1. Importing the libraries and the data

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

2. Importing the data from .csv file

In [2]:
```
data = pd.read_csv('Customer.csv', encoding='unicode_escape')
data
```

Out[2]:

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

541909 rows × 8 columns

In [3]:
```
data.shape
```

Out[3]:
```
(541909, 8)
```

In [4]:
```
#droping all repeatative data from Country,CustomerID Columns
country_cust=data[['Country','CustomerID']].drop_duplicates()
#printing the existing data after droping and also sorting the data
country_cust.groupby(['Country'])['CustomerID'].aggregate('count').reset_index().sort_values('CustomerID', ascending=False)
```

Out[4]:

	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

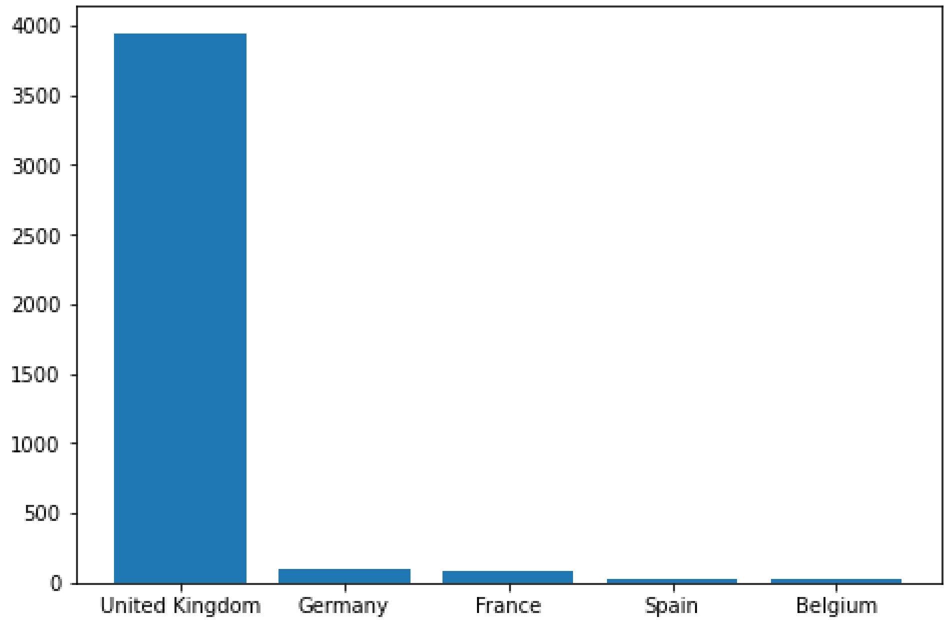
	Country	CustomerID
37	Unspecified	4
18	Israel	4
15	Greece	4
10	EIRE	3
23	Malta	2
35	United Arab Emirates	2
2	Bahrain	2
22	Lithuania	1
8	Czech Republic	1
21	Lebanon	1
28	RSA	1
29	Saudi Arabia	1
30	Singapore	1
17	Iceland	1
4	Brazil	1
11	European Community	1
16	Hong Kong	0

In [5]:

```
CC=country_cust.groupby(['Country'])['CustomerID'].aggregate('count').reset_index().sort_values('CustomerID', ascending=False)
```

In [6]:

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(CC['Country'].head(5).values,CC['CustomerID'].head(5).values)
plt.show()
```



In [7]:
```
#hence there is 90% of data is from UK so we are keeping only data of UK
data=data.query("Country=='United Kingdom'").reset_index(drop=True)
data.shape
```

Out[7]: (495478, 8)

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

Out[8]:

	Quantity	UnitPrice	CustomerID
count	495478.000000	495478.000000	361878.000000
mean	8.605486	4.532422	15547.871368
std	227.588756	99.315438	1594.402590
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	14194.000000
50%	3.000000	2.100000	15514.000000
75%	10.000000	4.130000	16931.000000

	Quantity	UnitPrice	CustomerID
max	80995.000000	38970.000000	18287.000000

4. Checking the data for inconsistencies and further cleaning the data if needed.

In [9]:
```
data.isnull()
```

Out[9]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...
495473	False	False	False	False	False	False	False	False
495474	False	False	False	False	False	False	False	False
495475	False	False	False	False	False	False	False	False
495476	False	False	False	False	False	False	False	False
495477	False	False	False	False	False	False	False	False

495478 rows × 8 columns

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

Out[10]:
```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     133600
```

```
Country         0
dtype: int64
```

In [11]:
```
#droping the null data
data=data[pd.notnull(data['CustomerID'])]
```

In [12]:
```
data.isnull().sum()
```

Out[12]:
```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

In [13]:
```
#validate if there any negative values of Quantity
data.Quantity.min()
```

Out[13]:
```
-80995
```

In [14]:
```
#removing all the negative values in the Quantity Column
data=data[(data['Quantity']>=0)]
data.Quantity.min()
```

Out[14]:
```
1
```

In [15]:
```
#validate if there any negative values of UnitPrice
data.UnitPrice.min()
```

Out[15]:
```
0.0
```

In [16]:
```
#calculating the total amount and storing the data into a newly added column
data['TotalAmount']=data['Quantity']*data['UnitPrice']
data
```

Out[16]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	TotalAmount
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01-12-2010 08:26	2.55	17850.0	United Kingdom	15.30

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	TotalAmount
1	536365	71053	WHITE METAL LANTERN	6	01-12-2010 08:26	3.39	17850.0	United Kingdom	20.34
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01-12-2010 08:26	2.75	17850.0	United Kingdom	22.00
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01-12-2010 08:26	3.39	17850.0	United Kingdom	20.34
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01-12-2010 08:26	3.39	17850.0	United Kingdom	20.34
...	...	...	...	...	...	...	...	...	...
495473	581585	22466	FAIRY TALE COTTAGE NIGHT LIGHT	12	09-12-2011 12:31	1.95	15804.0	United Kingdom	23.40
495474	581586	22061	LARGE CAKE STAND HANGING STRAWBERY	8	09-12-2011 12:49	2.95	13113.0	United Kingdom	23.60
495475	581586	23275	SET OF 3 HANGING OWLS OLLIE BEAK	24	09-12-2011 12:49	1.25	13113.0	United Kingdom	30.00
495476	581586	21217	RED RETROSPOT ROUND CAKE TINS	24	09-12-2011 12:49	8.95	13113.0	United Kingdom	214.80
495477	581586	20685	DOORMAT RED RETROSPOT	10	09-12-2011 12:49	7.08	13113.0	United Kingdom	70.80

