6	3.39	17850.0	36	3712
4	0	2918	2911	6	3.39	17850.0	36	3712
...	...	...	...	...	...	...	...	...
541904	20606	1492	2379	12	0.85	12680.0	13	3339
541905	20606	1768	743	6	2.10	12680.0	13	3339
541906	20606	2110	749	4	4.15	12680.0	13	3339
541907	20606	2111	748	4	4.15	12680.0	13	3339
541908	20606	1059	304	3	4.95	12680.0	13	3339

535187 rows × 8 columns

In [140]:
```
# Standardizing the dataset
from sklearn.preprocessing import MinMaxScaler

# Instantiation
mms = MinMaxScaler()

# Fitting and transforming on the encoded training data
retail_train_enc_df.drop('CustomerID',axis=1,inplace=True)
retail_train_enc_df = pd.DataFrame(mms.fit_transform(retail_train_enc_df),columns=retail_train_enc_df.columns)
retail_train_enc_df
```

Out[140]:

	InvoiceNo	StockCode	Description	Quantity	UnitPrice	Country	Days
0	0.000000	0.868840	0.927996	0.500037	0.221150	0.972973	1.0
1	0.000000	0.692191	0.929891	0.500037	0.221167	0.972973	1.0
2	0.000000	0.751832	0.216248	0.500049	0.221154	0.972973	1.0
3	0.000000	0.737680	0.452392	0.500037	0.221167	0.972973	1.0
4	0.000000	0.737427	0.689484	0.500037	0.221167	0.972973	1.0
...	...	...	...	...	...	...	...
535182	0.842954	0.377053	0.563477	0.500074	0.221116	0.351351	0.0
535183	0.842954	0.446803	0.175983	0.500037	0.221141	0.351351	0.0
535184	0.842954	0.533232	0.177404	0.500025	0.221182	0.351351	0.0
535185	0.842954	0.533485	0.177167	0.500025	0.221182	0.351351	0.0
535186	0.842954	0.267627	0.072004	0.500019	0.221198	0.351351	0.0

535187 rows × 7 columns

In [141]:
```
# Appending customer id to encoded and standardized train data
retail_train_enc_df['CustomerID'] = retail_train_df['CustomerID']
retail_train_enc_df
```

Out[141]:

	InvoiceNo	StockCode	Description	Quantity	UnitPrice	Country	Days	CustomerID
0	0.000000	0.868840	0.927996	0.500037	0.221150	0.972973	1.0	17850.0
1	0.000000	0.692191	0.929891	0.500037	0.221167	0.972973	1.0	17850.0
2	0.000000	0.751832	0.216248	0.500049	0.221154	0.972973	1.0	17850.0
3	0.000000	0.737680	0.452392	0.500037	0.221167	0.972973	1.0	17850.0
4	0.000000	0.737427	0.689484	0.500037	0.221167	0.972973	1.0	17850.0
...	...	...	...	...	...	...	...	...
535182	0.842954	0.377053	0.563477	0.500074	0.221116	0.351351	0.0	17449.0
535183	0.842954	0.446803	0.175983	0.500037	0.221141	0.351351	0.0	17449.0
535184	0.842954	0.533232	0.177404	0.500025	0.221182	0.351351	0.0	17449.0
535185	0.842954	0.533485	0.177167	0.500025	0.221182	0.351351	0.0	17449.0
535186	0.842954	0.267627	0.072004	0.500019	0.221198	0.351351	0.0	17449.0

535187 rows × 8 columns

In [142]:
```
# Imputing missing values in customer id column with KNN imputer
from sklearn.impute import KNNImputer

# Defining imputer
imputer = KNNImputer(n_neighbors=3)

# Fitting and transforming on the training data
retail_train_trans_df = pd.DataFrame(imputer.fit_transform(retail_train_enc_df),columns=retail_train_enc_df.columns)

# Checking for missing values after imputing
retail_train_trans_df.isna().sum()
```

Out[142]:
```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
UnitPrice      0
Country        0
Days           0
CustomerID     0
dtype: int64
```

In [143]:
```
# Unique Number of values in each column
import numpy as np

retail_train_trans_df['CustomerID'] = np.round(retail_train_trans_df['CustomerID'])
retail_train_trans_df
```

Out[143]:

	InvoiceNo	StockCode	Description	Quantity	UnitPrice	Country	Days	CustomerID
0	0.000000	0.868840	0.927996	0.500037	0.221150	0.972973	1.0	17850.0
1	0.000000	0.692191	0.929891	0.500037	0.221167	0.972973	1.0	17850.0
2	0.000000	0.751832	0.216248	0.500049	0.221154	0.972973	1.0	17850.0
3	0.000000	0.737680	0.452392	0.500037	0.221167	0.972973	1.0	17850.0
4	0.000000	0.737427	0.689484	0.500037	0.221167	0.972973	1.0	17850.0
...	...	...	...	...	...	...	...	...
535182	0.842954	0.377053	0.563477	0.500074	0.221116	0.351351	0.0	17449.0
535183	0.842954	0.446803	0.175983	0.500037	0.221141	0.351351	0.0	17449.0
535184	0.842954	0.533232	0.177404	0.500025	0.221182	0.351351	0.0	17449.0
535185	0.842954	0.533485	0.177167	0.500025	0.221182	0.351351	0.0	17449.0
535186	0.842954	0.267627	0.072004	0.500019	0.221198	0.351351	0.0	17449.0

535187 rows × 8 columns

In [144]:
```
# Reset index of original training data
retail_train_df.reset_index(drop=True,inplace=True)
```

In [145]:
```
# Replacing customer ID in original train dataset with the imputed column
retail_train_df['CustomerID'] = retail_train_trans_df['CustomerID']
retail_train_df.isna().sum()

# Observations:
# Customer ID column has zero missing values as they are imputed using KNN imputation technique
```

