RFM Analysis in Python

RFM analysis is scoring our customers based on their Recency, Frequency and Monetary values. - Recency: How recently a customer made a purchase. - Frequency: How often customers make a purchase. - Monetary Value: How much money a customer spends on purchases.

In [38]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
import datetime as dt
import warnings
warnings.filterwarnings('ignore')
```

Loading data

In [39]:
```
data=pd.read_csv('sales_data.csv')
data
```

Out[39]:

	OrderNumber	Sales Channel	WarehouseCode	ProcuredDate	OrderDate	ShipDate	DeliveryDate	CurrencyCode	_SalesTeamID
0	SO - 000101	In-Store	WARE-UHY1004	12/31/2017	5/31/2018	6/14/2018	6/19/2018	USD	6
1	SO - 000102	Online	WARE-NMK1003	12/31/2017	5/31/2018	6/22/2018	7/2/2018	USD	14
2	SO - 000103	Distributor	WARE-UHY1004	12/31/2017	5/31/2018	6/21/2018	7/1/2018	USD	21
3	SO - 000104	Wholesale	WARE-NMK1003	12/31/2017	5/31/2018	6/2/2018	6/7/2018	USD	28
4	SO - 000105	Distributor	WARE-NMK1003	4/10/2018	5/31/2018	6/16/2018	6/26/2018	USD	22
...	...	...	...	...	...	...	...	...	...
7986	SO - 0008087	In-Store	WARE-MKL1006	9/26/2020	12/30/2020	1/7/2021	1/14/2021	USD	9
7987	SO - 0008088	Online	WARE-NMK1003	9/26/2020	12/30/2020	1/2/2021	1/4/2021	USD	14
7988	SO - 0008089	Online	WARE-UHY1004	9/26/2020	12/30/2020	1/23/2021	1/26/2021	USD	14
7989	SO - 0008090	Online	WARE-NMK1003	9/26/2020	12/30/2020	1/20/2021	1/25/2021	USD	20
7990	SO - 0008091	In-Store	WARE-UHY1004	9/26/2020	12/30/2020	1/13/2021	1/19/2021	USD	6

7991 rows × 16 columns

Understanding Data

In [36]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7991 entries, 0 to 7990
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   OrderNumber       7991 non-null   object 
 1   Sales Channel     7991 non-null   object 
 2   WarehouseCode     7991 non-null   object 
 3   ProcuredDate      7991 non-null   object 
 4   OrderDate         7991 non-null   object 
 5   ShipDate          7991 non-null   object 
 6   DeliveryDate      7991 non-null   object 
 7   CurrencyCode      7991 non-null   object 
 8   _SalesTeamID      7991 non-null   int64  
 9   _CustomerID       7991 non-null   int64  
 10  _StoreID          7991 non-null   int64  
 11  _ProductID        7991 non-null   int64  
 12  Order Quantity    7991 non-null   int64  
 13  Discount Applied  7991 non-null   float64
 14  Unit Price        7991 non-null   float64
 15  Unit Cost         7991 non-null   float64
dtypes: float64(3), int64(5), object(8)
memory usage: 999.0+ KB
```

In [40]:
```
data.shape
```

Out[40]:
```
(7991, 16)
```

In [41]:
```
data.columns
```

Out[41]:
```
Index(['OrderNumber', 'Sales Channel', 'WarehouseCode', 'ProcuredDate',
       'OrderDate', 'ShipDate', 'DeliveryDate', 'CurrencyCode', '_SalesTeamID',
       '_CustomerID', '_StoreID', '_ProductID', 'Order Quantity',
       'Discount Applied', 'Unit Price', 'Unit Cost'],
      dtype='object')
```

In [42]:
```
data.describe()
```

Out[42]:

	_SalesTeamID	_CustomerID	_StoreID	_ProductID	Order Quantity	Discount Applied	Unit Price	Unit Cost
count	7991.000000	7991.000000	7991.000000	7991.000000	7991.000000	7991.000000	7991.000000	7991.000000
mean	14.384307	25.457014	183.850081	23.771743	4.525341	0.114394	2284.536504	1431.911054
std	7.986086	14.414883	105.903946	13.526545	2.312631	0.085570	1673.096364	1112.413043
min	1.000000	1.000000	1.000000	1.000000	1.000000	0.050000	167.500000	68.675000
25%	8.000000	13.000000	91.000000	12.000000	3.000000	0.050000	1031.800000	606.115500
50%	14.000000	25.000000	183.000000	24.000000	5.000000	0.075000	1849.200000	1080.576000
75%	21.000000	38.000000	276.000000	36.000000	7.000000	0.150000	3611.300000	2040.250500
max	28.000000	50.000000	367.000000	47.000000	8.000000	0.400000	6566.000000	5498.556000

