Customer Segmentation

Customer Segmentation enables orgaizations to:

- Better understand their customers
- Tailor their product offerings
- Optimize marketing and growth strategies

Dataset can be found in the UCI Machine Learning repository [here (https://archive.ics.uci.edu/ml/datasets/online+retail)](https://archive.ics.uci.edu/ml/datasets/online+retail)

Dataset description

The tabluar dataset consists details of purchases made by the customers on an online retail platform.

Dataset features:

- InvoiceNo: The invoice number of each purchase made on the platform
- StockCode: The SKU of each product on the platform
- Description: The small description of the product
- Quantity: Quantity purchased by the customer
- InvoiceDate: The date of purchase
- UnitPrice: Price of each unit of that product
- CustomerID: The unique ID assigned to each customer
- Country: The location of each customer

Index

1. Importing libraries and dataset
2. Exploratory Data Analysis and Data Cleaning
3. Preparing data for analysis
4. Customer segmentation using RFM technique
5. Data visualization
6. Insignts

---

1. Importing libraries and dataset

In [1]:
```
# importing the required libraries
import pandas as pd
import numpy as np
import warnings
import datetime as dt
import matplotlib.pyplot as plt
```

In [2]:
```
# settings for displaying complete results
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
np.set_printoptions(threshold=np.inf)
warnings.filterwarnings('ignore')
```

In [3]:
```
# creating the dataframe with the online_retail.xlsx file
df = pd.read_excel("online_retail.xlsx")
```

In [4]:
```
# displaying the header and top 5 rows of the dataframe
df.head()
```

Out[4]:

	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

2. Exploratory Data Analysis

Performing exploratory data analysis on the dataset and trying to find some insights

In [5]:
```
# finding the unique countries in the dataset
df.Country.unique()
```

Out[5]:
```
array(['United Kingdom', 'France', 'Australia', 'Netherlands', 'Germany',
       'Norway', 'EIRE', 'Switzerland', 'Spain', 'Poland', 'Portugal',
       'Italy', 'Belgium', 'Lithuania', 'Japan', 'Iceland',
       'Channel Islands', 'Denmark', 'Cyprus', 'Sweden', 'Austria',
       'Israel', 'Finland', 'Bahrain', 'Greece', 'Hong Kong', 'Singapore',
       'Lebanon', 'United Arab Emirates', 'Saudi Arabia',
       'Czech Republic', 'Canada', 'Unspecified', 'Brazil', 'USA',
       'European Community', 'Malta', 'RSA'], dtype=object)
```

In [6]:

```
# finding number of customers from each country
country_customers = df[['Country', 'CustomerID']].drop_duplicates()
country_customers.groupby(['Country'])['CustomerID'].aggregate('count').reset_index(
```

Out[6]:

	Country	CustomerID
36	United Kingdom	3950
14	Germany	95
13	France	87
31	Spain	31
3	Belgium	25
33	Switzerland	21
27	Portugal	19
19	Italy	15
12	Finland	12
1	Austria	11
25	Norway	10
24	Netherlands	9
0	Australia	9
6	Channel Islands	9
9	Denmark	9
7	Cyprus	8
32	Sweden	8
20	Japan	8
26	Poland	6
34	USA	4
5	Canada	4
37	Unspecified	4
18	Israel	4
15	Greece	4
10	EIRE	3
23	Malta	2
35	United Arab Emirates	2
2	Bahrain	2
22	Lithuania	1
8	Czech Republic	1
21	Lebanon	1

	Country	CustomerID
28	RSA	1
29	Saudi Arabia	1
30	Singapore	1
17	Iceland	1
4	Brazil	1
11	European Community	1
16	Hong Kong	0

The result shows that most of the customers are from UK. Therefore, we can limit our analysis to UK.

In [7]:
```
# creating a new dataframe which will contain details related to UK only
df_uk = df.loc[df['Country'] == 'United Kingdom']
```

In [8]:
```
# checking for NULL values in the UK dataframe
df_uk.isnull().sum()
```

Out[8]:
```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     133600
Country             0
dtype: int64
```

The above result shows that there are 1454 null values in description and 133600 null values in CustomerID. However, we do not require these columns for our analysis. Therefore, we can wither drop these columns or ignore them.