Out[145]:
```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

In [146]:
```
# Number of unique values in each column
retail_train_df.nunique()
```

Out[146]:
```
InvoiceNo      24446
StockCode       3958
Description     4223
Quantity         671
InvoiceDate    22309
UnitPrice       1630
CustomerID      5838
Country           38
dtype: int64
```

In [147]:
```
# Computing Sales value of a transaction
retail_train_df['Sales'] = retail_train_df['Quantity']*retail_train_df['UnitPrice']
retail_train_df
```

Out[147]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Sales
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	15.30
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	22.00
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34
...	...	...	...	...	...	...	...	...	...
535182	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	17449.0	France	10.20
535183	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	17449.0	France	12.60
535184	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	17449.0	France	16.60
535185	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	17449.0	France	16.60
535186	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	17449.0	France	14.85

535187 rows × 9 columns

In [148]:
```
#!pip install -U plotly
```

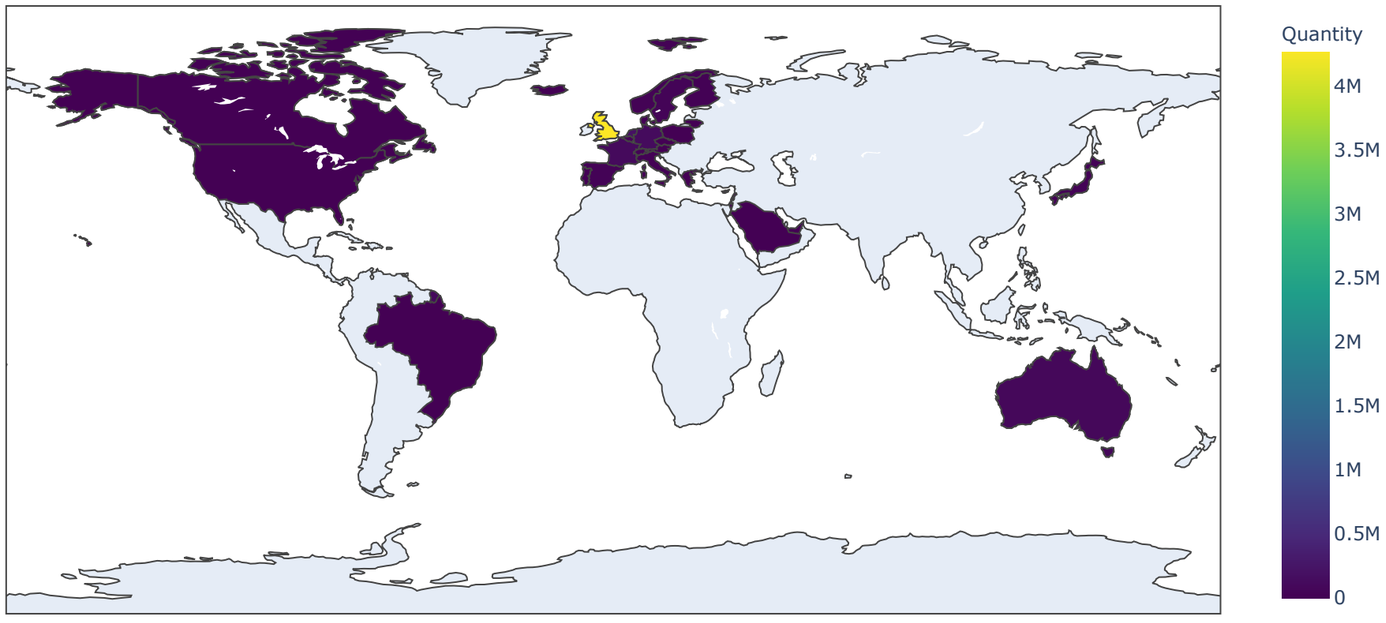
In [149]:
```
# Order size by Country
country_group = retail_train_df.groupby('Country')
country_qty_df = pd.DataFrame(country_group['Quantity'].agg(np.sum))
country_qty_df.sort_values('Quantity',ascending=False,inplace=True)
country_qty_df
```

Out[149]:

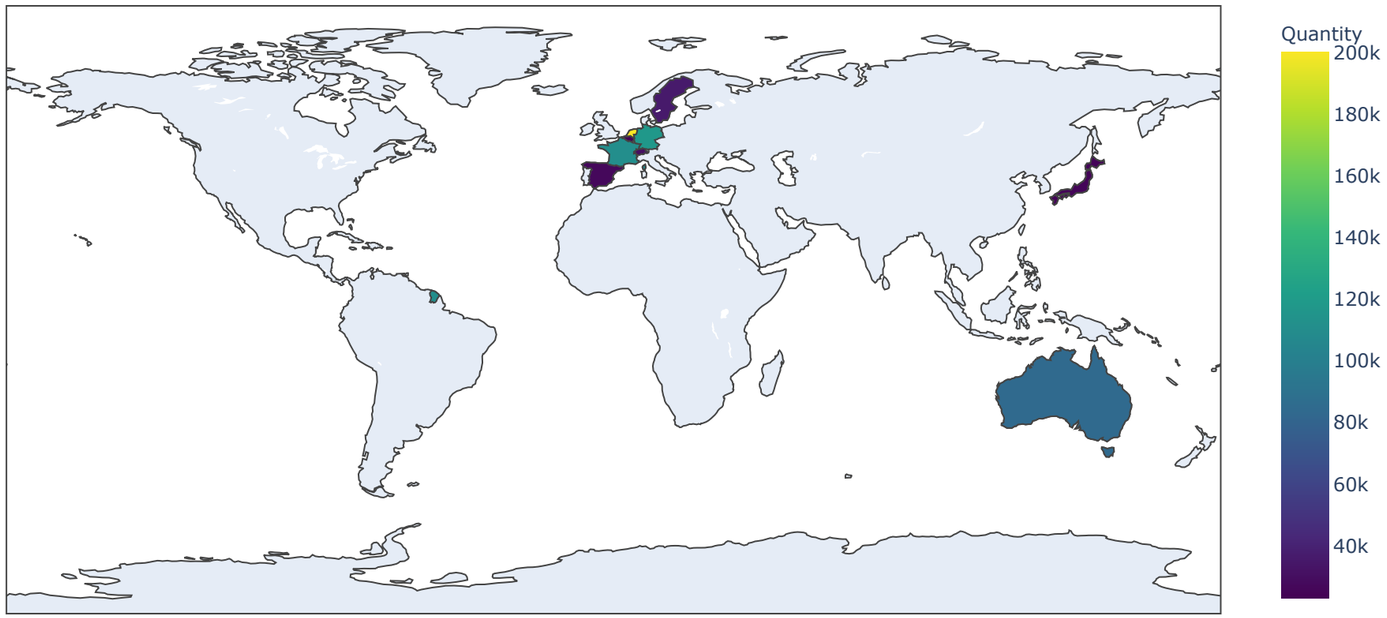
Country	Quantity
United Kingdom	4263937
Netherlands	200128
EIRE	142495
Germany	117341
France	110438
Australia	83643
Sweden	35632
Switzerland	30313
Spain	26817
Japan	25218
Belgium	23152
Norway	19247
Portugal	16153
Finland	10666
Channel Islands	9473
Denmark	8188
Italy	7999
Cyprus	6296
Singapore	5234
Austria	4827
Hong Kong	4709
Israel	4350
Poland	3653
Unspecified	3295
Canada	2763
Iceland	2458
Greece	1556
USA	1034
United Arab Emirates	982
Malta	944
Lithuania	652
Czech Republic	592
European Community	497

Country	Quantity
Lebanon	386
Brazil	356
RSA	352
Bahrain	260
Saudi Arabia	75

In [150]:
```
# Perform descriptive analytics on the given data.
# Orders from different countries
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
import seaborn as sns


fig = px.choropleth(country_qty_df,
                    locations=country_qty_df.index,
                    locationmode='country names',
                    color='Quantity',
                    color_continuous_scale='viridis')
fig.show()

# Observations:
# Since It is a UK-based online retailer, majority of the orders are from UK which is dwarfing the order from other countries
```



In [151]:
```
# Orders from top 10 countries other than UK
fig = px.choropleth(country_qty_df[1:11],
                    locations=country_qty_df.index[1:11],
                    locationmode='country names',
                    color='Quantity',
                    color_continuous_scale='viridis')
fig.show()

# Observations:
# Majority of the top 10 countries (other than UK) ordering are from Europe with only exception being Australia and Japan
```



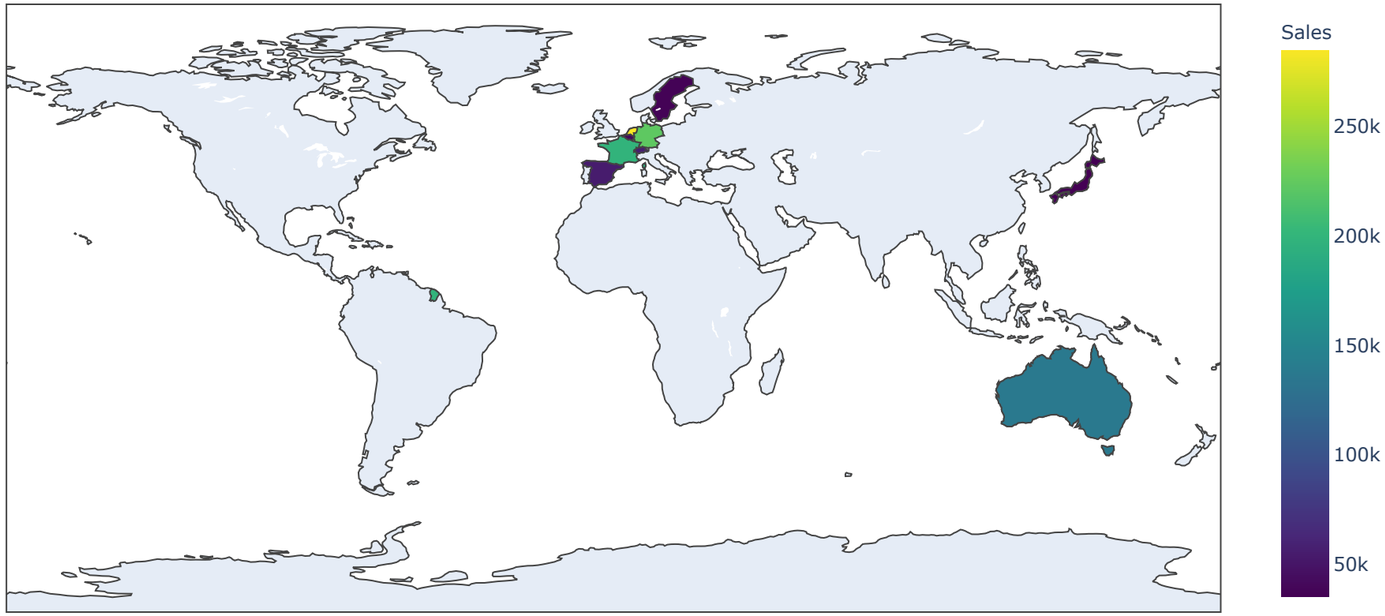
In [152]:
```
# Sales by Country
country_sls_df = pd.DataFrame(country_group['Sales'].agg(np.sum))
country_sls_df.sort_values('Sales',ascending=False,inplace=True)
country_sls_df