Data preparation

In [43]:
```
#checking null values:
data.isnull().sum()
```

Out[43]:
```
OrderNumber         0
Sales Channel       0
WarehouseCode       0
ProcuredDate        0
OrderDate           0
ShipDate            0
DeliveryDate        0
CurrencyCode        0
_SalesTeamID        0
_CustomerID         0
_StoreID            0
_ProductID          0
Order Quantity      0
Discount Applied    0
Unit Price          0
Unit Cost           0
dtype: int64
```

In [44]:
```
#Dropping Duplicated Values
data = data.drop_duplicates()
```

In [45]:
```
#how many of each product?
data["WarehouseCode"].value_counts().head()
```

Out[45]:
```
WARE-NMK1003    2505
WARE-PUJ1005    1451
WARE-UHY1004    1265
WARE-XYS1001    1222
WARE-MKL1006     857
Name: WarehouseCode, dtype: int64
```

In [46]:
```
#What are the most expensive products?
data.sort_values("Unit Price", ascending = False).head()
```

Out[46]:

	OrderNumber	Sales Channel	WarehouseCode	ProcuredDate	OrderDate	ShipDate	DeliveryDate	CurrencyCode	_SalesTeamID
1776	SO - 0001877	Distributor	WARE-PUJ1005	10/27/2018	12/26/2018	1/3/2019	1/13/2019	USD	23
3963	SO - 0004064	In-Store	WARE-XYS1001	5/15/2019	9/12/2019	9/23/2019	9/29/2019	USD	11
4350	SO - 0004451	Wholesale	WARE-NMK1003	8/23/2019	10/31/2019	11/10/2019	11/17/2019	USD	28
3603	SO - 0003704	Online	WARE-UHY1004	2/4/2019	7/29/2019	8/18/2019	8/27/2019	USD	15
5265	SO - 0005366	Distributor	WARE-NMK1003	12/1/2019	2/16/2020	2/26/2020	2/28/2020	USD	25

In [47]:
```
#Creating ‘Total Price’ Column
data['Total_Price'] = data['Unit Price']*data['Order Quantity']
data['Total_Price']
```

```
Out[47]: 0        9815.5
1       11818.8
2        1775.5
3       18599.2
4       14579.2
          ...
7986      234.5
7987    19215.6
7988    19128.5
7989     8576.0
7990    11055.0
Name: Total_Price, Length: 7991, dtype: float64
```

Creating RFM Dataframe

```
In [48]: #converting'OrderDate' column to datetime format
data['OrderDate'] = pd.to_datetime(data['OrderDate'])
```

```
In [49]: #last purchase date:
data['OrderDate'].max()
```

```
Out[49]: Timestamp('2020-12-30 00:00:00')
```

```
In [50]: #date for analysis :
today =  dt.datetime(2020,12,30)
```

```
In [51]: today
```

```
Out[51]: datetime.datetime(2020, 12, 30, 0, 0)
```

```
In [54]: rfm_data = data.groupby('_CustomerID').agg({
    'OrderDate': lambda x: (today - x.max()).days, #For Recency, Calculate the number of days between present da
    '_CustomerID': 'count',#For Frequency, Calculate the number of orders for each customer.
    'Total_Price': 'sum' #For Monetary, Calculate sum of purchase price for each customer.
})

# Rename columns for clarity
rfm_data.rename(columns={
    'OrderDate': 'Recency',
    '_CustomerID': 'Frequency',
    'Total_Price': 'Monetary'
}, inplace=True)
```

```
In [56]: rfm_data.head()
```

Out[56]:

_CustomerID	Recency	Frequency	Monetary
1	7	152	1322278.5
2	7	135	1346264.5
3	8	181	1831947.5
4	3	167	1770582.2
5	28	159	1609232.8