In [9]:
```
# proceeding with dropping the columns that has null value
columns_to_remove = ['Description']
df_uk = df_uk.drop(columns = columns_to_remove)
```

In [10]:
```python
# displaying the top 5 columns in the new UK dataframe
df_uk.head()
```

Out[10]:

	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

In [11]:
```python
# checking if the Quantity column in the UK dataframe has negative values
has_negative = (df_uk['Quantity'] < 0).sum()

# printing the total negative values in the column
print("Total negative values in Quantity column:", has_negative)
```

```
Total negative values in Quantity column: 9192
```

In [12]:
```python
# removing the rows that have Quantity < 0
df_uk = df_uk[(df_uk['Quantity'] >= 0)]
```

In [13]:
```python
# displaying the top 5 rows of the new df_uk dataframe
df_uk.head()
```

Out[13]:

	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

In [14]:
```python
# creating a new column for 'TotalPrice' by multiplying 'Quantity' with 'UnitPrice'
df_uk['TotalPrice'] = df['Quantity'] * df['UnitPrice']
```

In [15]:
```
# viewing the new TotalPrice column
df_uk.head()
```

Out[15]:

	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	TotalPrice
0	536365	85123A	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	15.30
1	536365	71053	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34
2	536365	84406B	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	22.00
3	536365	84029G	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34
4	536365	84029E	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34

In [16]:
```
# finding the earliest invoice date
df_uk['InvoiceDate'].min()
```

Out[16]:
```
Timestamp('2010-12-01 08:26:00')
```

In [17]:
```
# finding the latest invoice date
df_uk['InvoiceDate'].max()
```

Out[17]:
```
Timestamp('2011-12-09 12:49:00')
```

In [18]:
```
# calculating the recency
most_recent = dt.datetime(2011,12,10)
df_uk['InvoiceDate'] = pd.to_datetime(df_uk['InvoiceDate'])
```

3. Data Preparation

We will use the RFM customer segmentation technique for finding out the best customers.

The R, F and M in RFM stands for Recency, Frequency and Monetary.

- **R**ecency: Number of days since last purchase.
- **F**requency: The total number of purchases.
- **M**onetary: The total amount of money spent.

Creating a RFM table for analysis and segmenting customers.

In [19]:
```
# creating the RFM table
table = df_uk.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (most_recent - x.max()).days,
    'InvoiceNo': lambda x: len(x),
    'TotalPrice': lambda x: x.sum()
})

# type-casting the InvoiceDate column to integer
table['InvoiceDate'] = table['InvoiceDate'].astype(int)

# renaming the columns for better understanding
table.rename(columns = {
    'InvoiceDate': 'Recency',
    'InvoiceNo': 'Frequency',
    'TotalPrice': 'Monetary_value'
}, inplace = True)
```

In [20]:
```python
# displaying the RFM table
table.head(10)
```

Out[20]:

	Recency	Frequency	Monetary_value
CustomerID			
12346.0	325	1	77183.60
12747.0	2	103	4196.01
12748.0	0	4596	33719.73
12749.0	3	199	4090.88
12820.0	3	59	942.34
12821.0	214	6	92.72
12822.0	70	46	948.88
12823.0	74	5	1759.50
12824.0	59	25	397.12
12826.0	2	91	1474.72

4. Customer Segmentation using RFM technique

In [21]:
```python
# creating segments using quartiles
segments = table.quantile(q = [0.25, 0.5, 0.75])
segments = segments.to_dict()
```

In [22]:
```python
# creating a RFM table for customers
customer_table = table
```

In [23]:
```python
# assigning a quartile to Recency values for splitting into segments (1 - good to 4
def R_value_score(x, p, d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.5]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4
```

In [24]:
```python
# assigning a quartile to Recency, Frequency and Monetary values for splitting into
def FM_value_score(x, p, d):
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.5]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1
```

In [25]:
```python
# assigning the segment numbers to Recency
customer_table['R_quartile'] = customer_table['Recency'].apply(R_value_score, args =

# assigning the segment numbers to Frequency
customer_table['F_quartile'] = customer_table['Frequency'].apply(FM_value_score, arg

# assigning the segment numbers to Monetary value
customer_table['M_quartile'] = customer_table['Monetary_value'].apply(FM_value_score
```

In [26]:
```python
# displaying the Recency, Frequency and Monetary quartiles for each customer
customer_table.head(10)
```

Out[26]:

	Recency	Frequency	Monetary_value	R_quartile	F_quartile	M_quartile
CustomerID						
12346.0	325	1	77183.60	4	4	1
12747.0	2	103	4196.01	1	1	1
12748.0	0	4596	33719.73	1	1	1
12749.0	3	199	4090.88	1	1	1
12820.0	3	59	942.34	1	2	2
12821.0	214	6	92.72	4	4	4
12822.0	70	46	948.88	3	2	2
12823.0	74	5	1759.50	3	4	1
12824.0	59	25	397.12	3	3	3
12826.0	2	91	1474.72	1	2	2

In [27]:

```
# assigning a score by combining the values of R_quartile, F_quartile, M_quartile
customer_table['RFM_score'] = customer_table.R_quartile.map(str) + customer_table.F_
```

In [28]:

```
# displaying the RFM Scores for each customer
customer_table.head(10)
```

Out[28]:

CustomerID	Recency	Frequency	Monetary_value	R_quartile	F_quartile	M_quartile	RFM_score
12346.0	325	1	77183.60	4	4	1	441
12747.0	2	103	4196.01	1	1	1	111
12748.0	0	4596	33719.73	1	1	1	111
12749.0	3	199	4090.88	1	1	1	111
12820.0	3	59	942.34	1	2	2	122
12821.0	214	6	92.72	4	4	4	444
12822.0	70	46	948.88	3	2	2	322
12823.0	74	5	1759.50	3	4	1	341
12824.0	59	25	397.12	3	3	3	333
12826.0	2	91	1474.72	1	2	2	122

In [29]:
```
# displaying the top 25 customers (RFM_score=111)
customer_table[customer_table['RFM_score'] == '111'].sort_values('Monetary_value', a
```

Out[29]:

	Recency	Frequency	Monetary_value	R_quartile	F_quartile	M_quartile	RFM_score
CustomerID							
18102.0	0	431	259657.30	1	1	1	111
17450.0	8	337	194550.79	1	1	1	111
17511.0	2	963	91062.38	1	1	1	111
16684.0	4	277	66653.56	1	1	1	111
14096.0	4	5111	65164.79	1	1	1	111
13694.0	3	568	65039.62	1	1	1	111
15311.0	0	2379	60767.90	1	1	1	111
13089.0	2	1818	58825.83	1	1	1	111
15769.0	7	130	56252.72	1	1	1	111
15061.0	3	403	54534.14	1	1	1	111
14298.0	8	1637	51527.30	1	1	1	111
14088.0	10	589	50491.81	1	1	1	111
17841.0	1	7847	40991.57	1	1	1	111
13798.0	1	349	37153.85	1	1	1	111
16013.0	3	139	37130.60	1	1	1	111
16422.0	17	369	34684.40	1	1	1	111
12748.0	0	4596	33719.73	1	1	1	111
15838.0	11	167	33643.08	1	1	1	111
17389.0	0	213	31833.68	1	1	1	111
13098.0	1	572	28882.44	1	1	1	111
13081.0	11	1028	28337.38	1	1	1	111
13408.0	1	478	28117.04	1	1	1	111
13777.0	0	197	25977.16	1	1	1	111
16210.0	1	123	21086.30	1	1	1	111
17675.0	1	705	20374.28	1	1	1	111

In [30]:
```
# displaying the top 25 customers by recency
customer_table.sort_values('Recency', ascending=False).head(25)
```

Out[30]:

CustomerID	Recency	Frequency	Monetary_value	R_quartile	F_quartile	M_quartile	RFM_score
17643.0	373	8	101.55	4	4	4	444
15165.0	373	27	487.75	4	3	3	433
13747.0	373	1	79.60	4	4	4	444
17908.0	373	58	243.28	4	2	4	424
17968.0	373	85	277.35	4	2	4	424
16048.0	373	8	256.44	4	4	4	444
15922.0	373	11	369.50	4	4	3	443
16583.0	373	14	233.45	4	4	4	444
18011.0	373	28	102.79	4	3	4	434
13065.0	373	14	205.86	4	4	4	444
18074.0	373	13	489.60	4	4	3	443
14142.0	373	22	311.81	4	3	3	433
15350.0	373	5	115.65	4	4	4	444
14237.0	373	9	161.00	4	4	4	444
16274.0	373	67	357.95	4	2	3	423
14729.0	373	71	313.49	4	2	3	423
17925.0	372	1	244.08	4	4	4	444
17855.0	372	17	208.97	4	4	4	444
16754.0	372	2	2002.40	4	4	1	441
12855.0	372	3	38.10	4	4	4	444
16752.0	372	9	207.50	4	4	4	444
17732.0	372	18	303.97	4	3	3	433
15823.0	372	1	15.00	4	4	4	444
15070.0	372	1	106.20	4	4	4	444
13011.0	372	3	50.55	4	4	4	444