# Observations:
# UK tops the chart with 8.16Mn in Sales
```

Out[152]:

Country	Sales
United Kingdom	8.167128e+06
Netherlands	2.846615e+05
EIRE	2.629934e+05
Germany	2.215095e+05
France	1.973171e+05
Australia	1.370098e+05
Switzerland	5.636305e+04
Spain	5.475603e+04
Belgium	4.091096e+04
Sweden	3.658541e+04
Japan	3.534062e+04
Norway	3.516346e+04
Portugal	2.930297e+04
Finland	2.232674e+04
Channel Islands	2.007639e+04
Denmark	1.876814e+04
Italy	1.689051e+04
Cyprus	1.285876e+04
Austria	1.015432e+04
Hong Kong	9.908240e+03
Singapore	9.120390e+03
Israel	7.901970e+03
Poland	7.213140e+03
Unspecified	4.740940e+03
Greece	4.710520e+03
Iceland	4.310000e+03
Canada	3.666380e+03
Malta	2.505470e+03
United Arab Emirates	1.902280e+03
USA	1.730920e+03
Lebanon	1.693880e+03
Lithuania	1.661060e+03

Country	Sales
European Community	1.291750e+03
Brazil	1.143600e+03
RSA	1.002310e+03
Czech Republic	7.077200e+02
Bahrain	5.484000e+02
Saudi Arabia	1.311700e+02

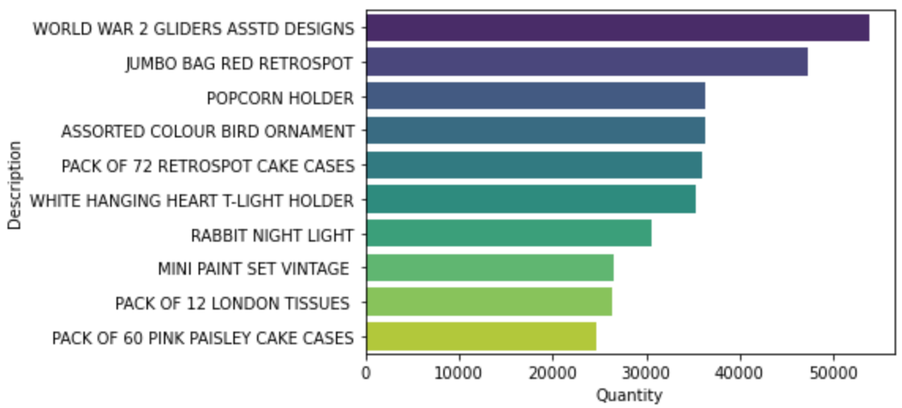
In [153]:
```
# Sales from top 10 countries other than UK
fig = px.choropleth(country_sls_df[1:11],
                    locations=country_sls_df.index[1:11],
                    locationmode='country names',
                    color='Sales',
                    color_continuous_scale='viridis')
fig.show()

# Observations:
# Sales from the top 10 countries (other than UK) are dominated by European countries with only exception being Australia
# and japan
```



In [154]:
```
# Top 10 products by quantity sold
product_group = retail_train_df.groupby('Description')
product_df = pd.DataFrame(product_group['Quantity'].agg(np.sum).nlargest(10))
print(product_df,"\n")

# Bar Plot
sns.barplot(x=product_df.Quantity,y=product_df.index,palette='viridis')

# Observations:
# WW2 Gliders have been sold the most (53,751 units), followed by bags, cases, holders, tissues and ornaments
```

```
                                  Quantity
Description
WORLD WAR 2 GLIDERS ASSTD DESIGNS     53751
JUMBO BAG RED RETROSPOT               47260
POPCORN HOLDER                        36322
ASSORTED COLOUR BIRD ORNAMENT         36282
PACK OF 72 RETROSPOT CAKE CASES       36016
WHITE HANGING HEART T-LIGHT HOLDER    35298
RABBIT NIGHT LIGHT                    30631
MINI PAINT SET VINTAGE                26437
PACK OF 12 LONDON TISSUES             26299
PACK OF 60 PINK PAISLEY CAKE CASES    24719
```

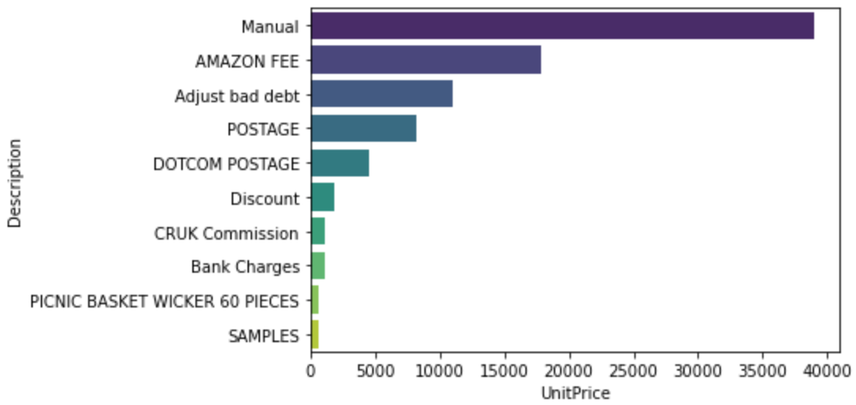
Out[154]: <AxesSubplot:xlabel='Quantity', ylabel='Description'>



In [155]:
```
# Top 10 products by Unit Price
product_df = pd.DataFrame(product_group['UnitPrice'].agg(np.max).nlargest(10))
print(product_df,"\n")

# Bar Plot
sns.barplot(x=product_df.UnitPrice,y=product_df.index,palette='viridis')

# Observations:
# fees, commissions, postage and bank charges featured in top items with high unit price
```

```
                               UnitPrice
Description
Manual                          38970.00
AMAZON FEE                      17836.46
Adjust bad debt                 11062.06
POSTAGE                          8142.75
DOTCOM POSTAGE                   4505.17
Discount                         1867.86
CRUK Commission                  1100.44
Bank Charges                     1050.15
PICNIC BASKET WICKER 60 PIECES    649.50
SAMPLES                           570.00
```

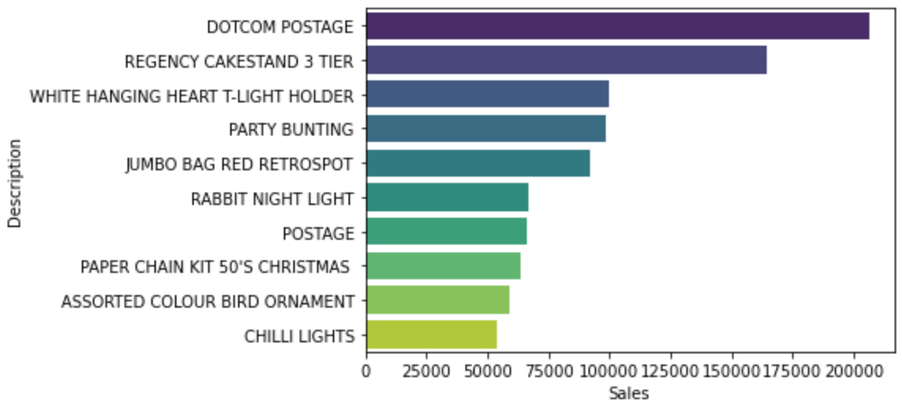
Out[155]: <AxesSubplot:xlabel='UnitPrice', ylabel='Description'>



In [156]:
```
# Top 10 products by Value
product_df = pd.DataFrame(product_group['Sales'].agg(np.sum).nlargest(10))
print(product_df,"\n")

# Bar Plot
sns.barplot(x=product_df.Sales,y=product_df.index,palette='viridis')

# Observations:
# DOTCOm Postage and cake Stand accounted for ~370,000 sales followed by holders, bags, ornament and lights
```

```
                                    Sales
Description
DOTCOM POSTAGE                   206245.48
REGENCY CAKESTAND 3 TIER         164459.49
WHITE HANGING HEART T-LIGHT HOLDER  99612.42
PARTY BUNTING                     98243.88
JUMBO BAG RED RETROSPOT           92175.79
RABBIT NIGHT LIGHT                66661.63
POSTAGE                           66230.64
PAPER CHAIN KIT 50'S CHRISTMAS    63715.24
ASSORTED COLOUR BIRD ORNAMENT     58792.42
CHILLI LIGHTS                     53746.66
```

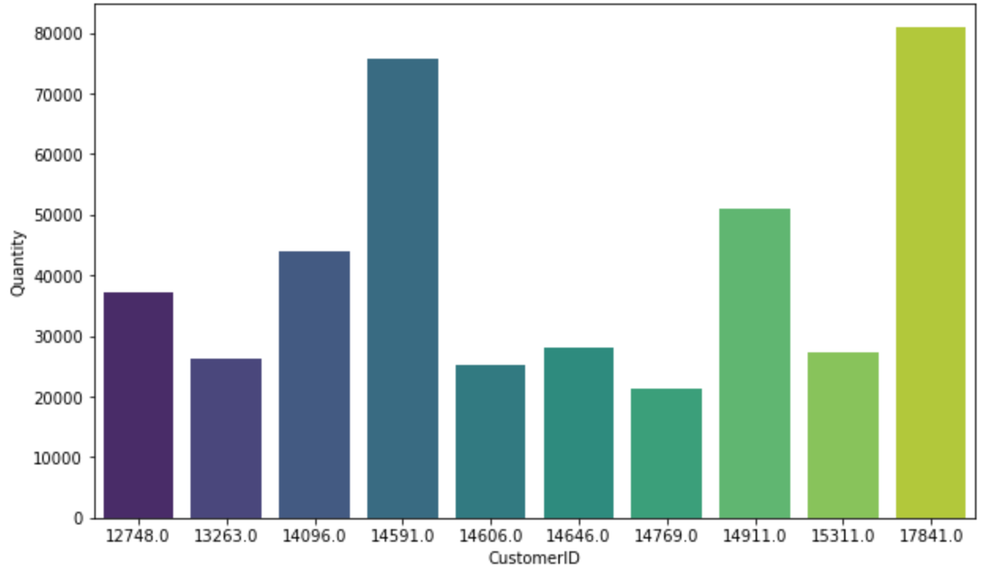
Out[156]: <AxesSubplot:xlabel='Sales', ylabel='Description'>



In [157]:
```
# Top 10 Customers by Quantity
customer_group = retail_train_df.groupby('CustomerID')
customer_df = pd.DataFrame(customer_group['Quantity'].agg(np.sum).nlargest(10))
print(customer_df,"\n")