354345 rows × 9 columns

In [17]:
```
data.InvoiceDate
```

Out[17]:
```
0         01-12-2010 08:26
1         01-12-2010 08:26
2         01-12-2010 08:26
3         01-12-2010 08:26
4         01-12-2010 08:26
                ...
495473    09-12-2011 12:31
495474    09-12-2011 12:49
495475    09-12-2011 12:49
495476    09-12-2011 12:49
```

```
495477    09-12-2011 12:49
Name: InvoiceDate, Length: 354345, dtype: object
```

In [18]:
```
#Changing the data type of InvoiceDate from object to datetime
data['InvoiceDate']=pd.to_datetime(data['InvoiceDate'])
data.InvoiceDate
```

Out[18]:
```
0        2010-01-12 08:26:00
1        2010-01-12 08:26:00
2        2010-01-12 08:26:00
3        2010-01-12 08:26:00
4        2010-01-12 08:26:00
                 ...
495473   2011-09-12 12:31:00
495474   2011-09-12 12:49:00
495475   2011-09-12 12:49:00
495476   2011-09-12 12:49:00
495477   2011-09-12 12:49:00
Name: InvoiceDate, Length: 354345, dtype: datetime64[ns]
```

In [19]:
```
#Searching for start and end date of this dataset
dataset=data.sort_values(['InvoiceDate'])
dataset
```

Out[19]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	TotalAmount
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-01-12 08:26:00	2.55	17850.0	United Kingdom	15.30
1	536365	71053	WHITE METAL LANTERN	6	2010-01-12 08:26:00	3.39	17850.0	United Kingdom	20.34
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-01-12 08:26:00	2.75	17850.0	United Kingdom	22.00
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-01-12 08:26:00	3.39	17850.0	United Kingdom	20.34
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-01-12 08:26:00	3.39	17850.0	United Kingdom	20.34
...	...	...	...	...	...	...	...	...	...
359471	570876	46000M	POLYESTER FILLER PAD 45x45cm	1	2011-12-10 17:19:00	1.55	16085.0	United Kingdom	1.55

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	TotalAmount
359470	570876	46000S	POLYESTER FILLER PAD 40x40cm	1	2011-12-10 17:19:00	1.45	16085.0	United Kingdom	1.45
359490	570876	23118	PARISIENNE JEWELLERY DRAWER	2	2011-12-10 17:19:00	7.50	16085.0	United Kingdom	15.00
359478	570876	22645	CERAMIC HEART FAIRY CAKE MONEY BANK	2	2011-12-10 17:19:00	1.45	16085.0	United Kingdom	2.90
359488	570876	23503	PLAYING CARDS KEEP CALM & CARRY ON	4	2011-12-10 17:19:00	1.25	16085.0	United Kingdom	5.00

354345 rows × 9 columns

In [20]:
```
import datetime as dt
#Hence we want to calculate recency we set the system date as the letest date from the dataset
Letest_Date=dt.datetime(2011,12,10)
```

In [21]:
```
# Calculating the Recency,Frequency,Monetary
RFMScore=data.groupby('CustomerID').agg({'InvoiceDate': lambda x: (Letest_Date-x.max()).days, 'InvoiceNo': lambda x: len(x),
                                         'TotalAmount': lambda x: x.sum()})
```

In [22]:
```
#Changing the 'InvoiceDate' datatype from datetime to integer
RFMScore['InvoiceDate']= RFMScore['InvoiceDate'].astype(int)
```

In [23]:
```
#Renaming the Column names
RFMScore.rename(columns={'InvoiceDate':'Recency','InvoiceNo':'Frequency','TotalAmount':'Monetary'},inplace=True)
```

In [24]:
```
RFMScore.head()
```

Out[24]:

CustomerID	Recency	Frequency	Monetary
12346.0	325	1	77183.60
12747.0	22	103	4196.01
12748.0	4	4596	33719.73
12749.0	22	199	4090.88

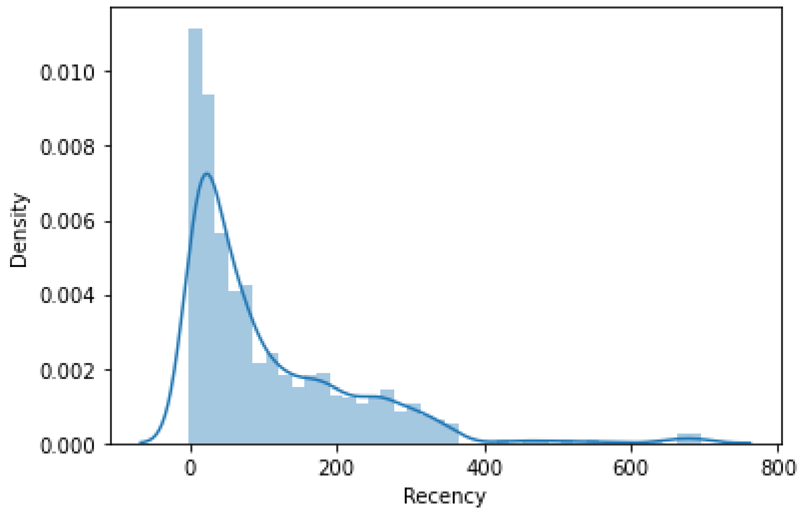
	Recency	Frequency	Monetary
CustomerID			
12820.0	44	59	942.34

In [25]:
```
RFMScore.Recency.describe()
```

Out[25]:
```
count    3921.000000
mean      104.586585
std       115.044919
min        -1.000000
25%        21.000000
50%        60.000000
75%       161.000000
max       696.000000
Name: Recency, dtype: float64
```