In [31]:
```
# displaying the top 25 customers by frequency
customer_table.sort_values('Frequency', ascending=False).head(25)
```

Out[31]:

CustomerID	Recency	Frequency	Monetary_value	R_quartile	F_quartile	M_quartile	RFM_score
17841.0	1	7847	40991.57	1	1	1	111
14096.0	4	5111	65164.79	1	1	1	111
12748.0	0	4596	33719.73	1	1	1	111
14606.0	1	2700	12156.65	1	1	1	111
15311.0	0	2379	60767.90	1	1	1	111
13089.0	2	1818	58825.83	1	1	1	111
13263.0	1	1677	7454.07	1	1	1	111
14298.0	8	1637	51527.30	1	1	1	111
15039.0	9	1502	19914.44	1	1	1	111
18118.0	10	1279	5653.82	1	1	1	111
14159.0	19	1204	4693.01	2	1	1	211
14796.0	1	1141	8022.49	1	1	1	111
16033.0	5	1137	8816.40	1	1	1	111
15005.0	15	1119	6316.57	1	1	1	111
14056.0	1	1106	8214.65	1	1	1	111
14769.0	2	1090	10674.75	1	1	1	111
13081.0	11	1028	28337.38	1	1	1	111
16549.0	10	981	4154.64	1	1	1	111
14527.0	2	972	8508.82	1	1	1	111
14456.0	5	970	3062.40	1	1	1	111
17511.0	2	963	91062.38	1	1	1	111
15719.0	32	937	5045.61	2	1	1	211
15555.0	12	899	4805.17	1	1	1	111
16931.0	5	898	4604.22	1	1	1	111
17811.0	4	851	7837.73	1	1	1	111

In [32]:
```
# displaying the top 25 customers by monetary value
customer_table.sort_values('Monetary_value', ascending=False).head(25)
```

Out[32]:

CustomerID	Recency	Frequency	Monetary_value	R_quartile	F_quartile	M_quartile	RFM_score
18102.0	0	431	259657.30	1	1	1	111
17450.0	8	337	194550.79	1	1	1	111
16446.0	0	3	168472.50	1	4	1	141
17511.0	2	963	91062.38	1	1	1	111
16029.0	38	242	81024.84	2	1	1	211
12346.0	325	1	77183.60	4	4	1	441
16684.0	4	277	66653.56	1	1	1	111
14096.0	4	5111	65164.79	1	1	1	111
13694.0	3	568	65039.62	1	1	1	111
15311.0	0	2379	60767.90	1	1	1	111
13089.0	2	1818	58825.83	1	1	1	111
17949.0	1	70	58510.48	1	2	1	121
15769.0	7	130	56252.72	1	1	1	111
15061.0	3	403	54534.14	1	1	1	111
14298.0	8	1637	51527.30	1	1	1	111
14088.0	10	589	50491.81	1	1	1	111
15749.0	235	10	44534.30	4	4	1	441
12931.0	21	82	42055.96	2	2	1	221
17841.0	1	7847	40991.57	1	1	1	111
15098.0	182	3	39916.50	4	4	1	441
13798.0	1	349	37153.85	1	1	1	111
16013.0	3	139	37130.60	1	1	1	111
16422.0	17	369	34684.40	1	1	1	111
12748.0	0	4596	33719.73	1	1	1	111
15838.0	11	167	33643.08	1	1	1	111

In [33]:
```
# counting unique RFM scores in the customer table
unique_rfm = customer_table['RFM_score'].nunique()
print(f"Unique RFM scores in the customer table:", unique_rfm)
```

```
Unique RFM scores in the customer table: 61
```