# Bar Plot
plt.figure(figsize=(10,6))
sns.barplot(y=customer_df.Quantity,x=customer_df.index,palette='viridis')

# Observations:
# Top 5 customers accounted for ~290,000 units of purchase
```

```
            Quantity
CustomerID
17841.0        80901
14591.0        75626
14911.0        51045
14096.0        43914
12748.0        37301
14646.0        28100
15311.0        27335
13263.0        26370
14606.0        25253
14769.0        21370
```

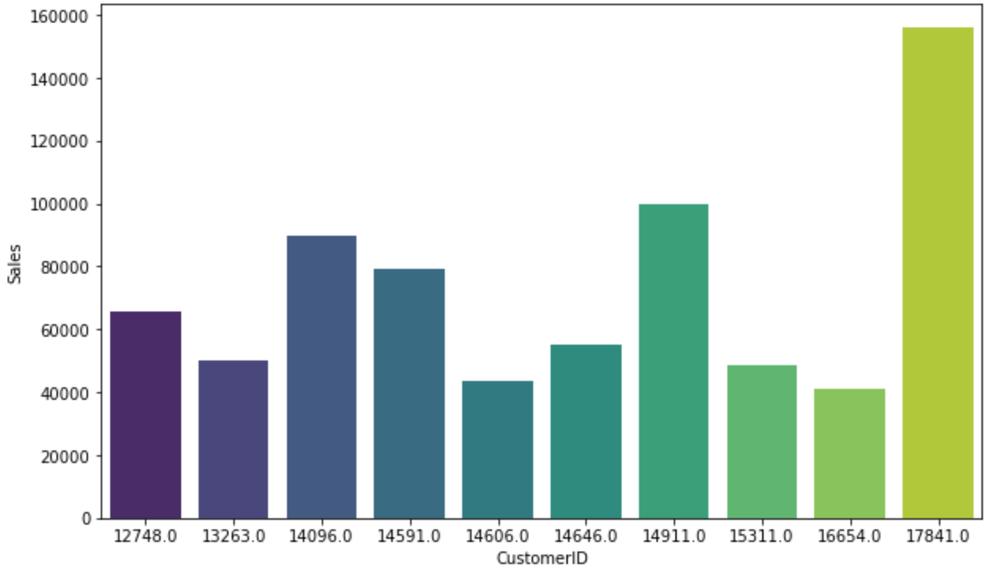
Out[157]: <AxesSubplot:xlabel='CustomerID', ylabel='Quantity'>



In [158]:
```
# Top 10 Customers by Value
customer_group = retail_train_df.groupby('CustomerID')
customer_df = pd.DataFrame(customer_group['Sales'].agg(np.sum).nlargest(10))
print(customer_df,"\n")

# Bar Plot
plt.figure(figsize=(10,6))
sns.barplot(y=customer_df.Sales,x=customer_df.index,palette='viridis')

# Observations:
# 9/10 customers who accounted for high quantity purchase also featured in top 10 sales category
```

```
                Sales
CustomerID
17841.0     155969.59
14911.0      99615.79
14096.0      89798.92
14591.0      79116.27
12748.0      65730.64
14646.0      55180.32
13263.0      50147.44
15311.0      48505.47
14606.0      43571.61
16654.0      41101.12
```

Out[158]: <AxesSubplot:xlabel='CustomerID', ylabel='Sales'>



In [159]:
```
# Creating date column from datetime column
retail_train_df['Date'] = retail_train_df['InvoiceDate'].dt.date
retail_train_df
```

Out[159]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Sales	Date
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	15.30	2010-12-01
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	2010-12-01
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	22.00	2010-12-01
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	2010-12-01
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	2010-12-01
...	...	...	...	...	...	...	...	...	...	...
535182	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	17449.0	France	10.20	2011-12-09
535183	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	17449.0	France	12.60	2011-12-09
535184	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	17449.0	France	16.60	2011-12-09
535185	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	17449.0	France	16.60	2011-12-09
535186	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	17449.0	France	14.85	2011-12-09

535187 rows × 10 columns

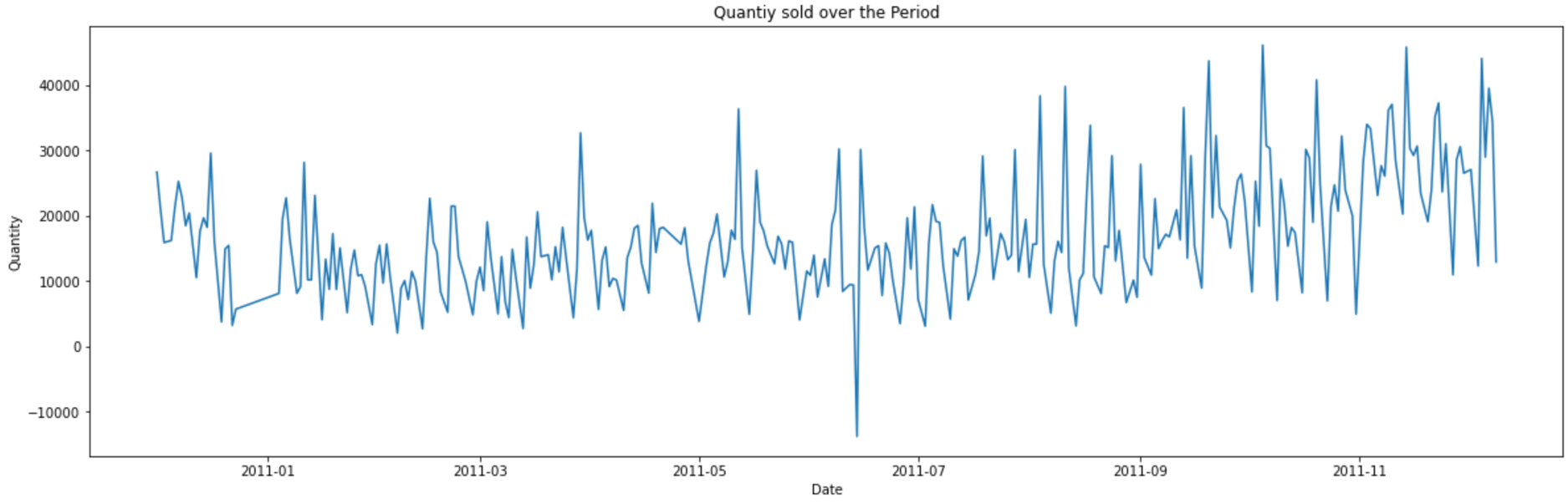
In [160]:
```
# Trend Analysis
date_group = retail_train_df.groupby('Date')
date_df = pd.DataFrame(date_group['Quantity','Sales'].agg(np.sum))
date_df
```

Out[160]:

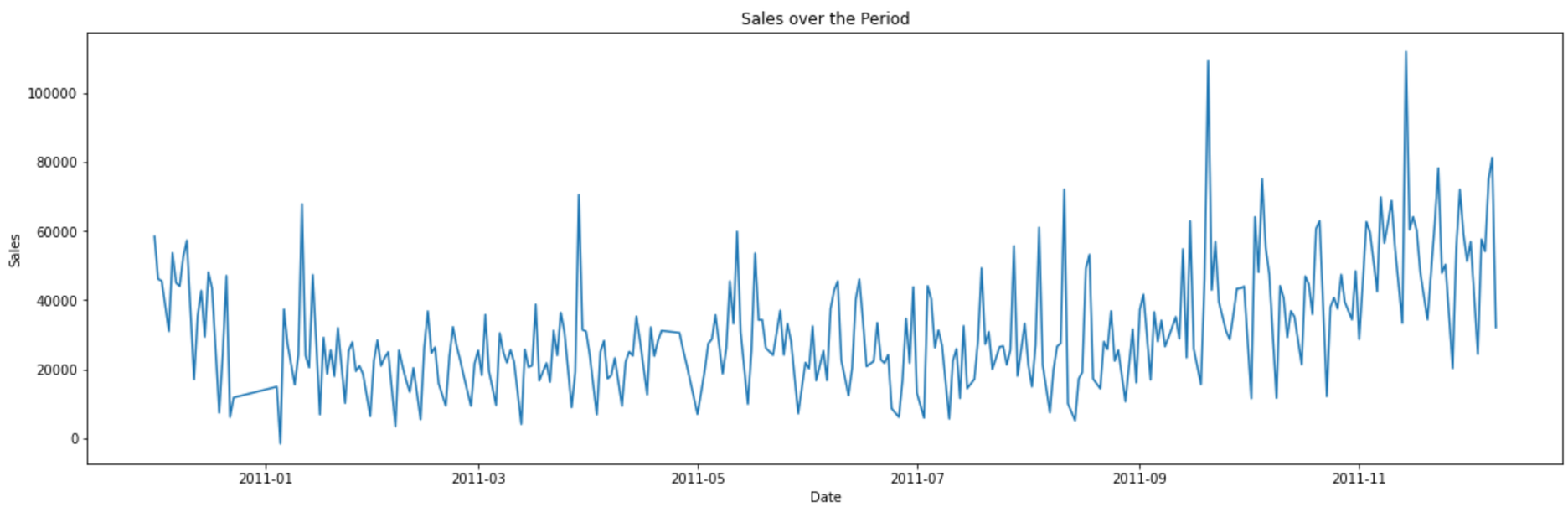
Date	Quantity	Sales
2010-12-01	26635	58451.56
2010-12-02	20977	46088.32
2010-12-03	15872	45575.38
2010-12-05	16187	30973.63
2010-12-06	21222	53653.87
...	...	...
2011-12-05	44023	57630.20
2011-12-06	28946	54109.39
2011-12-07	39478	74952.61
2011-12-08	34338	81294.33
2011-12-09	12919	32111.91

305 rows × 2 columns

In [161]:
```
# Quantity trend over the period

plt.figure(figsize=(20,6))
sns.lineplot(data=date_df,x=date_df.index,y='Quantity')
plt.title("Quantiy sold over the Period")
plt.show()
```



In [162]:
```
# Sales trend over the period

plt.figure(figsize=(20,6))
sns.lineplot(data=date_df,x=date_df.index,y='Sales')
plt.title("Sales over the Period")
plt.show()