In [26]:
```
#Recency distribution plot
import seaborn as sns
x = RFMScore['Recency']

ax = sns.distplot(x)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and w
ill be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibili
ty) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

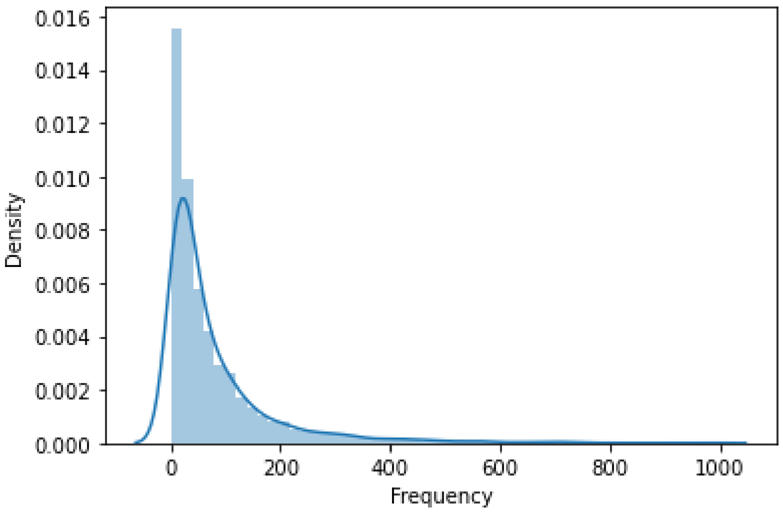


In [27]:
```
RFMScore.Frequency.describe()
```

Out[27]:
```
count    3921.000000
mean       90.371079
std       217.796155
min         1.000000
25%        17.000000
50%        41.000000
75%        99.000000
max      7847.000000
Name: Frequency, dtype: float64
```

In [28]:
```
#Frequency distribution plot, taking observations which have frequency less than 1000
import seaborn as sns
x = RFMScore.query('Frequency < 1000')['Frequency']

ax = sns.distplot(x)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and w
ill be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibili
ty) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

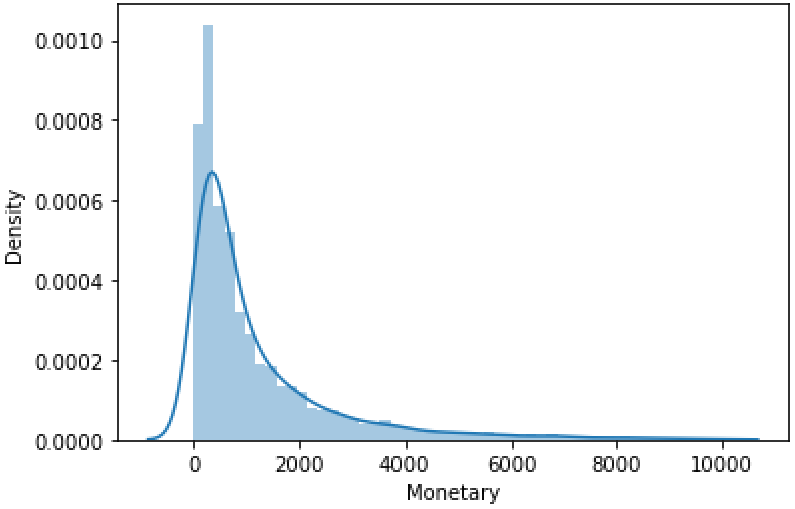


In [29]:
```
RFMScore.Monetary.describe()
```

Out[29]:
```
count       3921.000000
mean        1863.910113
std         7481.922217
min            0.000000
25%          300.040000
50%          651.820000
75%         1575.890000
max       259657.300000
Name: Monetary, dtype: float64
```

In [30]:
```
#Monateray distribution plot, taking observations which have monetary value less than 10000
import seaborn as sns
x = RFMScore.query('Monetary < 10000')['Monetary']

ax = sns.distplot(x)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and w
ill be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibili
ty) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [31]:
```
#Creating Different quantile level to seggrigate the customer
quantiles=RFMScore.quantile(q=[0.2,0.4,0.6,0.8])
quantiles=quantiles.to_dict()
```

In [32]:
```
quantiles
```

Out[32]:
```
{'Recency': {0.2: 17.0, 0.4: 42.0, 0.6: 86.0, 0.8: 187.0},
 'Frequency': {0.2: 13.0, 0.4: 29.0, 0.6: 58.0, 0.8: 120.0},
 'Monetary': {0.2: 241.61999999999998,
  0.4: 465.5499999999999,
  0.6: 897.62,
  0.8: 1957.3200000000002}}
```

In [33]:
```
#Defining Function to allote points based on the customers
def RScore(x,p,d):
    if x<=d[p][0.2]:
        return 4
    elif x<=d[p][0.4]:
        return 3
    elif x<=d[p][0.6]:
        return 2
    elif x<=d[p][0.8]:
        return 1
    else:
        return 0

#Defining Function to allote points based on the customers
```

```python
def FScore(x,p,d):
    if x<=d[p][0.2]:
        return 1
    elif x<=d[p][0.4]:
        return 2
    elif x<=d[p][0.6]:
        return 3
    elif x<=d[p][0.8]:
        return 4
    else:
        return 5

#Defining Function to allote points based on the customers
def MScore(x,p,d):
    if x<=d[p][0.2]:
        return 1
    elif x<=d[p][0.4]:
        return 2
    elif x<=d[p][0.6]:
        return 3
    elif x<=d[p][0.8]:
        return 4
    else:
        return 5
```

In [34]:

```python
RFMScore['R']=RFMScore['Recency'].apply(RScore, args=('Recency',quantiles))
RFMScore['F']=RFMScore['Frequency'].apply(RScore, args=('Frequency',quantiles))
RFMScore['M']=RFMScore['Monetary'].apply(MScore, args=('Monetary',quantiles))
```

In [35]:

```python
RFMScore.head(7)
```

Out[35]:

	Recency	Frequency	Monetary	R	F	M
CustomerID						
12346.0	325	1	77183.60	0	4	5
12747.0	22	103	4196.01	3	1	5
12748.0	4	4596	33719.73	4	0	5
12749.0	22	199	4090.88	3	0	5
12820.0	44	59	942.34	2	1	4

CustomerID	Recency	Frequency	Monetary	R	F	M
12821.0	95	6	92.72	1	4	1
12822.0	70	46	948.88	2	2	4

In [36]:

```
RFMScore['RFMTotal']=RFMScore[['R','F','M']].sum(axis=1)
RFMScore.head(7)
```

Out[36]:

CustomerID	Recency	Frequency	Monetary	R	F	M	RFMTotal
12346.0	325	1	77183.60	0	4	5	9
12747.0	22	103	4196.01	3	1	5	9
12748.0	4	4596	33719.73	4	0	5	9
12749.0	22	199	4090.88	3	0	5	8
12820.0	44	59	942.34	2	1	4	7
12821.0	95	6	92.72	1	4	1	6
12822.0	70	46	948.88	2	2	4	8

In [37]:

```
#Assign Loyalty Level to each customer
Loyalty_Level = ['BAD','AVERAGE', 'GOOD', 'VALUABLE', 'PREMIUME']
Score_cuts = pd.qcut(RFMScore.RFMTotal, q = 5, labels = Loyalty_Level)
RFMScore['RFM_Loyalty_Level'] = Score_cuts.values
RFMScore.reset_index().head()
```

Out[37]:

	CustomerID	Recency	Frequency	Monetary	R	F	M	RFMTotal	RFM_Loyalty_Level
0	12346.0	325	1	77183.60	0	4	5	9	VALUABLE
1	12747.0	22	103	4196.01	3	1	5	9	VALUABLE
2	12748.0	4	4596	33719.73	4	0	5	9	VALUABLE
3	12749.0	22	199	4090.88	3	0	5	8	GOOD

	CustomerID	Recency	Frequency	Monetary	R	F	M	RFMTotal	RFM_Loyalty_Level
4	12820.0	44	59	942.34	2	1	4	7	AVERAGE

In [38]:

```
#Validate the data for RFMGroup = 111
RFMScore[RFMScore['RFM_Loyalty_Level']=='VALUABLE'].sort_values('Monetary', ascending=False).reset_index().head(10)
```

Out[38]:

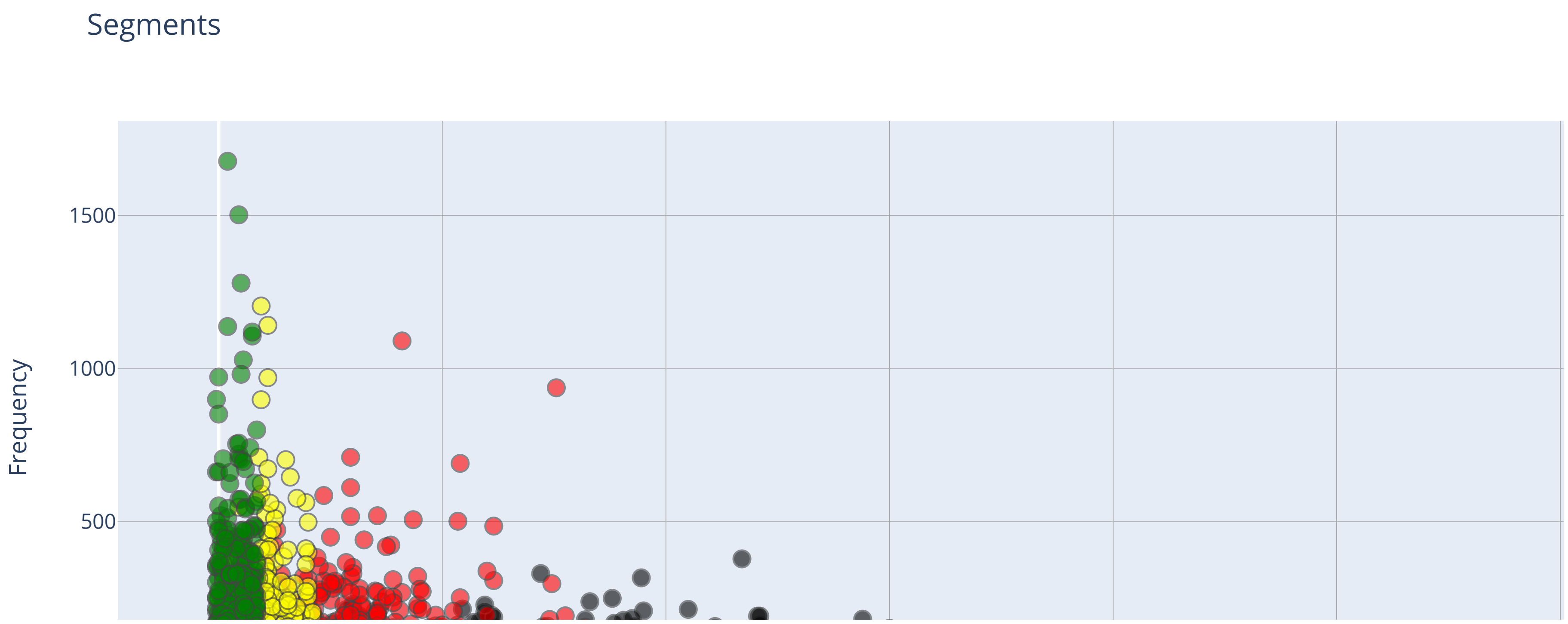
	CustomerID	Recency	Frequency	Monetary	R	F	M	RFMTotal	RFM_Loyalty_Level
0	18102.0	11	431	259657.30	4	0	5	9	VALUABLE
1	17450.0	2	337	194550.79	4	0	5	9	VALUABLE
2	17511.0	5	963	91062.38	4	0	5	9	VALUABLE
3	12346.0	325	1	77183.60	0	4	5	9	VALUABLE
4	16684.0	11	277	66653.56	4	0	5	9	VALUABLE
5	14096.0	11	5111	65164.79	4	0	5	9	VALUABLE
6	15311.0	-1	2379	60767.90	4	0	5	9	VALUABLE
7	13089.0	5	1818	58825.83	4	0	5	9	VALUABLE
8	15061.0	4	403	54534.14	4	0	5	9	VALUABLE
9	14088.0	10	589	50491.81	4	0	5	9	VALUABLE

In [39]:

```
import chart_studio as cs
import plotly.offline as po
import plotly.graph_objs as gobj

#Recency Vs Frequency
graph = RFMScore.query("Monetary < 50000 and Frequency < 2000")

plot_data = [
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'BAD'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'BAD'")['Frequency'],
        mode='markers',
        name='BAD',
        marker= dict(size= 10,
            line= dict(width=1),
```

```
            color= 'black',
            opacity= 0.6
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'AVERAGE'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'AVERAGE'")['Frequency'],
        mode='markers',
        name='AVERAGE',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'red',
            opacity= 0.6
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'GOOD'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'GOOD'")['Frequency'],
        mode='markers',
        name='GOOD',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'yellow',
            opacity= 0.6
           )
    ),
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'VALUABLE'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'VALUABLE'")['Frequency'],
        mode='markers',
        name='VALUABLE',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'green',
            opacity= 0.6
           )
    ),
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'PREMIUME'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'PREMIUME'")['Frequency'],
        mode='markers',
        name='PREMIUME',
        marker= dict(size= 10,
            line= dict(width=1),
```

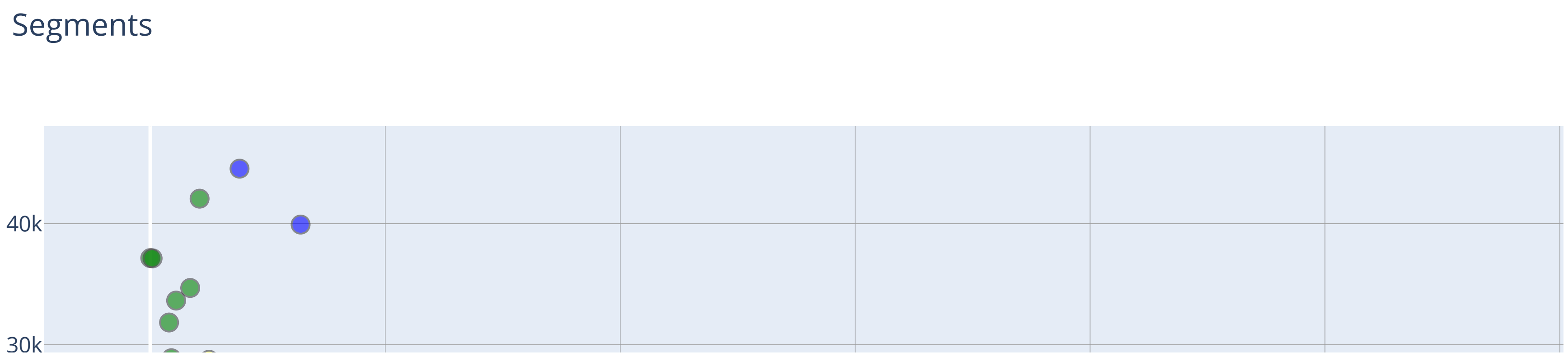
```
                color= 'blue',
                opacity= 0.6
               )
       ),
]

plot_layout = gobj.Layout(
        yaxis= {'title': "Frequency"},
        xaxis= {'title': "Recency"},
        title='Segments'
    )
fig = gobj.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```

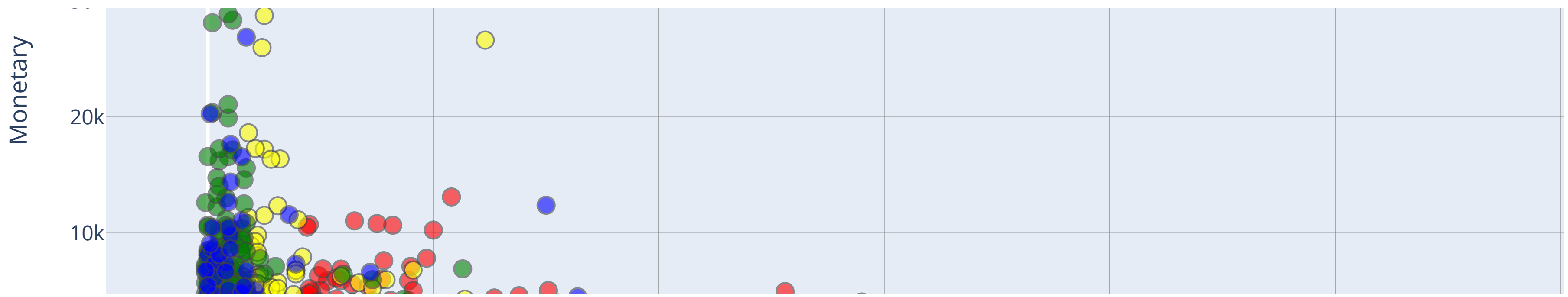


Recency Vs Monetary

In [40]:
```
graph = RFMScore.query("Monetary < 50000 and Frequency < 2000")

plot_data = [
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'BAD'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'BAD'")['Monetary'],
        mode='markers',
        name='BAD',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'black',
            opacity= 0.6
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'AVERAGE'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'AVERAGE'")['Monetary'],
        mode='markers',
        name='AVERAGE',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'red',
            opacity= 0.6
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'GOOD'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'GOOD'")['Monetary'],
        mode='markers',
        name='GOOD',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'yellow',
            opacity= 0.6
           )
    ),
    gobj.Scatter(
```

```
        x=graph.query("RFM_Loyalty_Level == 'VALUABLE'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'VALUABLE'")['Monetary'],
        mode='markers',
        name='VALUABLE',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'green',
            opacity= 0.6
           )
    ),
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'PREMIUME'")['Recency'],
        y=graph.query("RFM_Loyalty_Level == 'PREMIUME'")['Monetary'],
        mode='markers',
        name='PREMIUME',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.6
           )
    ),
]

plot_layout = gobj.Layout(
        yaxis= {'title': "Monetary"},
        xaxis= {'title': "Recency"},
        title='Segments'
    )
fig = gobj.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```





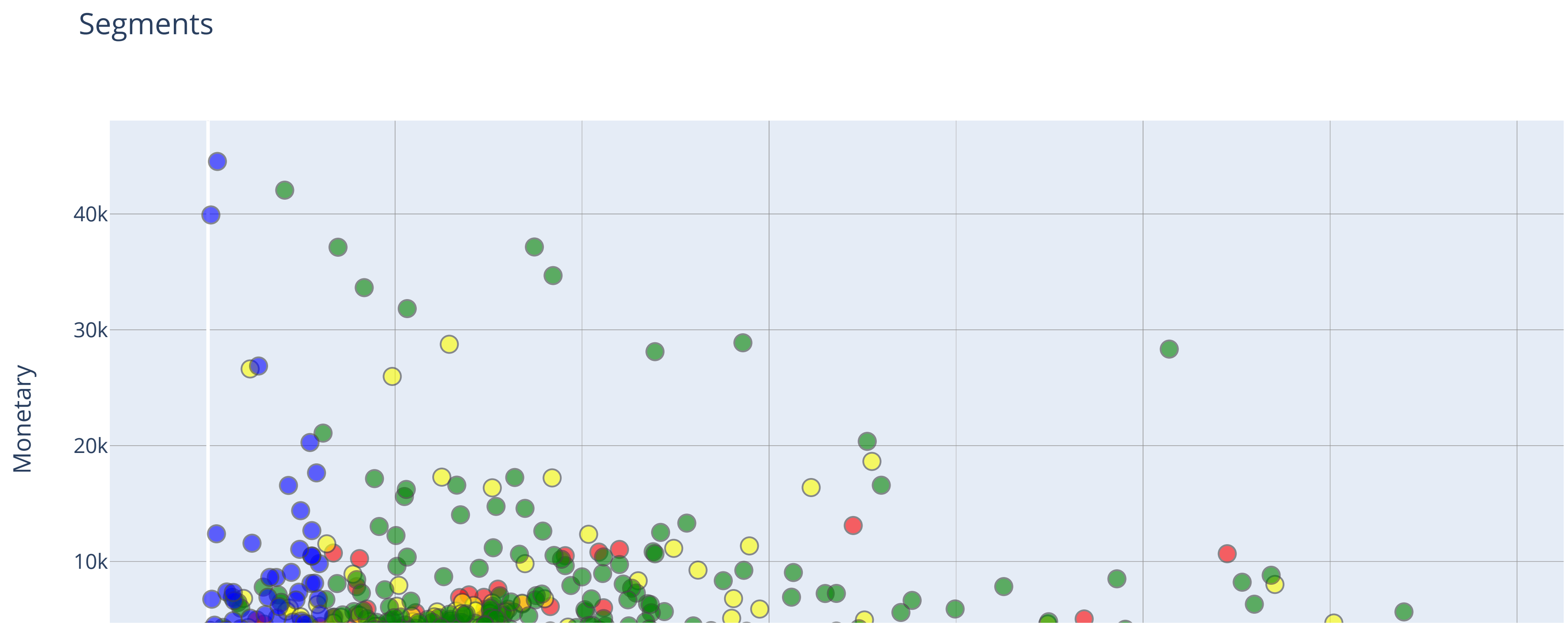
Frequency Vs Monetary

In [41]:

```
graph = RFMScore.query("Monetary < 50000 and Frequency < 2000")

plot_data = [
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'BAD'")['Frequency'],
        y=graph.query("RFM_Loyalty_Level == 'BAD'")['Monetary'],
        mode='markers',
        name='BAD',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'black',
            opacity= 0.8
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'AVERAGE'")['Frequency'],
        y=graph.query("RFM_Loyalty_Level == 'AVERAGE'")['Monetary'],
        mode='markers',
        name='AVERAGE',
        marker= dict(size= 10,
            line= dict(width=1),
```

```
            color= 'red',
            opacity= 0.6
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'GOOD'")['Frequency'],
        y=graph.query("RFM_Loyalty_Level == 'GOOD'")['Monetary'],
        mode='markers',
        name='GOOD',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'yellow',
            opacity= 0.6
           )
    ),
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'VALUABLE'")['Frequency'],
        y=graph.query("RFM_Loyalty_Level == 'VALUABLE'")['Monetary'],
        mode='markers',
        name='VALUABLE',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'green',
            opacity= 0.6
           )
    ),
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'PREMIUME'")['Frequency'],
        y=graph.query("RFM_Loyalty_Level == 'PREMIUME'")['Monetary'],
        mode='markers',
        name='PREMIUME',
        marker= dict(size= 10,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.6
           )
    ),
]

plot_layout = gobj.Layout(
        yaxis= {'title': "Monetary"},
        xaxis= {'title': "Frequency"},
        title='Segments'
    )
```

```
fig = gobj.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```



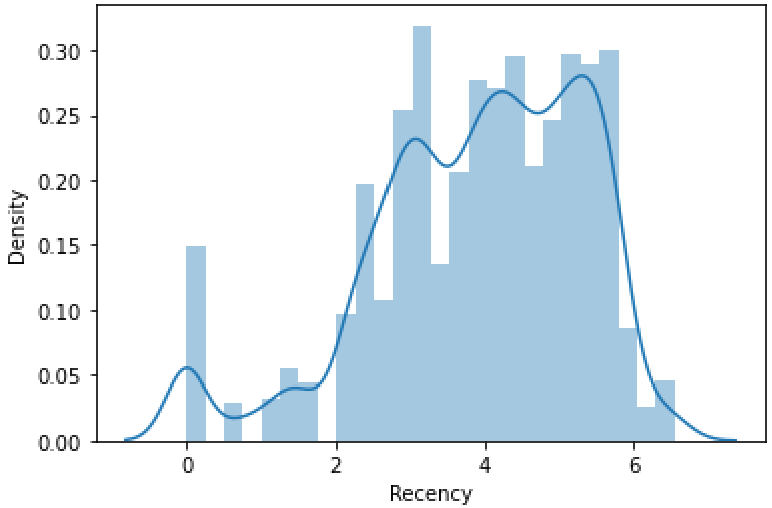
In [42]:

```
#Handle negative and zero values so as to handle infinite numbers during log transformation
def handle_neg_n_zero(num):
    if num <= 0:
        return 1
    else:
        return num
#Apply handle_neg_n_zero function to Recency and Monetary columns
```

```
RFMScore['Recency'] = [handle_neg_n_zero(x) for x in RFMScore.Recency]
RFMScore['Monetary'] = [handle_neg_n_zero(x) for x in RFMScore.Monetary]

#Perform Log transformation to bring data into normal or near normal distribution
Log_Tfd_Data = RFMScore[['Recency', 'Frequency', 'Monetary']].apply(np.log, axis = 1).round(3)
```

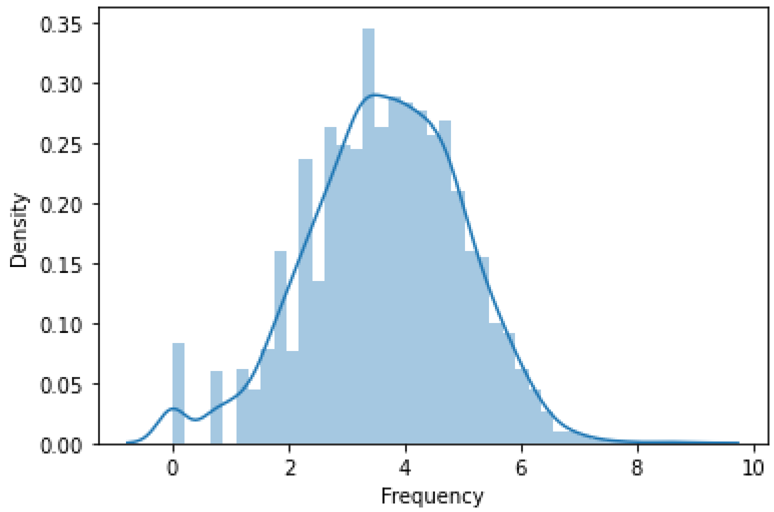
In [43]:
```
#Data distribution after data normalization for Recency
Recency_Plot = Log_Tfd_Data['Recency']
ax = sns.distplot(Recency_Plot)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a fig
ure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
```



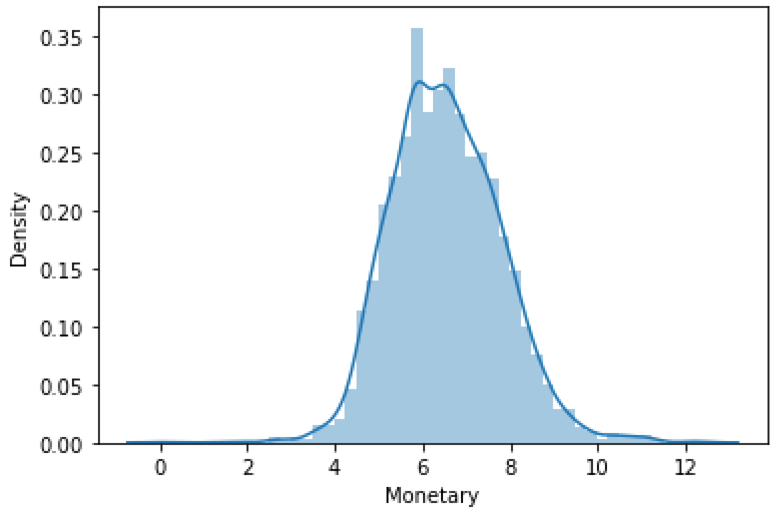
In [44]:
```
#Data distribution after data normalization for Frequency
Frequency_Plot = Log_Tfd_Data.query('Frequency < 1000')['Frequency']
ax = sns.distplot(Frequency_Plot)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a fig
ure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
```



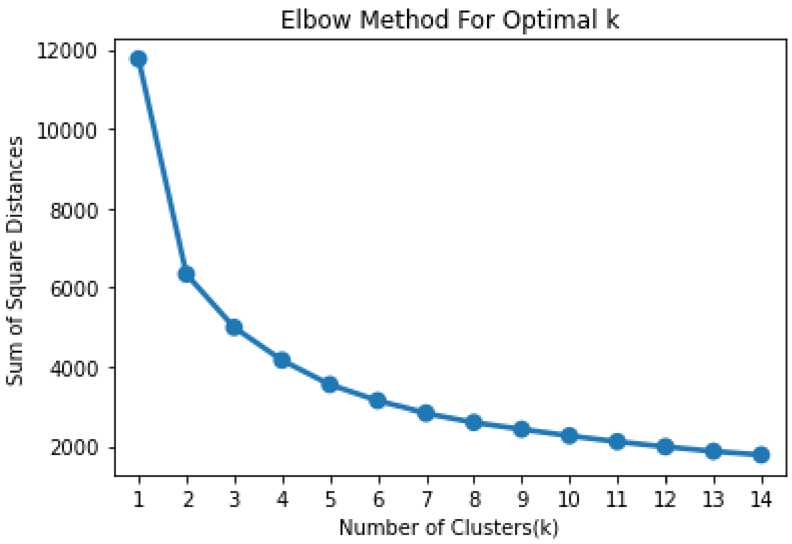
In [45]:
```
#Data distribution after data normalization for Monetary
Monetary_Plot = Log_Tfd_Data.query('Monetary < 10000')['Monetary']
ax = sns.distplot(Monetary_Plot)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a fig
ure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
```



In [46]:
```python
from sklearn.preprocessing import StandardScaler

#Bring the data on same scale
scaleobj = StandardScaler()
Scaled_Data = scaleobj.fit_transform(Log_Tfd_Data)

#Transform it back to dataframe
Scaled_Data = pd.DataFrame(Scaled_Data, index = RFMScore.index, columns = Log_Tfd_Data.columns)
```

In [47]:
```python
from sklearn.cluster import KMeans

sum_of_sq_dist = {}
for k in range(1,15):
    km = KMeans(n_clusters= k, init= 'k-means++', max_iter= 1000)
    km = km.fit(Scaled_Data)
    sum_of_sq_dist[k] = km.inertia_

#Plot the graph for the sum of square distance values and Number of Clusters
sns.pointplot(x = list(sum_of_sq_dist.keys()), y = list(sum_of_sq_dist.values()))
plt.xlabel('Number of Clusters(k)')
plt.ylabel('Sum of Square Distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



In [48]:
```python
#Perform K-Mean Clustering or build the K-Means clustering model
KMean_clust = KMeans(n_clusters= 3, init= 'k-means++', max_iter= 10000)
```

```
KMean_clust.fit(Scaled_Data)

#Find the clusters for the observation given in the dataset
RFMScore['Cluster'] = KMean_clust.labels_
RFMScore.head()
```

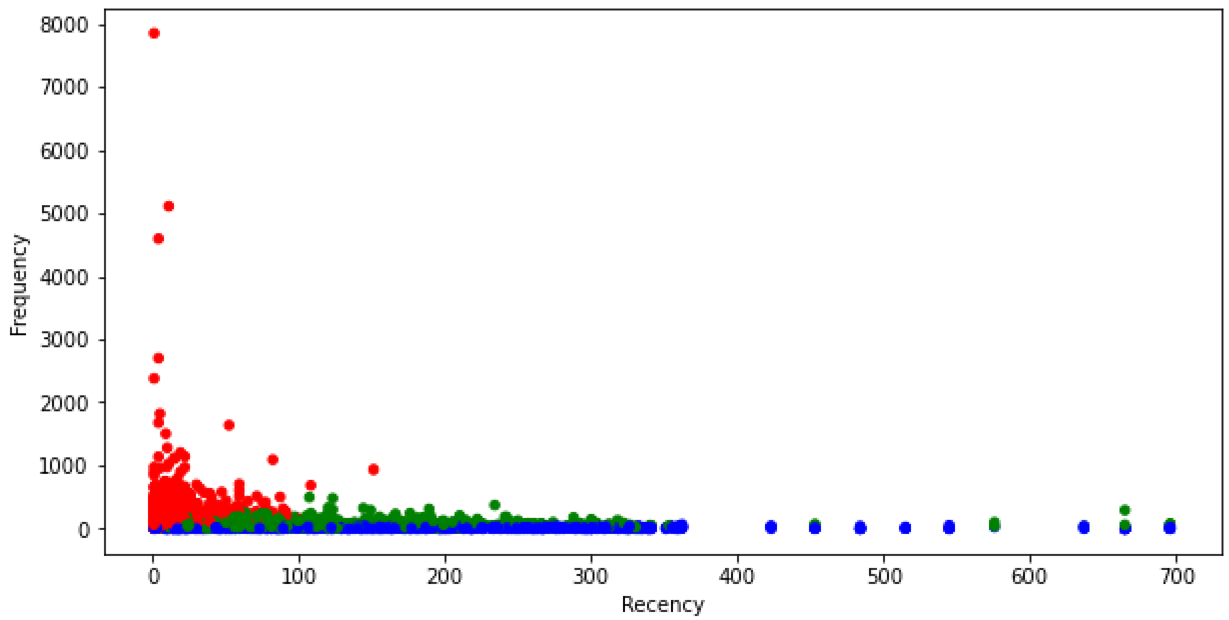
Out[48]:

CustomerID	Recency	Frequency	Monetary	R	F	M	RFMTotal	RFM_Loyalty_Level	Cluster
12346.0	325	1	77183.60	0	4	5	9	VALUABLE	1
12747.0	22	103	4196.01	3	1	5	9	VALUABLE	0
12748.0	4	4596	33719.73	4	0	5	9	VALUABLE	0
12749.0	22	199	4090.88	3	0	5	8	GOOD	0
12820.0	44	59	942.34	2	1	4	7	AVERAGE	1

In [49]:

```
from matplotlib import pyplot as plt
plt.figure(figsize=(7,7))

##Scatter Plot Frequency Vs Recency
Colors = ["red", "green", "blue"]
RFMScore['Color'] = RFMScore['Cluster'].map(lambda p: Colors[p])
ax = RFMScore.plot(
    kind="scatter",
    x="Recency", y="Frequency",
    figsize=(10,5),
    c = RFMScore['Color']
)
```

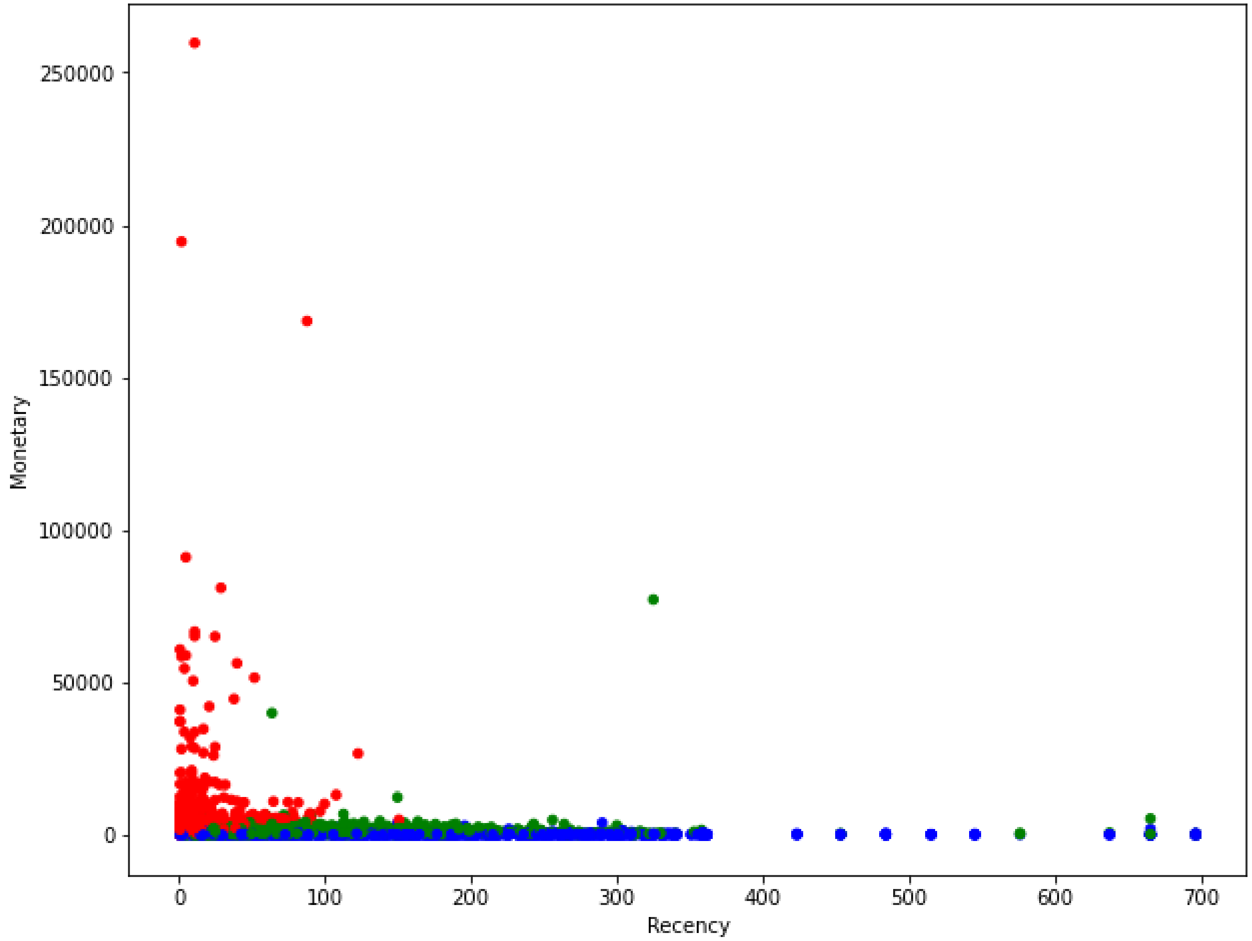
```
<Figure size 504x504 with 0 Axes>
```



In [50]:

```
from matplotlib import pyplot as plt
plt.figure(figsize=(7,7))

##Scatter Plot Recency Vs Monetary
Colors = ["red", "green", "blue"]
RFMScore['Color'] = RFMScore['Cluster'].map(lambda p: Colors[p])
ax = RFMScore.plot(
    kind="scatter",
    x="Recency", y="Monetary",
    figsize=(10,8),
    c = RFMScore['Color']
)
```

```
<Figure size 504x504 with 0 Axes>
```



In [51]:
```
from matplotlib import pyplot as plt
plt.figure(figsize=(7,7))

##Scatter Plot Monetary Vs Frequency
Colors = ["red", "green", "blue"]
RFMScore['Color'] = RFMScore['Cluster'].map(lambda p: Colors[p])
ax = RFMScore.plot(
    kind="scatter",
    x="Monetary", y="Frequency",
    figsize=(10,8),
    c = RFMScore['Color']
)
```

<Figure size 504x504 with 0 Axes>