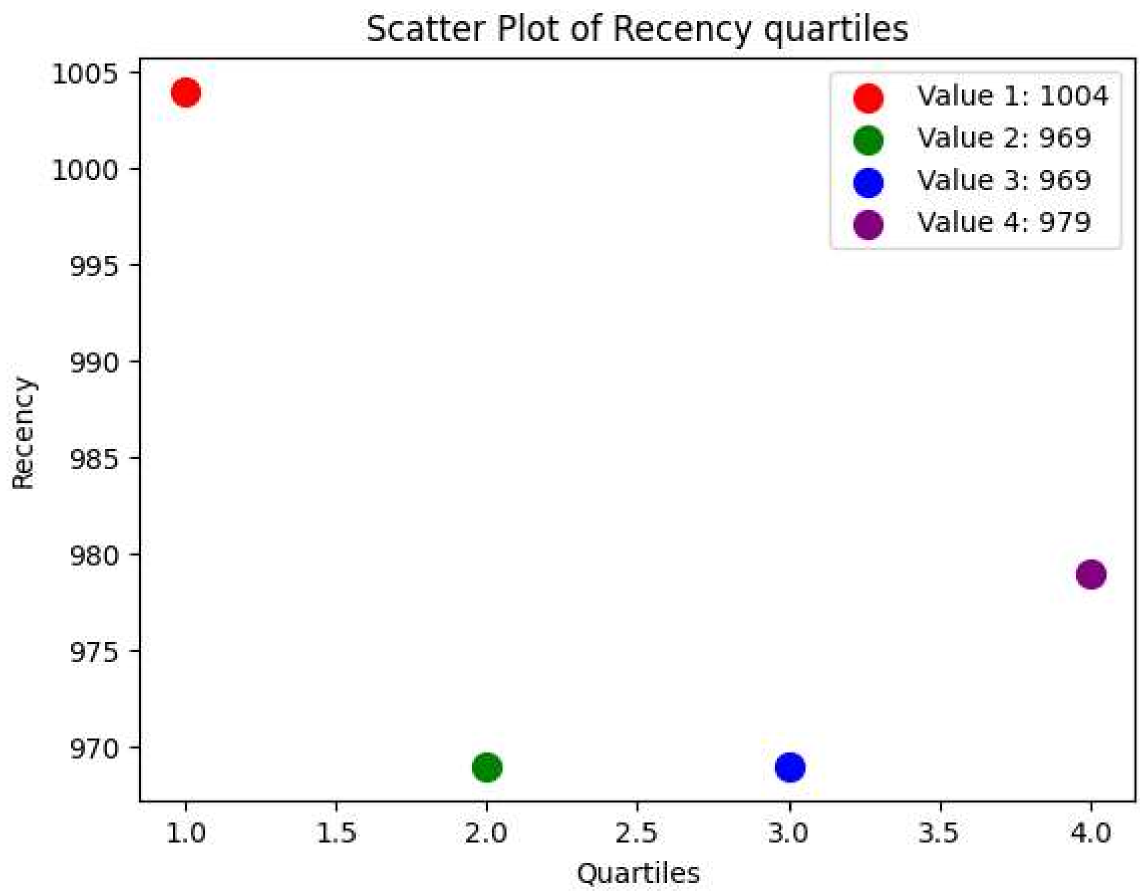
In [34]:
```
# count of each unique RFM score in the customer table
rfm_count = customer_table['RFM_score'].value_counts()
print(f'score count')
print(rfm_count)
```

score	count
111	409
444	343
211	186
433	180
344	168
222	156
322	142
333	141
122	124
244	112
233	107
443	104
311	92
434	90
212	83
112	81
343	79
121	78
332	77
133	66
223	63
144	59
422	58
221	53
234	52
321	51
232	49
423	49
323	48
334	42
132	40
123	39
312	39
432	35
424	34
243	33
342	27
224	25
442	23
134	21
143	21
324	21
411	20
412	19
213	18
131	18
331	16
113	16
242	14