# Observations:
# On an average, Quantity sold and sales have increased over the period
```



In [163]:
```
'''
Data Transformation:

2. Perform cohort analysis (a cohort is a group of subjects that share a defining characteristic).
   Observe how a cohort behaves across time and compare it to other cohorts.

a. Create month cohorts and analyze active customers for each cohort.

b. Analyze the retention rate of customers.
'''
```

Out[163]:
```
'\nData Transformation:\n\n2. Perform cohort analysis (a cohort is a group of subjects that share a defining characteristic). \n   Observe how a cohort behaves across time and compare it to other cohorts.\n\na. Create month cohorts and analyze active customers for each cohort.\n\nb. Analyze the retention rate of customers.\n'
```

In [164]:
```
# Cohort Analysis
# Creating a column with only year and month

retail_train_df['OrderMonth'] = retail_train_df['InvoiceDate'].dt.to_period('M')
retail_train_df
```

Out[164]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Sales	Date	OrderMonth
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	15.30	2010-12-01	2010-12
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	2010-12-01	2010-12
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	22.00	2010-12-01	2010-12
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	2010-12-01	2010-12
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	2010-12-01	2010-12
...	...	...	...	...	...	...	...	...	...	...	...
535182	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	17449.0	France	10.20	2011-12-09	2011-12
535183	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	17449.0	France	12.60	2011-12-09	2011-12
535184	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	17449.0	France	16.60	2011-12-09	2011-12
535185	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	17449.0	France	16.60	2011-12-09	2011-12
535186	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	17449.0	France	14.85	2011-12-09	2011-12

535187 rows × 11 columns

In [165]:
```
# Creating cohorts based on first purchase period (month) of a customer
retail_train_df['Cohort'] = retail_train_df.groupby('CustomerID')['InvoiceDate'].transform('min').dt.to_period('M')
retail_train_df
```

Out[165]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Sales	Date	OrderMonth	Cohort
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	15.30	2010-12-01	2010-12	2010-12
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	2010-12-01	2010-12	2010-12
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	22.00	2010-12-01	2010-12	2010-12
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	2010-12-01	2010-12	2010-12
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	2010-12-01	2010-12	2010-12
...	...	...	...	...	...	...	...	...	...	...	...	...
535182	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	17449.0	France	10.20	2011-12-09	2011-12	2010-12
535183	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	17449.0	France	12.60	2011-12-09	2011-12	2010-12
535184	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	17449.0	France	16.60	2011-12-09	2011-12	2010-12
535185	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	17449.0	France	16.60	2011-12-09	2011-12	2010-12
535186	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	17449.0	France	14.85	2011-12-09	2011-12	2010-12

535187 rows × 12 columns

In [166]:
```
# Computing unique customers in each cohort during the period
cohort_df = retail_train_df.groupby(['Cohort', 'OrderMonth']).agg(N_Customers=('CustomerID', 'nunique')).reset_index(drop=False)
cohort_df
```

Out[166]:

	Cohort	OrderMonth	N_Customers
0	2010-12	2010-12	4050
1	2010-12	2011-01	3092
2	2010-12	2011-02	2626
3	2010-12	2011-03	2881
4	2010-12	2011-04	2563
...	...	...	...
86	2011-10	2011-11	16
87	2011-10	2011-12	11
88	2011-11	2011-11	52
89	2011-11	2011-12	4
90	2011-12	2011-12	13

91 rows × 3 columns

In [167]:
```
# Computing period number for cohorts
from operator import attrgetter

cohort_df['PeriodNumber'] = (cohort_df.OrderMonth - cohort_df.Cohort).apply(attrgetter('n'))
cohort_df
```

Out[167]:

	Cohort	OrderMonth	N_Customers	PeriodNumber
0	2010-12	2010-12	4050	0
1	2010-12	2011-01	3092	1
2	2010-12	2011-02	2626	2
3	2010-12	2011-03	2881	3
4	2010-12	2011-04	2563	4
...	...	...	...	...
86	2011-10	2011-11	16	1
87	2011-10	2011-12	11	2
88	2011-11	2011-11	52	0
89	2011-11	2011-12	4	1
90	2011-12	2011-12	13	0

91 rows × 4 columns

In [168]:
```
# Creating a table with cohorts in rows and period number in columns with Number of customers as values

cohort_pivot = cohort_df.pivot_table(index = 'Cohort',
                                     columns = 'PeriodNumber',
                                     values = 'N_Customers')
cohort_pivot
```

Out[168]:

PeriodNumber	0	1	2	3	4	5	6	7	8	9	10	11	12
Cohort													
2010-12	4050.0	3092.0	2626.0	2881.0	2563.0	2752.0	2895.0	3120.0	2458.0	3158.0	3099.0	3550.0	2536.0
2011-01	810.0	402.0	435.0	366.0	409.0	434.0	500.0	340.0	455.0	469.0	588.0	322.0	NaN
2011-02	268.0	118.0	86.0	115.0	111.0	136.0	110.0	126.0	121.0	168.0	71.0	NaN	NaN
2011-03	185.0	61.0	71.0	60.0	82.0	49.0	78.0	75.0	99.0	53.0	NaN	NaN	NaN
2011-04	89.0	26.0	25.0	31.0	23.0	28.0	34.0	41.0	17.0	NaN	NaN	NaN	NaN
2011-05	98.0	33.0	32.0	31.0	34.0	26.0	39.0	20.0	NaN	NaN	NaN	NaN	NaN
2011-06	68.0	23.0	13.0	25.0	30.0	38.0	19.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07	76.0	16.0	29.0	25.0	24.0	12.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08	40.0	11.0	15.0	19.0	11.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09	46.0	12.0	18.0	8.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10	43.0	16.0	11.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11	52.0	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12	13.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [169]:
```
# Overall retention values
Total_df = pd.DataFrame(cohort_pivot.sum(axis=0)).rename(columns={0: 'Total Value'}).transpose()
Total_Pct_df = Total_df.divide(Total_df.iloc[:,0],axis=0)
Total_Pct_df
```

Out[169]:

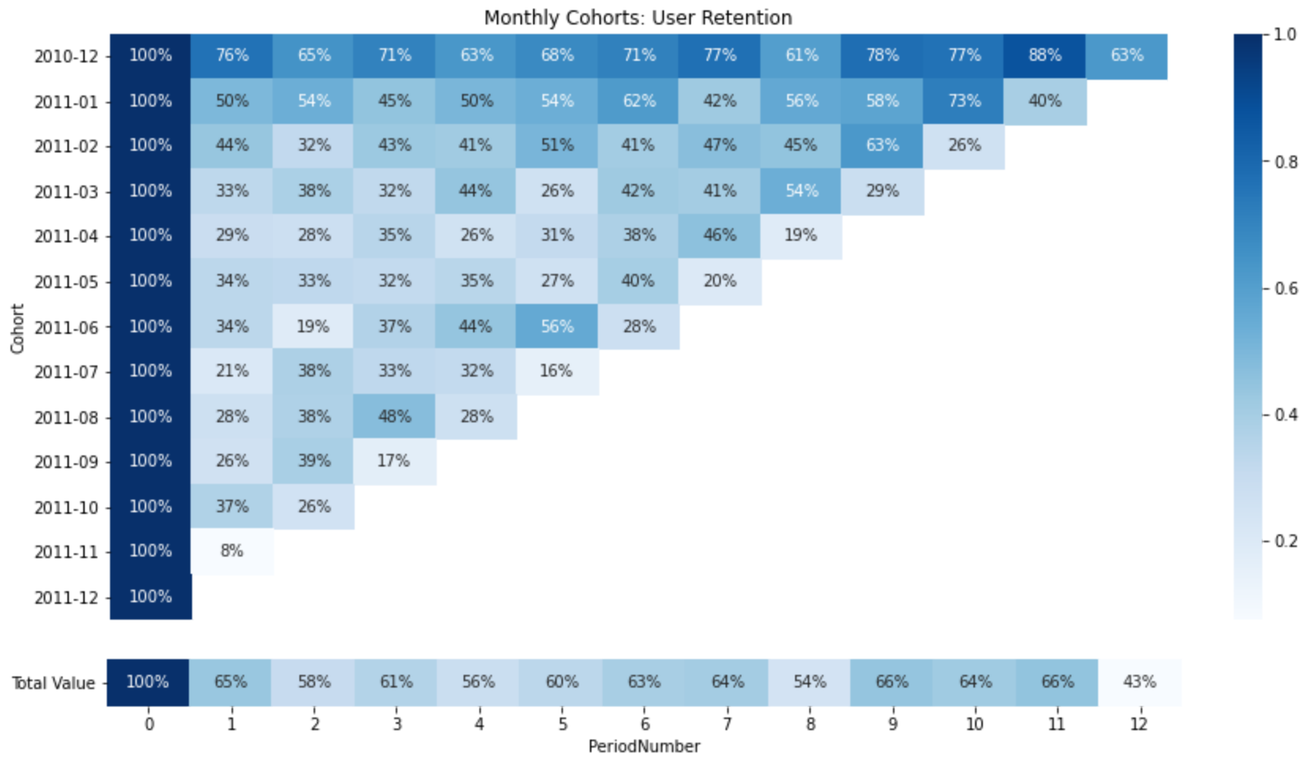
PeriodNumber	0	1	2	3	4	5	6	7	8	9	10	11	12
Total Value	1.0	0.653306	0.575711	0.609969	0.563035	0.595238	0.629496	0.637547	0.539568	0.65913	0.643714	0.663241	0.434395

In [170]:
```
# Divide each element in a row by its first element to fetch retention percentages