RFM Quartiles:

Dividing the RFM values into quartiles to categorize customers.

```
In [59]: # Calculate quartiles for Recency, Frequency, and Monetary
quartiles = rfm_data.quantile(q=[0.25, 0.5, 0.75])

# Create functions to assign R, F, and M scores based on quartiles
def assign_fm_score(x, quartiles):
    if x <= quartiles[0.25]:
        return 1
    elif x <= quartiles[0.50]:
        return 2
    elif x <= quartiles[0.75]:
        return 3
    else:
        return 4

# Assign scores to the RFM values
rfm_data['R'] = rfm_data['Recency'].apply(assign_r_score, args=(quartiles,))
rfm_data['F'] = rfm_data['Frequency'].apply(assign_fm_score, args=(quartiles['Frequency'],))
```

```
rfm_data['M'] = rfm_data['Monetary'].apply(assign_fm_score, args=(quartiles['Monetary'],))
```

RFM Score Calculation:

Combine the R, F, and M scores to calculate the RFM score for each customer.

In [60]:
```
# Calculate the RFM score by combining R, F, and M
rfm_data['RFM_Score'] = rfm_data['R'] * 100 + rfm_data['F'] * 10 + rfm_data['M']
```

In [66]:
```
rfm_data['RFM_Score']
```

Out[66]:
```
_CustomerID
1     221
2     211
3     144
4     333
5     132
6     412
7     422
8     311
9     244
10    133
11    244
12    444
13    444
14    321
15    411
16    412
17    244
18    244
19    433
20    232
21    434
22    411
23    133
24    111
25    333
26    122
27    411
28    211
29    444
30    433
31    421
32    444
33    424
34    144
35    112
36    422
37    123
38    211
39    444
40    213
41    432
42    433
43    312
44    422
45    323
46    421
47    242
48    143
49    221
50    133
Name: RFM_Score, dtype: int64
```

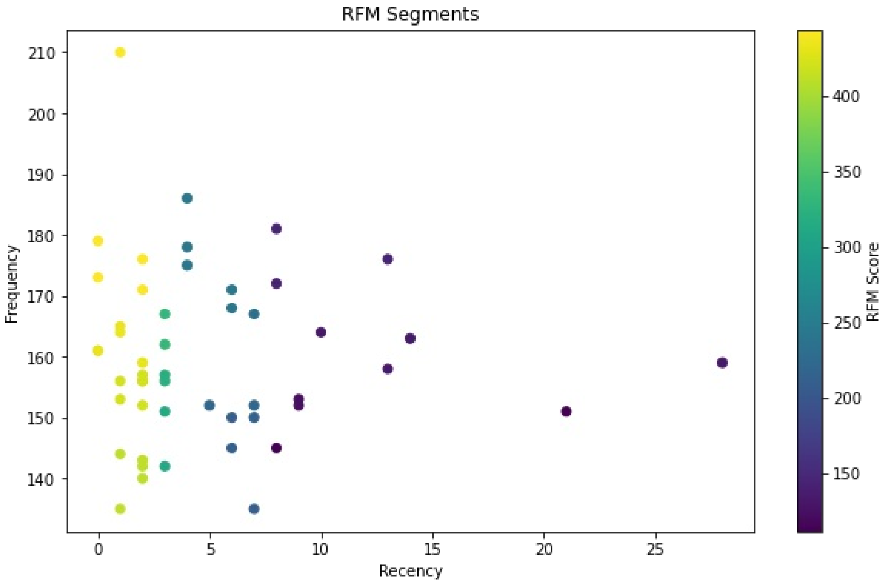
In [67]:
```
rfm_data['RFM_Score'].max()
```

Out[67]:
```
444
```

In [68]:
```
rfm_data['RFM_Score'].min()
```

Out[68]:
```
111
```

In [62]:
```
# Visualize RFM Segments
plt.figure(figsize=(10, 6))
plt.scatter(rfm_data['Recency'], rfm_data['Frequency'], c=rfm_data['RFM_Score'], cmap='viridis')
plt.xlabel('Recency')
plt.ylabel('Frequency')
plt.title('RFM Segments')
plt.colorbar(label='RFM Score')
plt.show()
```



This provides a visual representation of how customers are distributed based on their RFM scores.