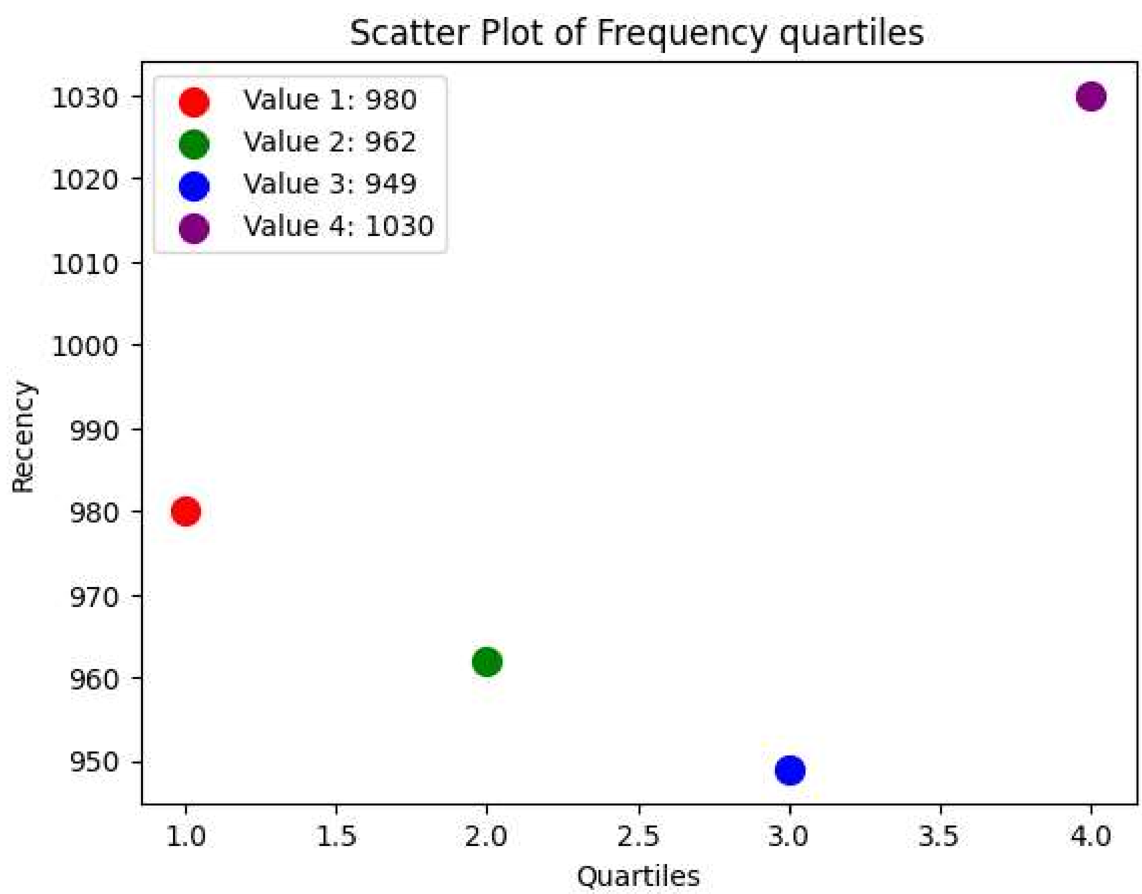
```
341    14
142    13
124    13
313    12
231    11
441     8
421     8
241     7
141     5
431     4
413     4
114     1
Name: RFM_score, dtype: int64
```