cohort_size = cohort_pivot.iloc[:,0]
retention_matrix = cohort_pivot.divide(cohort_size, axis = 0)

retention_matrix
```

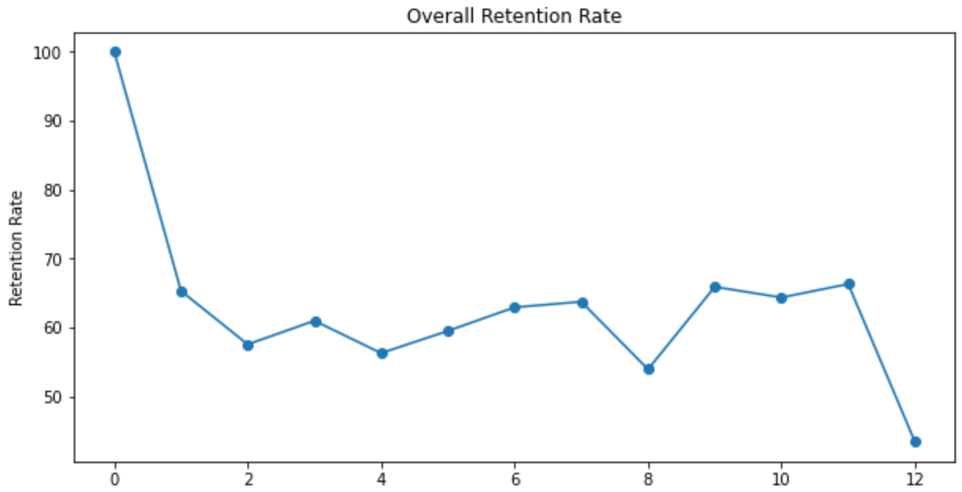
Out[170]:

PeriodNumber	0	1	2	3	4	5	6	7	8	9	10	11	12
Cohort													
2010-12	1.0	0.763457	0.648395	0.711358	0.632840	0.679506	0.714815	0.770370	0.606914	0.779753	0.765185	0.876543	0.626173
2011-01	1.0	0.496296	0.537037	0.451852	0.504938	0.535802	0.617284	0.419753	0.561728	0.579012	0.725926	0.397531	NaN
2011-02	1.0	0.440299	0.320896	0.429104	0.414179	0.507463	0.410448	0.470149	0.451493	0.626866	0.264925	NaN	NaN
2011-03	1.0	0.329730	0.383784	0.324324	0.443243	0.264865	0.421622	0.405405	0.535135	0.286486	NaN	NaN	NaN
2011-04	1.0	0.292135	0.280899	0.348315	0.258427	0.314607	0.382022	0.460674	0.191011	NaN	NaN	NaN	NaN
2011-05	1.0	0.336735	0.326531	0.316327	0.346939	0.265306	0.397959	0.204082	NaN	NaN	NaN	NaN	NaN
2011-06	1.0	0.338235	0.191176	0.367647	0.441176	0.558824	0.279412	NaN	NaN	NaN	NaN	NaN	NaN
2011-07	1.0	0.210526	0.381579	0.328947	0.315789	0.157895	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08	1.0	0.275000	0.375000	0.475000	0.275000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09	1.0	0.260870	0.391304	0.173913	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10	1.0	0.372093	0.255814	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11	1.0	0.076923	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [171]:
```
# Plotting User Retention Matrix

plt.figure(figsize=(15,15))
plt.subplot(2,1,1)
sns.heatmap(retention_matrix,mask=retention_matrix.isna(),annot=True,cmap='Blues',fmt=".0%")
plt.title("Monthly Cohorts: User Retention")
plt.xticks(ticks=[])
plt.xlabel("")

plt.figure(figsize=(12.2,1.2))
plt.subplot(2,1,2)
sns.heatmap(Total_Pct_df,annot=True,cmap='Blues',fmt=".0%",cbar=False)
plt.yticks(rotation=0)
plt.show()

# Observations:
# Initial cohorts have higher retention rates compared to the recent ones
# Retention rates increased in September, October and November 2011 before dropping in December
# Since, We have partial data for December 2011, We should not take that month into consideration
```



In [172]:
```
# Plotting overall retention trend
plt.figure(figsize=(10,5))
plt.plot(Total_Pct_df.transpose().mul(100),marker='o')
plt.title("Overall Retention Rate")
plt.xlabel("Period Number")
plt.ylabel("Retention Rate")
plt.show()

# Observations:
# Retention rates decreased initially before surging in later periods
# Overall retention rate lies between 54% to 65% excluding December 2011
```



In [173]:
```
'''
Data Modeling :

1. Build a RFM (Recency Frequency Monetary) model. Recency means the number of days since a customer made the last purchase.
Frequency is the number of purchase in a given period. It could be 3 months, 6 months or 1 year.
Monetary is the total amount of money a customer spent in that given period. Therefore, big spenders will be differentiated among other customers
such as MVP (Minimum Viable Product) or VIP.

2. Calculate RFM metrics.

3. Build RFM Segments. Give recency, frequency, and monetary scores individually by dividing them into quartiles.

b1. Combine three ratings to get a RFM segment (as strings).

b2. Get the RFM score by adding up the three ratings.

b3. Analyze the RFM segments by summarizing them and comment on the findings.

Note: Rate “recency" for customer who has been active more recently higher than the less recent customer, because each company wants its customers to be recent.

Note: Rate “frequency" and “monetary" higher, because the company wants the customer to visit more often and spend more money
'''
```

Out[173]:
```
'\nData Modeling :\n\n1. Build a RFM (Recency Frequency Monetary) model. Recency means the number of days since a customer made the last purchase. \nFrequency is the number of purchase in a given period. It could be 3 months, 6 months or 1 year. \nMonetary is the total amount of money a customer spent in that given period. Therefore, big spenders will be differentiated among other customers\nsuch as MVP (Minimum Viable Product) or VIP.\n\n2. Calculate RFM metrics.\n\n3. Build RFM Segments. Give recency, frequency, and monetary scores individually by dividing them into quartiles.\n\nb1. Combine three ratings to get a RFM segment (as strings).\n\nb2. Get the RFM score by adding up the three ratings.\n\nb3. Analyze the RFM segments by summarizing them and comment on the findings.\n\nNote: Rate “recency" for customer who has been active more recently higher than the less recent customer, because each company wants its customers to be recent.\n\nNote: Rate “frequency" and “monetary" higher, because the company wants the customer to visit more often and spend more money\n'
```

In [174]:
```
# Create reference date to compute recency
from datetime import timedelta

ref_date = retail_train_df['InvoiceDate'].max() + timedelta(days=1)
print(ref_date)
```

```
2011-12-10 12:50:00
```

In [223]:
```
# Computing Recency, Frequency and Monetary value
rfm_df = retail_train_df.groupby(['CustomerID']).agg({
            'InvoiceDate': lambda x: (ref_date - x.max()).days,
            'InvoiceNo': 'count',
            'Sales': 'sum'})

rfm_df
```

Out[223]:

CustomerID	InvoiceDate	InvoiceNo	Sales
12346.0	325	2	26.44
12347.0	1	191	2804.99
12348.0	73	31	717.89
12349.0	17	73	820.87
12350.0	309	17	325.80
...	...	...	...
18280.0	277	10	52.05
18281.0	179	7	37.07
18282.0	4	13	878.61
18283.0	2	726	10132.56
18287.0	40	71	1311.37

5838 rows × 3 columns

In [224]:
```
# Renaming columns
rfm_df.rename(columns={'InvoiceDate':'Recency','InvoiceNo':'Frequency','Sales':'MonetaryValue'},inplace=True)
rfm_df
```

Out[224]:

CustomerID	Recency	Frequency	MonetaryValue
12346.0	325	2	26.44
12347.0	1	191	2804.99
12348.0	73	31	717.89
12349.0	17	73	820.87
12350.0	309	17	325.80
...	...	...	...
18280.0	277	10	52.05
18281.0	179	7	37.07
18282.0	4	13	878.61
18283.0	2	726	10132.56
18287.0	40	71	1311.37

5838 rows × 3 columns

In [225]:
```
# Computing RFM Scores by dividing them into quartiles
r_group = pd.qcut(rfm_df['Recency'].rank(method='first'),q=4,labels=range(4,0,-1))
f_group = pd.qcut(rfm_df['Frequency'],q=4,labels=range(1,5))
m_group = pd.qcut(rfm_df['MonetaryValue'],q=4,labels=range(1,5))

# Appending R,F and M scores to the data frame
rfm_df = rfm_df.assign(R=r_group.values,F=f_group.values,M=m_group.values)
rfm_df
```

Out[225]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M
12346.0	325	2	26.44	1	1	1
12347.0	1	191	2804.99	4	4	4
12348.0	73	31	717.89	1	2	2
12349.0	17	73	820.87	2	3	3
12350.0	309	17	325.80	1	1	1
...	...	...	...	...	...	...
18280.0	277	10	52.05	1	1	1
18281.0	179	7	37.07	1	1	1
18282.0	4	13	878.61	3	1	3
18283.0	2	726	10132.56	4	4	4
18287.0	40	71	1311.37	1	3	3

5838 rows × 6 columns

In [227]:
```
# Concatenating RFM values to create RFM segments
def rfm_seg(x):
  return str(x['R'])+str(x['F'])+str(x['M'])

rfm_df['RFMSegment'] = rfm_df.apply(rfm_seg,axis=1)
rfm_df
```

Out[227]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFMSegment
12346.0	325	2	26.44	1	1	1	111
12347.0	1	191	2804.99	4	4	4	444
12348.0	73	31	717.89	1	2	2	122
12349.0	17	73	820.87	2	3	3	233
12350.0	309	17	325.80	1	1	1	111
...	...	...	...	...	...	...	...
18280.0	277	10	52.05	1	1	1	111
18281.0	179	7	37.07	1	1	1	111
18282.0	4	13	878.61	3	1	3	313
18283.0	2	726	10132.56	4	4	4	444
18287.0	40	71	1311.37	1	3	3	133

5838 rows × 7 columns

In [228]:
```
# Number of Segments
rfm_df['RFMSegment'].nunique()

# Observations:
# 60 unique segments are present
```

Out[228]: 64

In [229]:
```
# Computing RFM Score
rfm_df['RFMScore'] = rfm_df[['R','F','M']].sum(axis=1)
rfm_df
```

Out[229]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFMSegment	RFMScore
12346.0	325	2	26.44	1	1	1	111	3
12347.0	1	191	2804.99	4	4	4	444	12
12348.0	73	31	717.89	1	2	2	122	5
12349.0	17	73	820.87	2	3	3	233	8
12350.0	309	17	325.80	1	1	1	111	3
...	...	...	...	...	...	...	...	...
18280.0	277	10	52.05	1	1	1	111	3
18281.0	179	7	37.07	1	1	1	111	3
18282.0	4	13	878.61	3	1	3	313	7
18283.0	2	726	10132.56	4	4	4	444	12
18287.0	40	71	1311.37	1	3	3	133	7

5838 rows × 8 columns

In [230]:
```
# Grouping customers into 6 RFM levels
# Define rfm_level function
def rfm_level(df):
    if df['RFMScore'] >= 9:
        return 'Can\'t Loose Them'
    elif ((df['RFMScore'] >= 8) and (df['RFMScore'] < 9)):
        return 'Champions'
    elif ((df['RFMScore'] >= 7) and (df['RFMScore'] < 8)):
        return 'Loyal'
    elif ((df['RFMScore'] >= 6) and (df['RFMScore'] < 7)):
        return 'Potential'
    elif ((df['RFMScore'] >= 5) and (df['RFMScore'] < 6)):
        return 'Promising'
    elif ((df['RFMScore'] >= 4) and (df['RFMScore'] < 5)):
        return 'Needs Attention'
    else:
        return 'Require Activation'


# Creating a new variable RFMLevel
rfm_df['RFMLevel'] = rfm_df.apply(rfm_level, axis=1)
rfm_df
```

Out[230]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFMSegment	RFMScore	RFMLevel
12346.0	325	2	26.44	1	1	1	111	3	Require Activation
12347.0	1	191	2804.99	4	4	4	444	12	Can't Loose Them
12348.0	73	31	717.89	1	2	2	122	5	Promising
12349.0	17	73	820.87	2	3	3	233	8	Champions
12350.0	309	17	325.80	1	1	1	111	3	Require Activation
...	...	...	...	...	...	...	...	...	...
18280.0	277	10	52.05	1	1	1	111	3	Require Activation
18281.0	179	7	37.07	1	1	1	111	3	Require Activation
18282.0	4	13	878.61	3	1	3	313	7	Loyal
18283.0	2	726	10132.56	4	4	4	444	12	Can't Loose Them
18287.0	40	71	1311.37	1	3	3	133	7	Loyal

5838 rows × 9 columns

In [182]:
```
#!pip install squarify
```

In [183]:
```
# RFM Level Counts and Percentage
rfm_levels = pd.DataFrame(pd.concat([pd.DataFrame(rfm_df['RFMLevel'].value_counts()),pd.DataFrame(rfm_df['RFMLevel'].value_counts(normalize=True).mul(100).round(2))],axis=1))
rfm_levels.columns=['Count','Percent']
rfm_levels
```

Out[183]:

	Count	Percent
Can't Loose Them	2307	39.52
Champions	756	12.95
Require Activation	650	11.13
Loyal	625	10.71
Potential	592	10.14
Promising	464	7.95
Needs Attention	444	7.61

In [184]:
```
# Plotting RFM Levels
import matplotlib
import squarify

plt.figure(figsize=(15,8))

# Color Settings
norm = matplotlib.colors.Normalize(vmin=min(rfm_levels.Count), vmax=max(rfm_levels.Count))
colors = [matplotlib.cm.Blues(norm(value)) for value in rfm_levels.Count]

# Label Settings
perc = [str('{:5.2f}'.format(i/rfm_levels['Count'].sum()*100)) + "%" for i in rfm_levels['Count']]
lbl = [el[0] + " (" + el[1] + ")" for el in zip(rfm_levels.index, perc)]

# Square Plot
squarify.plot(sizes=rfm_levels.Count,label=lbl,color=colors)
plt.title("RFM Segments")
plt.axis('off')
plt.show()

# Observations:
# ~19% of the customers needs attention and require activation => Online store is doing well interms of retaining customers and
# driving sales from them
```

In [185]:
```
'''
Data Modeling :

1. Create clusters using k-means clustering algorithm.

a. Prepare the data for the algorithm. If the data is asymmetrically distributed, manage the skewness with appropriate transformation. Standardize the data.

b. Decide the optimum number of clusters to be formed.

c. Analyze these clusters and comment on the results.
'''
```

Out[185]:
```
'\nData Modeling :\n\n1. Create clusters using k-means clustering algorithm.\n\na. Prepare the data for the algorithm. If the data is asymmetrically distributed, manage the skewness with appropri
ate transformation. Standardize the data.\n\nb. Decide the optimum number of clusters to be formed.\n\nc. Analyze these clusters and comment on the results.\n'
```

In [186]:
```
# Grouping by customer ID
final_df = retail_train_df.groupby(['CustomerID']).agg({
            'InvoiceDate': lambda x: (ref_date - x.max()).days,
            'InvoiceNo': 'count',
            'Sales': 'sum',
            'Description':'count',
            'Quantity':'sum',
            'Country':'nunique'})

# Renaming columns
final_df.rename(columns={'InvoiceDate':'Recency','InvoiceNo':'Frequency','Description':'ItemCount','Country':'CountryCount'},inplace=True)
final_df
```

Out[186]:

CustomerID	Recency	Frequency	Sales	ItemCount	Quantity	CountryCount
12346.0	325	2	26.44	2	6	1
12347.0	1	191	2804.99	191	1140	6
12348.0	73	31	717.89	31	308	3
12349.0	17	73	820.87	73	167	1
12350.0	309	17	325.80	17	212	1
...	...	...	...	...	...	...
18280.0	277	10	52.05	10	12	1
18281.0	179	7	37.07	7	23	1
18282.0	4	13	878.61	13	875	1
18283.0	2	726	10132.56	726	5071	1
18287.0	40	71	1311.37	71	565	2

5838 rows × 6 columns

In [187]:
```
# Appending RFM Score to the final data
final_df['RFMScore'] = rfm_df['RFMScore']
final_df
```

Out[187]:

CustomerID	Recency	Frequency	Sales	ItemCount	Quantity	CountryCount	RFMScore
12346.0	325	2	26.44	2	6	1	3
12347.0	1	191	2804.99	191	1140	6	12
12348.0	73	31	717.89	31	308	3	5
12349.0	17	73	820.87	73	167	1	8
12350.0	309	17	325.80	17	212	1	3
...	...	...	...	...	...	...	...
18280.0	277	10	52.05	10	12	1	3
18281.0	179	7	37.07	7	23	1	3
18282.0	4	13	878.61	13	875	1	7
18283.0	2	726	10132.56	726	5071	1	12
18287.0	40	71	1311.37	71	565	2	7

5838 rows × 7 columns

In [188]:
```
# Standardizing Data
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()

# Fitting and transforming final data frame
final_scaled_df = mms.fit_transform(final_df)

# Converting array into dataframe
final_scaled_df = pd.DataFrame(final_scaled_df,columns=final_df.columns)
final_scaled_df.index = final_df.index
final_scaled_df
```

Out[188]:

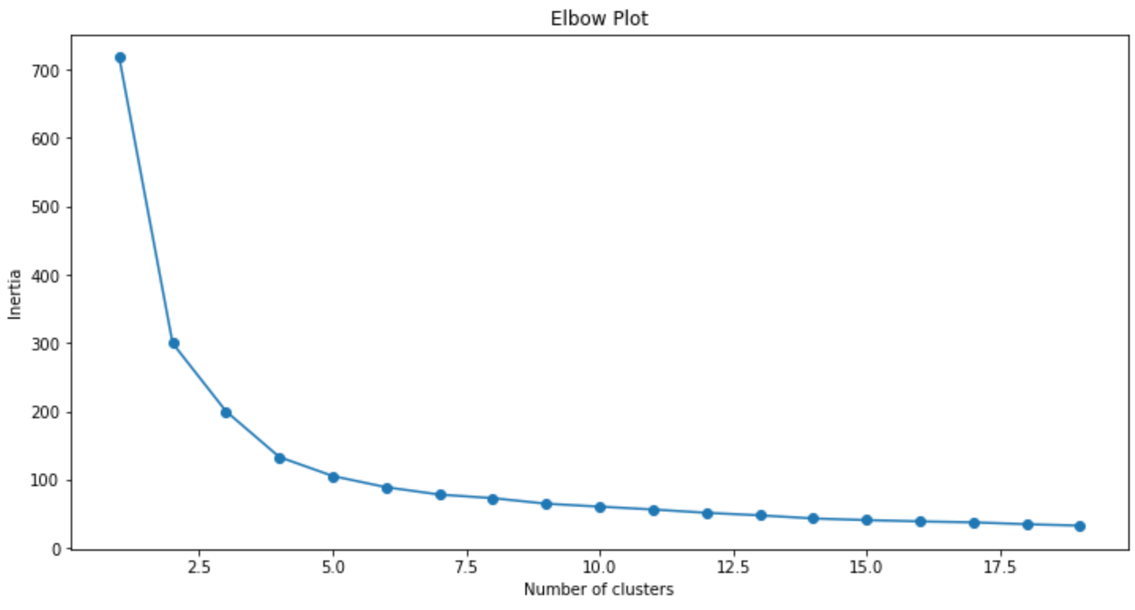
CustomerID	Recency	Frequency	Sales	ItemCount	Quantity	CountryCount	RFMScore
12346.0	0.870968	0.000127	0.329497	0.000127	0.477881	0.000000	0.000000
12347.0	0.000000	0.024216	0.341444	0.024216	0.485200	0.277778	1.000000
12348.0	0.193548	0.003824	0.332470	0.003824	0.479830	0.111111	0.222222
12349.0	0.043011	0.009177	0.332913	0.009177	0.478920	0.000000	0.555556
12350.0	0.827957	0.002039	0.330784	0.002039	0.479211	0.000000	0.000000
...	...	...	...	...	...	...	...
18280.0	0.741935	0.001147	0.329607	0.001147	0.477920	0.000000	0.000000
18281.0	0.478495	0.000765	0.329542	0.000765	0.477991	0.000000	0.000000
18282.0	0.008065	0.001529	0.333161	0.001529	0.483490	0.000000	0.444444
18283.0	0.002688	0.092404	0.372950	0.092404	0.510572	0.000000	1.000000
18287.0	0.104839	0.008922	0.335021	0.008922	0.481489	0.055556	0.444444

5838 rows × 7 columns

In [189]:
```
# KMeans Clustering
from sklearn.cluster import KMeans

# defining the kmeans function with initialization as k-means++
kmeans = KMeans(n_clusters=5, init='k-means++',random_state=10)

# fitting the k means algorithm on scaled data
kmeans.fit(final_scaled_df)

# inertia on the fitted data
kmeans.inertia_
```

Out[189]:
```
105.20214048684335
```

In [190]:
```
# Finding optimum  number of columns
# fitting multiple k-means algorithms and storing the values in an empty list
SSE = []
for cluster in range(1,20):
    kmeans = KMeans(n_clusters = cluster, init='k-means++',random_state=10)
    kmeans.fit(final_scaled_df)
    SSE.append(kmeans.inertia_)

# converting the results into a dataframe and plotting them
sse_df = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(sse_df['Cluster'], sse_df['SSE'], marker='o')
plt.title("Elbow Plot")
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

# Observations:
# Optimal number of clusters using elbow plot is '5'
```



In [218]:
```
# defining the kmeans function with initialization as k-means++
kmeans = KMeans(n_clusters=5,init='k-means++',random_state=10)

# fitting the k means algorithm on scaled data
y_pred = kmeans.fit_predict(final_scaled_df)

# inertia on the fitted data
kmeans.inertia_
```

Out[218]: 105.20214048684335

In [219]:
```
# Predicted Clusters
y_pred = pd.DataFrame(y_pred,columns=['Cluster'])
y_pred.index = final_df.index
y_pred
```

Out[219]:

CustomerID	Cluster
12346.0	1
12347.0	0
12348.0	2
12349.0	4
12350.0	1
...	...
18280.0	1
18281.0	1
18282.0	3
18283.0	0
18287.0	3

5838 rows × 1 columns

In [220]:
```
# Appending clusters to original dataframe
final_df['Cluster'] = y_pred['Cluster']
final_df
```

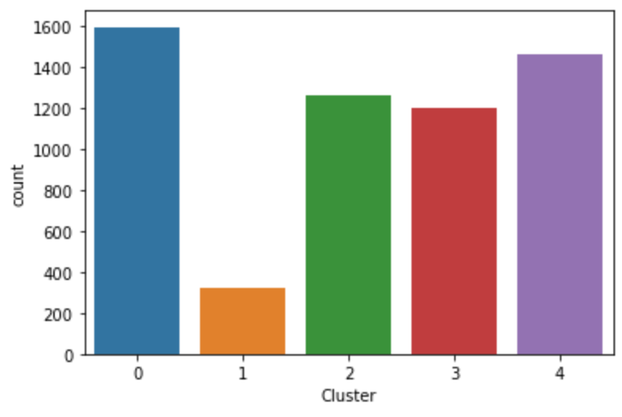
Out[220]:

CustomerID	Recency	Frequency	Sales	ItemCount	Quantity	CountryCount	RFMScore	Cluster
12346.0	325	2	26.44	2	6	1	3	1
12347.0	1	191	2804.99	191	1140	6	12	0
12348.0	73	31	717.89	31	308	3	5	2
12349.0	17	73	820.87	73	167	1	8	4
12350.0	309	17	325.80	17	212	1	3	1
...	...	...	...	...	...	...	...	...
18280.0	277	10	52.05	10	12	1	3	1
18281.0	179	7	37.07	7	23	1	3	1
18282.0	4	13	878.61	13	875	1	7	3
18283.0	2	726	10132.56	726	5071	1	12	0
18287.0	40	71	1311.37	71	565	2	7	3

5838 rows × 8 columns

In [221]:
```
# Count of customers in each of the clusters
print(final_df['Cluster'].value_counts())
sns.countplot(final_df['Cluster'])
plt.show()

# Observations:
# Cluster 2 has maximum number of customers(2324) where as cluster 3 has the minimum(130)
```

```
0    1596
4    1465
2    1260
3    1198
1     319
Name: Cluster, dtype: int64
```



In [222]:
```
# Visualizing Clusters
plt.figure(figsize=(10,6))
sns.scatterplot(final_df['Cluster'],final_df['RFMScore'],hue=final_df['Cluster'])
plt.show()

# Observations:
# Cluster 1- RFM Score > 10 --Customers you can't afford to loose (High Value Customers)
# Cluster 2- RFM Score <= 8 --Customers who need attention(Potential,Promising, Needs attention and require Activation)
# Cluster 3- RFM Score < 5 --Customers who require some reactivation campaigns(Require Activation)
# Clsuter 4- RFM Score = 6 and 7 --Customers who require some attention interms of personalized offers(Needs attention)
# Clsuter 5- RFM Score = 8 and 9 --Loyal customers with potential to become high value customers(Champions,Loyal)
# Cluster 2 is a combination of cluster 3 and 4
```