5. Data visualization

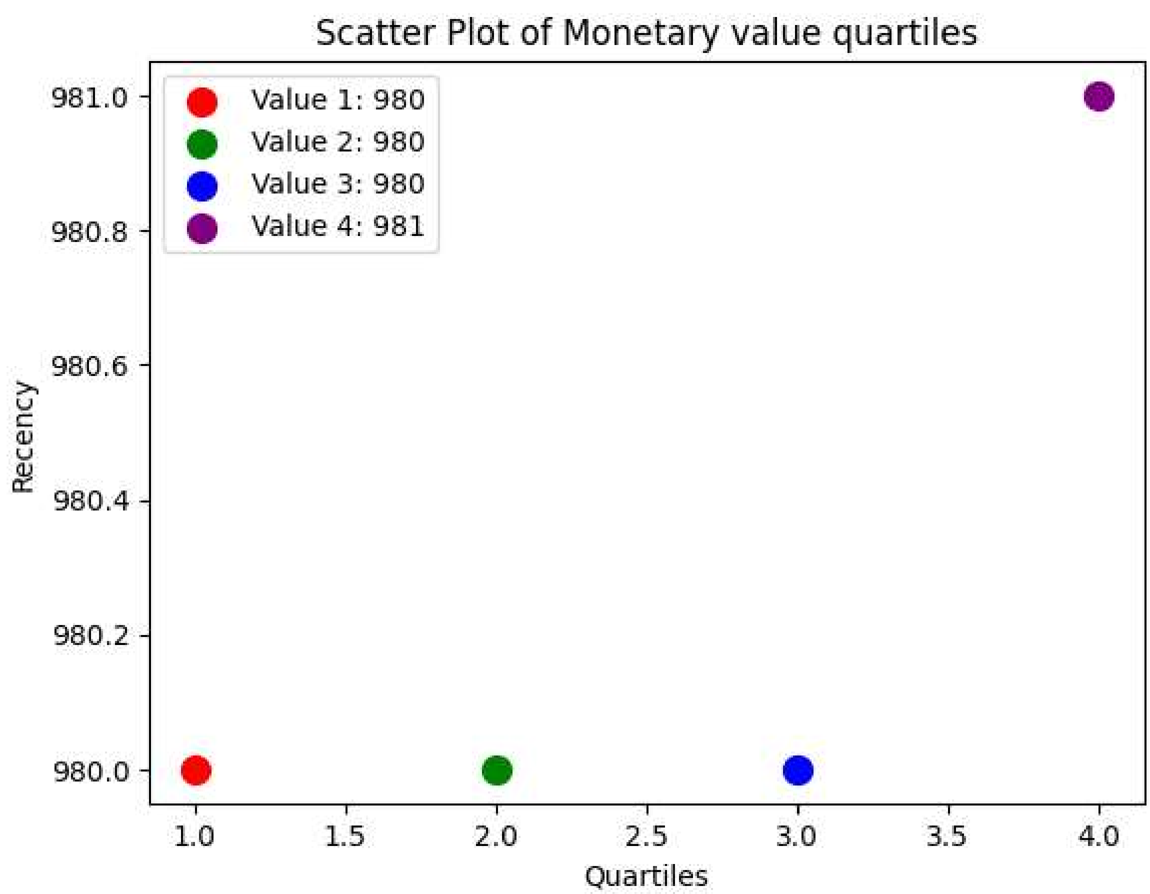
In [35]:
```
# plotting a scatterplot for the Recency quartile

# counting the total number of Recency in each Recency quartile
value_counts = customer_table['R_quartile'].value_counts().sort_index()

# generate colors for each value
colors = ['red', 'green', 'blue', 'purple']

# plotting X and Y labels
plt.xlabel('Quartiles')
plt.ylabel('Recency')
plt.title('Scatter Plot of Recency quartiles')

# create scatter plots for each value
for i, (value, count) in enumerate(value_counts.items()):
    plt.scatter(value, count, c = colors[i], s = 100, label = f'Value {value}: {cour

# show legend
plt.legend()

# show the plot
plt.show()
```


Scatter Plot of Recency quartiles
Value 1: 1004
Value 2: 969
Value 3: 969
Value 4: 979
1005
1000
995
990
985
980
975
970
Recency
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Quartiles

In [36]:
```
# plotting a scatterplot for the Frequency quartile
value_counts = customer_table['F_quartile'].value_counts().sort_index()

# generate colors for each value
colors = ['red', 'green', 'blue', 'purple']

# plotting X and Y labels
plt.xlabel('Quartiles')
plt.ylabel('Recency')
plt.title('Scatter Plot of Frequency quartiles')

# create scatter plots for each value
for i, (value, count) in enumerate(value_counts.items()):
    plt.scatter(value, count, c = colors[i], s = 100, label = f'Value {value}: {cour

# show legend
plt.legend()

# show the plot
plt.show()
```



In [37]:
```
# plotting a scatterplot for the Monetary_value quartile
value_counts = customer_table['M_quartile'].value_counts().sort_index()

# generate colors for each value
colors = ['red', 'green', 'blue', 'purple']

# plotting X and Y labels
plt.xlabel('Quartiles')
plt.ylabel('Recency')
plt.title('Scatter Plot of Monetary value quartiles')

# create scatter plots for each value
for i, (value, count) in enumerate(value_counts.items()):
    plt.scatter(value, count, c = colors[i], s = 100, label = f'Value {value}: {cour

# show legend
plt.legend()

# show the plot
plt.show()
```



Insights

1. The highest number of customers are from UK.
2. The highest value of Recency is 373 days.
3. The highest value of Frequency is 7847.
4. The highest value of Monetary value is 259657.30
5. Total unique RFM scores are 61.
6. The top 10 RFM scores and their counts are

RFM Score	Count
111	409
444	343
211	186
433	180
344	168
222	156
322	142
333	141
122	124
244	112

In [ ]: