Import Necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Import data

In [2]:
```
sales_data = pd.read_excel('sales_data.xlsx')
sales_data
```

Out[2]:

	CustomerID	TOTAL_ORDERS	REVENUE	AVERAGE_ORDER_VALUE	CARRIAGE_REVE
0	22.0	124.0	11986.54	96.67	52
1	29.0	82.0	11025.96	134.46	9
2	83.0	43.0	7259.69	168.83	17
3	95.0	44.0	6992.27	158.92	9
4	124.0	55.0	6263.44	113.88	17
...	...	...	...	...	
4996	173987.0	1.0	117.49	117.49	
4997	174004.0	1.0	117.49	117.49	
4998	174038.0	1.0	117.49	117.49	
4999	200783.0	2.0	94.14	47.07	
5000	NaN	NaN	NaN	NaN	

5001 rows × 42 columns

Data Understanding

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

Out[3]: (5001, 42)

In [4]:
```
sales_data.isna().sum()
```

Out[4]:
```
CustomerID                    1
TOTAL_ORDERS                  1
REVENUE                       1
AVERAGE_ORDER_VALUE           1
CARRIAGE_REVENUE              1
AVERAGESHIPPING               1
FIRST_ORDER_DATE              1
LATEST_ORDER_DATE             1
AVGDAYSBETWEENORDERS          1
DAYSSINCELASTORDER            1
MONDAY_ORDERS                 0
TUESDAY_ORDERS                0
WEDNESDAY_ORDERS              0
THURSDAY_ORDERS               0
FRIDAY_ORDERS                 0
SATURDAY_ORDERS               0
SUNDAY_ORDERS                 0
MONDAY_REVENUE                0
TUESDAY_REVENUE               0
WEDNESDAY_REVENUE             0
THURSDAY_REVENUE              0
FRIDAY_REVENUE                0
SATURDAY_REVENUE              0
SUNDAY_REVENUE                0
WEEK1_DAY01_DAY07_ORDERS      1
WEEK2_DAY08_DAY15_ORDERS      1
WEEK3_DAY16_DAY23_ORDERS      1
WEEK4_DAY24_DAY31_ORDERS      1
WEEK1_DAY01_DAY07_REVENUE     1
WEEK2_DAY08_DAY15_REVENUE     1
WEEK3_DAY16_DAY23_REVENUE     1
WEEK4_DAY24_DAY31_REVENUE     1
TIME_0000_0600_ORDERS         1
TIME_0601_1200_ORDERS         1
TIME_1200_1800_ORDERS         1
TIME_1801_2359_ORDERS         1
TIME_0000_0600_REVENUE        1
TIME_0601_1200_REVENUE        1
TIME_1200_1800_REVENUE        1
TIME_1801_2359_REVENUE        1
Unnamed: 40                5001
Unnamed: 41                5000
dtype: int64
```

In [5]:
```
sales_data.dtypes
```

Out[5]:
```
CustomerID                         float64
TOTAL_ORDERS                       float64
REVENUE                            float64
AVERAGE_ORDER_VALUE                float64
CARRIAGE_REVENUE                   float64
AVERAGESHIPPING                    float64
FIRST_ORDER_DATE            datetime64[ns]
LATEST_ORDER_DATE           datetime64[ns]
AVGDAYSBETWEENORDERS               float64
DAYSSINCELASTORDER                 float64
MONDAY_ORDERS                        int64
TUESDAY_ORDERS                       int64
WEDNESDAY_ORDERS                     int64
THURSDAY_ORDERS                      int64
FRIDAY_ORDERS                        int64
SATURDAY_ORDERS                      int64
SUNDAY_ORDERS                        int64
MONDAY_REVENUE                     float64
TUESDAY_REVENUE                    float64
WEDNESDAY_REVENUE                  float64
THURSDAY_REVENUE                   float64
FRIDAY_REVENUE                     float64
SATURDAY_REVENUE                   float64
SUNDAY_REVENUE                     float64
WEEK1_DAY01_DAY07_ORDERS           float64
WEEK2_DAY08_DAY15_ORDERS           float64
WEEK3_DAY16_DAY23_ORDERS           float64
WEEK4_DAY24_DAY31_ORDERS           float64
WEEK1_DAY01_DAY07_REVENUE          float64
WEEK2_DAY08_DAY15_REVENUE          float64
WEEK3_DAY16_DAY23_REVENUE          float64
WEEK4_DAY24_DAY31_REVENUE          float64
TIME_0000_0600_ORDERS              float64
TIME_0601_1200_ORDERS              float64
TIME_1200_1800_ORDERS              float64
TIME_1801_2359_ORDERS              float64
TIME_0000_0600_REVENUE             float64
TIME_0601_1200_REVENUE             float64
TIME_1200_1800_REVENUE             float64
TIME_1801_2359_REVENUE             float64
Unnamed: 40                        float64
Unnamed: 41                        float64
dtype: object
```

In [6]:
```
sales_data.describe().T
```

Out[6]:

	count	mean	std	min	25%
CustomerID	5000.0	40709.227800	49949.848017	1.00	1687.5000
TOTAL_ORDERS	5000.0	12.870400	12.679880	1.00	3.0000

REVENUE	5000.0	1681.523840	1998.618678	38.50	315.0975
AVERAGE_ORDER_VALUE	5000.0	136.537378	91.651569	10.68	83.0250
CARRIAGE_REVENUE	5000.0	46.036376	47.879226	0.00	9.9800
AVERAGESHIPPING	5000.0	3.592574	2.021360	0.00	2.5000
AVGDAYSBETWEENORDERS	5000.0	163.159618	259.699496	0.00	21.6700
DAYSSINCELASTORDER	5000.0	87.420000	80.156513	1.00	7.0000
MONDAY_ORDERS	5001.0	3.257349	115.174855	0.00	0.0000
TUESDAY_ORDERS	5001.0	3.508098	124.041478	0.00	0.0000
WEDNESDAY_ORDERS	5001.0	3.595281	127.123556	0.00	0.0000
THURSDAY_ORDERS	5001.0	4.267147	150.871508	0.00	0.0000
FRIDAY_ORDERS	5001.0	3.891622	137.601408	0.00	0.0000
SATURDAY_ORDERS	5001.0	3.366127	119.023862	0.00	0.0000
SUNDAY_ORDERS	5001.0	3.850030	136.125183	0.00	0.0000
MONDAY_REVENUE	5001.0	430.330606	15218.162544	0.00	0.0000
TUESDAY_REVENUE	5001.0	466.927475	16511.866307	0.00	0.0000
WEDNESDAY_REVENUE	5001.0	471.284331	16665.499459	0.00	0.0000
THURSDAY_REVENUE	5001.0	531.793233	18803.767727	0.00	0.0000
FRIDAY_REVENUE	5001.0	501.060896	17717.933042	0.00	0.0000
SATURDAY_REVENUE	5001.0	441.862563	15625.231731	0.00	0.0000
SUNDAY_REVENUE	5001.0	521.782304	18450.432062	0.00	0.0000
WEEK1_DAY01_DAY07_ORDERS	5000.0	2.997800	3.256980	0.00	1.0000
WEEK2_DAY08_DAY15_ORDERS	5000.0	3.062600	3.792461	0.00	0.0000
WEEK3_DAY16_DAY23_ORDERS	5000.0	3.230000	3.921043	0.00	0.0000
WEEK4_DAY24_DAY31_ORDERS	5000.0	3.580000	3.970384	0.00	1.0000
WEEK1_DAY01_DAY07_REVENUE	5000.0	378.638346	515.590218	0.00	63.9900
WEEK2_DAY08_DAY15_REVENUE	5000.0	406.595734	619.413277	0.00	0.0000
WEEK3_DAY16_DAY23_REVENUE	5000.0	421.826908	643.449120	0.00	0.0000
WEEK4_DAY24_DAY31_REVENUE	5000.0	474.462852	617.579321	0.00	80.0000
TIME_0000_0600_ORDERS	5000.0	1.028800	2.174331	0.00	0.0000
TIME_0601_1200_ORDERS	5000.0	3.746200	4.700234	0.00	1.0000
TIME_1200_1800_ORDERS	5000.0	4.434000	5.044793	0.00	1.0000
TIME_1801_2359_ORDERS	5000.0	3.661400	4.581894	0.00	1.0000
TIME_0000_0600_REVENUE	5000.0	131.062636	331.289349	0.00	0.0000

TIME_0601_1200_REVENUE	5000.0	486.863868	789.029911	0.00	35.0000
TIME_1200_1800_REVENUE	5000.0	584.731626	804.290026	0.00	89.9900
TIME_1801_2359_REVENUE	5000.0	478.865710	743.244248	0.00	1.0000
Unnamed: 40	0.0	NaN	NaN	NaN	NaN
Unnamed: 41	1.0	33212.760000	NaN	33212.76	33212.7600

Data Understanding

In [7]:

```
sales_data.drop(['Unnamed: 40','Unnamed: 41'],axis =1,inplace=T
```

In [8]:

```
sales_data.dropna(inplace=True)
```

In [9]:

```
sales_data.shape
```

Out[9]:

```
(5000, 40)
```

In [10]:
```
sales_data.isna().sum()
```

Out[10]:
```
CustomerID                   0
TOTAL_ORDERS                 0
REVENUE                      0
AVERAGE_ORDER_VALUE          0
CARRIAGE_REVENUE             0
AVERAGESHIPPING              0
FIRST_ORDER_DATE             0
LATEST_ORDER_DATE            0
AVGDAYSBETWEENORDERS         0
DAYSSINCELASTORDER           0
MONDAY_ORDERS                0
TUESDAY_ORDERS               0
WEDNESDAY_ORDERS             0
THURSDAY_ORDERS              0
FRIDAY_ORDERS                0
SATURDAY_ORDERS              0
SUNDAY_ORDERS                0
MONDAY_REVENUE               0
TUESDAY_REVENUE              0
WEDNESDAY_REVENUE            0
THURSDAY_REVENUE             0
FRIDAY_REVENUE               0
SATURDAY_REVENUE             0
SUNDAY_REVENUE               0
WEEK1_DAY01_DAY07_ORDERS     0
WEEK2_DAY08_DAY15_ORDERS     0
WEEK3_DAY16_DAY23_ORDERS     0
WEEK4_DAY24_DAY31_ORDERS     0
WEEK1_DAY01_DAY07_REVENUE    0
WEEK2_DAY08_DAY15_REVENUE    0
WEEK3_DAY16_DAY23_REVENUE    0
WEEK4_DAY24_DAY31_REVENUE    0
TIME_0000_0600_ORDERS        0
TIME_0601_1200_ORDERS        0
TIME_1200_1800_ORDERS        0
TIME_1801_2359_ORDERS        0
TIME_0000_0600_REVENUE       0
TIME_0601_1200_REVENUE       0
TIME_1200_1800_REVENUE       0
TIME_1801_2359_REVENUE       0
dtype: int64
```

In [11]:
```
sales_data.columns
```

Out[11]:
```
Index(['CustomerID', 'TOTAL_ORDERS', 'REVENUE', 'AVERAGE_ORDER_VAL
UE',
       'CARRIAGE_REVENUE', 'AVERAGESHIPPING', 'FIRST_ORDER_DATE',
       'LATEST_ORDER_DATE', 'AVGDAYSBETWEENORDERS', 'DAYSSINCELAST
ORDER',
       'MONDAY_ORDERS', 'TUESDAY_ORDERS', 'WEDNESDAY_ORDERS',
       'THURSDAY_ORDERS', 'FRIDAY_ORDERS', 'SATURDAY_ORDERS', 'SUN
DAY_ORDERS',
       'MONDAY_REVENUE', 'TUESDAY_REVENUE', 'WEDNESDAY_REVENUE',
       'THURSDAY_REVENUE', 'FRIDAY_REVENUE', 'SATURDAY_REVENUE',
       'SUNDAY_REVENUE', 'WEEK1_DAY01_DAY07_ORDERS',
       'WEEK2_DAY08_DAY15_ORDERS', 'WEEK3_DAY16_DAY23_ORDERS',
       'WEEK4_DAY24_DAY31_ORDERS', 'WEEK1_DAY01_DAY07_REVENUE',
       'WEEK2_DAY08_DAY15_REVENUE', 'WEEK3_DAY16_DAY23_REVENUE',
       'WEEK4_DAY24_DAY31_REVENUE', 'TIME_0000_0600_ORDERS',
       'TIME_0601_1200_ORDERS', 'TIME_1200_1800_ORDERS',
       'TIME_1801_2359_ORDERS', 'TIME_0000_0600_REVENUE',
       'TIME_0601_1200_REVENUE', 'TIME_1200_1800_REVENUE',
       'TIME_1801_2359_REVENUE'],
      dtype='object')
```

In [12]:
```
sales_data.head()
```

Out[12]:

	CustomerID	TOTAL_ORDERS	REVENUE	AVERAGE_ORDER_VALUE	CARRIAGE_REVENUI
0	22.0	124.0	11986.54	96.67	529.5
1	29.0	82.0	11025.96	134.46	97.9
2	83.0	43.0	7259.69	168.83	171.6
3	95.0	44.0	6992.27	158.92	92.8
4	124.0	55.0	6263.44	113.88	179.0

5 rows × 40 columns

In [13]:
```
sales_data.dtypes
```

Out[13]:
```
CustomerID                           float64
TOTAL_ORDERS                         float64
REVENUE                              float64
AVERAGE_ORDER_VALUE                  float64
CARRIAGE_REVENUE                     float64
AVERAGESHIPPING                      float64
FIRST_ORDER_DATE              datetime64[ns]
LATEST_ORDER_DATE             datetime64[ns]
AVGDAYSBETWEENORDERS                 float64
DAYSSINCELASTORDER                   float64
MONDAY_ORDERS                          int64
TUESDAY_ORDERS                         int64
WEDNESDAY_ORDERS                       int64
THURSDAY_ORDERS                        int64
FRIDAY_ORDERS                          int64
SATURDAY_ORDERS                        int64
SUNDAY_ORDERS                          int64
MONDAY_REVENUE                       float64
TUESDAY_REVENUE                      float64
WEDNESDAY_REVENUE                    float64
THURSDAY_REVENUE                     float64
FRIDAY_REVENUE                       float64
SATURDAY_REVENUE                     float64
SUNDAY_REVENUE                       float64
WEEK1_DAY01_DAY07_ORDERS             float64
WEEK2_DAY08_DAY15_ORDERS             float64
WEEK3_DAY16_DAY23_ORDERS             float64
WEEK4_DAY24_DAY31_ORDERS             float64
WEEK1_DAY01_DAY07_REVENUE            float64
WEEK2_DAY08_DAY15_REVENUE            float64
WEEK3_DAY16_DAY23_REVENUE            float64
WEEK4_DAY24_DAY31_REVENUE            float64
TIME_0000_0600_ORDERS                float64
TIME_0601_1200_ORDERS                float64
TIME_1200_1800_ORDERS                float64
TIME_1801_2359_ORDERS                float64
TIME_0000_0600_REVENUE               float64
TIME_0601_1200_REVENUE               float64
TIME_1200_1800_REVENUE               float64
TIME_1801_2359_REVENUE               float64
dtype: object
```

In [14]:
```
sales_data.describe().T
```

Out[14]:

	count	mean	std	min	25%	
CustomerID	5000.0	40709.227800	49949.848017	1.00	1687.5000	137
TOTAL_ORDERS	5000.0	12.870400	12.679880	1.00	3.0000	
REVENUE	5000.0	1681.523840	1998.618678	38.50	315.0975	ς

AVERAGE_ORDER_VALUE	5000.0	136.537378	91.651569	10.68	83.0250	1
CARRIAGE_REVENUE	5000.0	46.036376	47.879226	0.00	9.9800	
AVERAGESHIPPING	5000.0	3.592574	2.021360	0.00	2.5000	
AVGDAYSBETWEENORDERS	5000.0	163.159618	259.699496	0.00	21.6700	
DAYSSINCELASTORDER	5000.0	87.420000	80.156513	1.00	7.0000	
MONDAY_ORDERS	5000.0	1.629000	2.236506	0.00	0.0000	
TUESDAY_ORDERS	5000.0	1.754400	2.433940	0.00	0.0000	
WEDNESDAY_ORDERS	5000.0	1.798000	2.464875	0.00	0.0000	
THURSDAY_ORDERS	5000.0	2.134000	2.468048	0.00	0.0000	
FRIDAY_ORDERS	5000.0	1.946200	2.652680	0.00	0.0000	
SATURDAY_ORDERS	5000.0	1.683400	2.449972	0.00	0.0000	
SUNDAY_ORDERS	5000.0	1.925400	2.315018	0.00	0.0000	
MONDAY_REVENUE	5000.0	215.208336	397.831999	0.00	0.0000	
TUESDAY_REVENUE	5000.0	233.510430	411.941787	0.00	0.0000	
WEDNESDAY_REVENUE	5000.0	235.689294	397.858311	0.00	0.0000	
THURSDAY_REVENUE	5000.0	265.949796	383.890024	0.00	0.0000	1
FRIDAY_REVENUE	5000.0	250.580554	400.543113	0.00	0.0000	
SATURDAY_REVENUE	5000.0	220.975468	378.892459	0.00	0.0000	
SUNDAY_REVENUE	5000.0	260.943330	406.926075	0.00	0.0000	1
WEEK1_DAY01_DAY07_ORDERS	5000.0	2.997800	3.256980	0.00	1.0000	
WEEK2_DAY08_DAY15_ORDERS	5000.0	3.062600	3.792461	0.00	0.0000	
WEEK3_DAY16_DAY23_ORDERS	5000.0	3.230000	3.921043	0.00	0.0000	
WEEK4_DAY24_DAY31_ORDERS	5000.0	3.580000	3.970384	0.00	1.0000	
WEEK1_DAY01_DAY07_REVENUE	5000.0	378.638346	515.590218	0.00	63.9900	1
WEEK2_DAY08_DAY15_REVENUE	5000.0	406.595734	619.413277	0.00	0.0000	1
WEEK3_DAY16_DAY23_REVENUE	5000.0	421.826908	643.449120	0.00	0.0000	1
WEEK4_DAY24_DAY31_REVENUE	5000.0	474.462852	617.579321	0.00	80.0000	2
TIME_0000_0600_ORDERS	5000.0	1.028800	2.174331	0.00	0.0000	
TIME_0601_1200_ORDERS	5000.0	3.746200	4.700234	0.00	1.0000	
TIME_1200_1800_ORDERS	5000.0	4.434000	5.044793	0.00	1.0000	
TIME_1801_2359_ORDERS	5000.0	3.661400	4.581894	0.00	1.0000	
TIME_0000_0600_REVENUE	5000.0	131.062636	331.289349	0.00	0.0000	
TIME_0601_1200_REVENUE	5000.0	486.863868	789.029911	0.00	35.0000	2
TIME_1200_1800_REVENUE	5000.0	584.731626	804.290026	0.00	89.9900	2

TIME_1801_2359_REVENUE	5000.0	478.865710	743.244248	0.00	1.0000	2

Data Understanding

RFM analysis

RFM stands for recency, frequency, monetary value. In business analytics, we often use this concept to divide customers into different segments, like high-value customers, medium value customers or low-value customers, and similarly many others.

Let's assume we are a company, our company name is geek, let's perform the RFM analysis on our customers

Recency: How recently has the customer made a transaction with us

Frequency: How frequent is the customer in ordering/buying some product from us

Monetary: How much does the customer spend on purchasing products from us.

In [15]:
```
# Recency

```

In [16]:
```
sales_data['CustomerID'].nunique()
```

Out[16]: 5000

In [17]:
```
RFMScores=pd.DataFrame(data =sales_data,columns=['CustomerID',"
RFMScores
```

Out[17]:

	CustomerID	DAYSSINCELASTORDER	TOTAL_ORDERS	REVENUE
0	22.0	1.0	124.0	11986.54
1	29.0	1.0	82.0	11025.96
2	83.0	1.0	43.0	7259.69
3	95.0	1.0	44.0	6992.27
4	124.0	1.0	55.0	6263.44
...	...	...	...	...
4995	173946.0	207.0	1.0	117.49
4996	173987.0	207.0	1.0	117.49
4997	174004.0	207.0	1.0	117.49
4998	174038.0	207.0	1.0	117.49
4999	200783.0	207.0	2.0	94.14

5000 rows × 4 columns

In [18]:
```
RFMScores.rename({'DAYSSINCELASTORDER':'Recency','TOTAL_ORDERS'
```

In [19]:
```
RFMScores.head()
```

Out[19]:

	CustomerID	Recency	Frequency	Monetary
0	22.0	1.0	124.0	11986.54
1	29.0	1.0	82.0	11025.96
2	83.0	1.0	43.0	7259.69
3	95.0	1.0	44.0	6992.27
4	124.0	1.0	55.0	6263.44

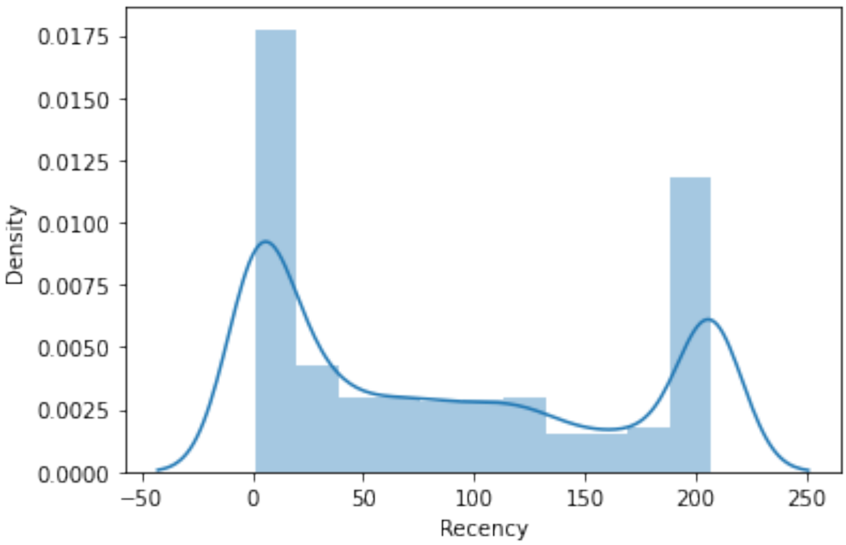
In [20]:
```
RFMScores['Frequency']=RFMScores['Frequency'].astype('int')
RFMScores['Frequency']
```

Out[20]:
```
0       124
1        82
2        43
3        44
4        55
       ...
4995      1
4996      1
4997      1
4998      1
4999      2
Name: Frequency, Length: 5000, dtype: int64
```

In [21]:
```
#Descriptive Statistics (Recency)
RFMScores.Recency.describe()
```

Out[21]:
```
count    5000.000000
mean       87.420000
std        80.156513
min         1.000000
25%         7.000000
50%        68.000000
75%       171.250000
max       207.000000
Name: Recency, dtype: float64
```

In [22]:
```
#Recency distribution plot
import seaborn as sns
x = RFMScores['Recency']

ax = sns.distplot(x)
```



In [23]:

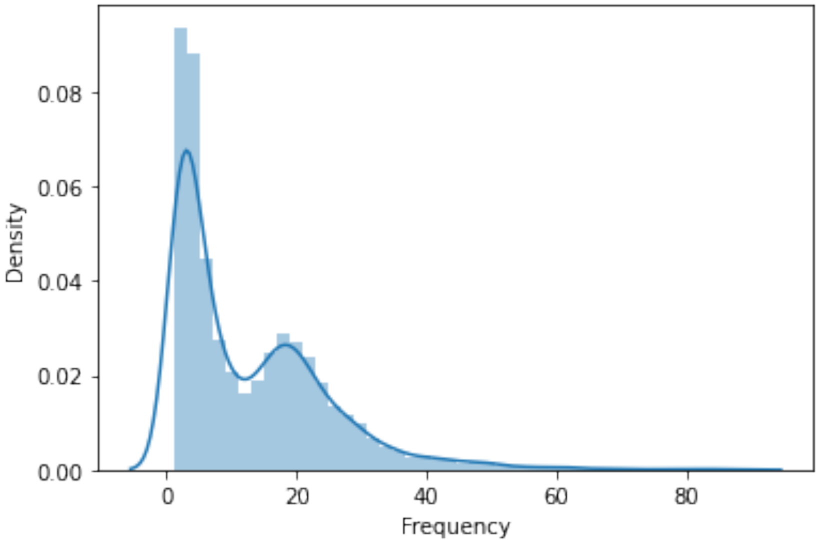
```
#Descriptive Statistics (Frequency)
RFMScores.Frequency.describe()
```

Out[23]:

```
count    5000.00000
mean       12.87040
std        12.67988
min         1.00000
25%         3.00000
50%         8.00000
75%        20.00000
max       156.00000
Name: Frequency, dtype: float64
```

In [24]:

```
#Frequency distribution plot, taking observations which have fr
import seaborn as sns
x = RFMScores.query('Frequency < 100')['Frequency']

ax = sns.distplot(x)
```



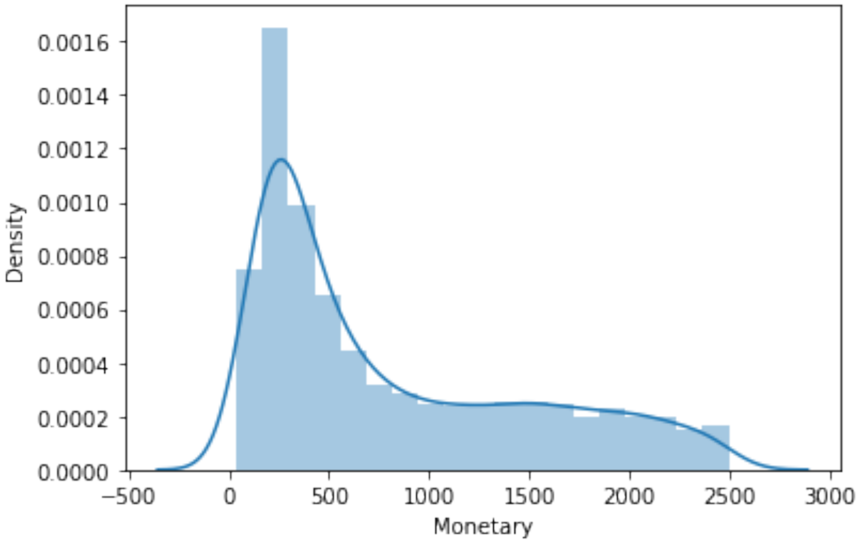
In [25]:

```
#Descriptive Statistics (Monetary)
RFMScores.Monetary.describe()
```

Out[25]:

```
count     5000.000000
mean      1681.523840
std       1998.618678
min         38.500000
25%        315.097500
50%        966.725000
75%       2493.072500
max      34847.400000
Name: Monetary, dtype: float64
```

In [26]:
```
#Monateray distribution plot, taking observations which have mo
import seaborn as sns
x = RFMScores.query('Monetary < 2500')['Monetary']

ax = sns.distplot(x)

```



In [27]:
```
#Split into four segments using quantiles
quantiles = RFMScores.drop('CustomerID',axis=1).quantile(q=[0.3
quantiles = quantiles.to_dict()
```

In [28]:
```
quantiles
```

Out[28]:
```
{'Recency': {0.33: 19.0, 0.66: 124.0, 1.0: 207.0},
 'Frequency': {0.33: 4.0, 0.66: 16.0, 1.0: 156.0},
 'Monetary': {0.33: 427.09400000000005, 0.66: 1867.2972, 1.0: 3484
7.4}}
```

In [ ]:
```

```

In [29]:

```
#Functions to create R, F and M segments
def RScoring(x,p,d):
    if x <= d[p][0.33]:
        return 1
    elif x > d[p][0.33] and x <= d[p][0.66]:
        return 2
    elif x <= d[p][1]:
        return 3

def FnMScoring(x,p,d):
    if x <= d[p][0.33]:
        return 3
    elif x > d[p][0.33] and x <= d[p][0.66]:
        return 2
    elif x <= d[p][1]:
        return 1
```

In [30]:

```
#Calculate Add R, F and M segment value columns in the existing
RFMScores['R'] = RFMScores['Recency'].apply(RScoring, args=('Re
RFMScores['F'] = RFMScores['Frequency'].apply(FnMScoring, args=
RFMScores['M'] = RFMScores['Monetary'].apply(FnMScoring, args=(
RFMScores
```

Out[30]:

	CustomerID	Recency	Frequency	Monetary	R	F	M
0	22.0	1.0	124	11986.54	1	1	1
1	29.0	1.0	82	11025.96	1	1	1
2	83.0	1.0	43	7259.69	1	1	1
3	95.0	1.0	44	6992.27	1	1	1
4	124.0	1.0	55	6263.44	1	1	1
...	...	...	...	...	...	...	...
4995	173946.0	207.0	1	117.49	3	3	3
4996	173987.0	207.0	1	117.49	3	3	3
4997	174004.0	207.0	1	117.49	3	3	3
4998	174038.0	207.0	1	117.49	3	3	3
4999	200783.0	207.0	2	94.14	3	3	3

5000 rows × 7 columns

In [31]:

```
#Calculate and Add RFMGroup value column showing combined conca
RFMScores['RFMGroup'] = RFMScores.R.map(str) + RFMScores.F.map(

#Calculate and Add RFMScore value column showing total sum of R
RFMScores['RFMScore'] = RFMScores[['R', 'F', 'M']].sum(axis = 1
RFMScores
```

Out[31]:

	CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore
0	22.0	1.0	124	11986.54	1	1	1	111	3
1	29.0	1.0	82	11025.96	1	1	1	111	3
2	83.0	1.0	43	7259.69	1	1	1	111	3
3	95.0	1.0	44	6992.27	1	1	1	111	3
4	124.0	1.0	55	6263.44	1	1	1	111	3
...	...	...	...	...	...	...	...	...	...
4995	173946.0	207.0	1	117.49	3	3	3	333	9
4996	173987.0	207.0	1	117.49	3	3	3	333	9
4997	174004.0	207.0	1	117.49	3	3	3	333	9
4998	174038.0	207.0	1	117.49	3	3	3	333	9
4999	200783.0	207.0	2	94.14	3	3	3	333	9

5000 rows × 9 columns

In [32]:

```
#Assign Loyalty Level to each customer
Loyalty_Level = ['champions', 'Potential customers', 'need atte
Score_cuts = pd.qcut(RFMScores.RFMScore, q =3, labels = Loyalty
RFMScores['RFM_Loyalty_Level'] = Score_cuts.values
RFMScores.reset_index().tail()
```

Out[32]:

	index	CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore
4995	4995	173946.0	207.0	1	117.49	3	3	3	333	9
4996	4996	173987.0	207.0	1	117.49	3	3	3	333	9
4997	4997	174004.0	207.0	1	117.49	3	3	3	333	9
4998	4998	174038.0	207.0	1	117.49	3	3	3	333	9
4999	4999	200783.0	207.0	2	94.14	3	3	3	333	9

In [33]:
```
#Validate the data for RFMGroup = 111
RFMScores[RFMScores['RFMGroup']=='111'].sort_values('Monetary',
```

Out[33]:

	index	CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore
0	1153	4.0	5.0	84	18554.49	1	1	1	111	3
1	1458	10.0	13.0	47	14309.92	1	1	1	111	3
2	1567	12.0	16.0	79	13775.96	1	1	1	111	3
3	1498	15.0	14.0	50	13180.85	1	1	1	111	3
4	1194	17.0	6.0	36	12969.98	1	1	1	111	3
...	...	...	...	...	...	...	...	...	...	...
419	1441	3316.0	12.0	19	1879.57	1	1	1	111	3
420	66	3335.0	1.0	19	1876.65	1	1	1	111	3
421	1365	3339.0	10.0	26	1875.33	1	1	1	111	3
422	1442	3363.0	12.0	23	1868.53	1	1	1	111	3
423	67	3365.0	1.0	21	1868.29	1	1	1	111	3

424 rows × 11 columns

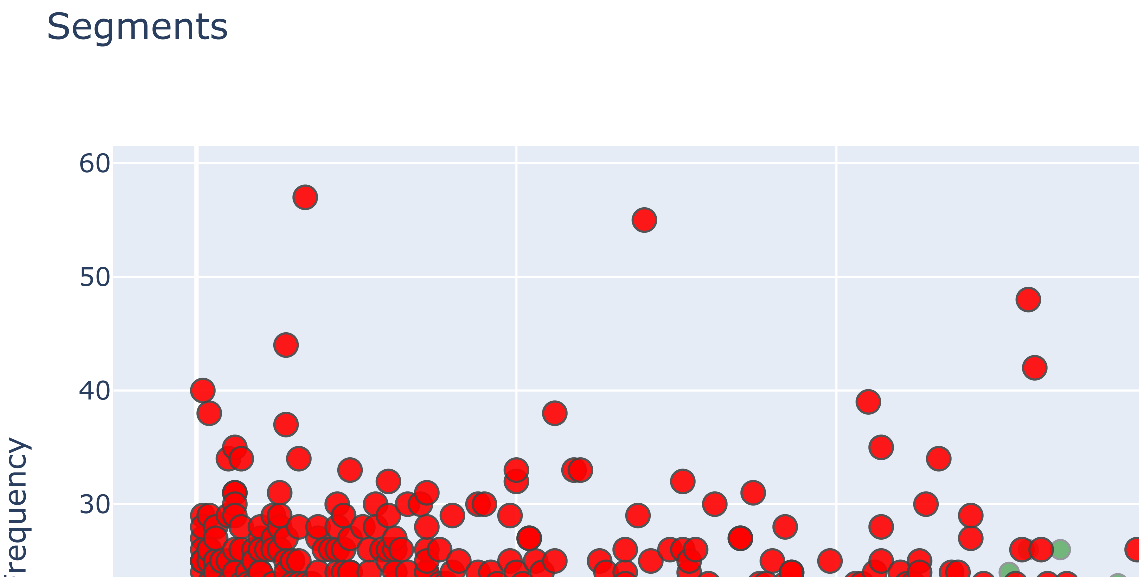
In [ ]:
```

```

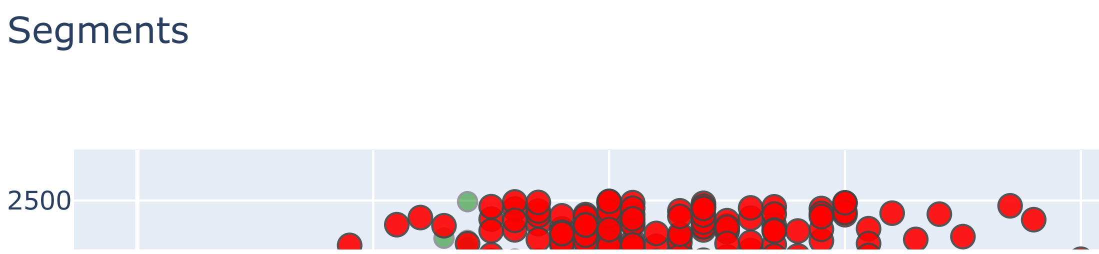
In [34]:
```
import chart_studio as cs
import plotly.offline as po
import plotly.graph_objs as gobj

#Recency Vs Frequency
graph = RFMScores.query("Monetary < 2500 and Frequency < 100 ")

plot_data = [
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'need attention'")
        y=graph.query("RFM_Loyalty_Level == 'need attention'")
        mode='markers',
        name='need attention',
        marker= dict(size= 7,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.8
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Potential custome
        y=graph.query("RFM_Loyalty_Level == 'Potential custome
        mode='markers',
        name='Potential customers',
        marker= dict(size= 9,
```

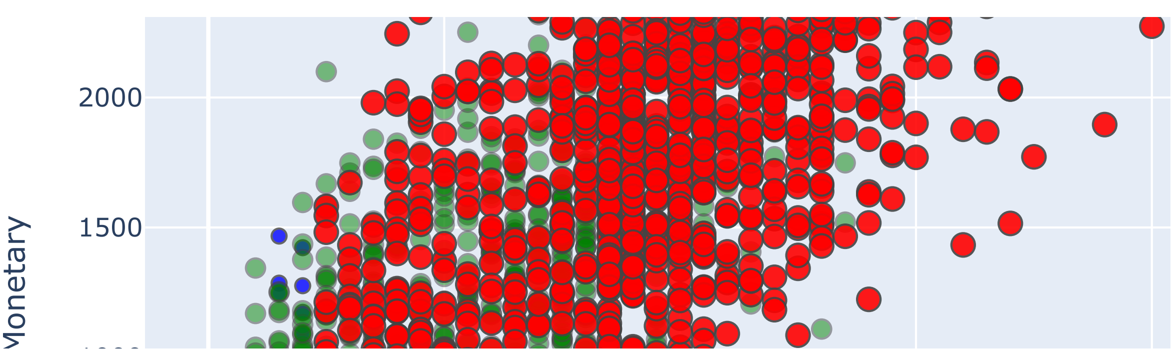
```
            line= dict(width=1),
            color= 'green',
            opacity= 0.5
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'champions'")['Rece
        y=graph.query("RFM_Loyalty_Level == 'champions'")['Freq
        mode='markers',
        name='champions',
        marker= dict(size= 11,
            line= dict(width=1),
            color= 'red',
            opacity= 0.9
           )
    ),
]

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

#Frequency Vs Monetary
graph = RFMScores.query("Monetary < 2500 and Frequency < 100")

plot_data = [
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'need attention'")
        y=graph.query("RFM_Loyalty_Level == 'need attention'")
        mode='markers',
        name='need attention',
        marker= dict(size= 7,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.8
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Potential custome
        y=graph.query("RFM_Loyalty_Level == 'Potential custome
        mode='markers',
        name='Potential customers',
        marker= dict(size= 9,
            line= dict(width=1),
            color= 'green',
            opacity= 0.5
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'champions'")['Freq
```

```
        x=graph.query("RFM_Loyalty_Level == 'champions'")['Fre
        y=graph.query("RFM_Loyalty_Level == 'champions'")['Mone
        mode='markers',
        name='champions',
        marker= dict(size= 11,
            line= dict(width=1),
            color= 'red',
            opacity= 0.9
           )
    ),
]

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

#Recency Vs Monetary
graph = RFMScores.query("Monetary < 50000 and Frequency < 2000'

plot_data = [
    gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'need attention'")
        y=graph.query("RFM_Loyalty_Level == 'need attention'")
        mode='markers',
        name='need attention',
        marker= dict(size= 7,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.8
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'Potential custome
        y=graph.query("RFM_Loyalty_Level == 'Potential custome
        mode='markers',
        name='Potential customers',
        marker= dict(size= 9,
            line= dict(width=1),
            color= 'green',
            opacity= 0.5
           )
    ),
        gobj.Scatter(
        x=graph.query("RFM_Loyalty_Level == 'champions'")['Rece
        y=graph.query("RFM_Loyalty_Level == 'champions'")['Mone
        mode='markers',
        name='champions',
        marker= dict(size= 11,
            line= dict(width=1),
            color= 'red',
            opacity= 0.9
```

```
            )
    ),
]

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





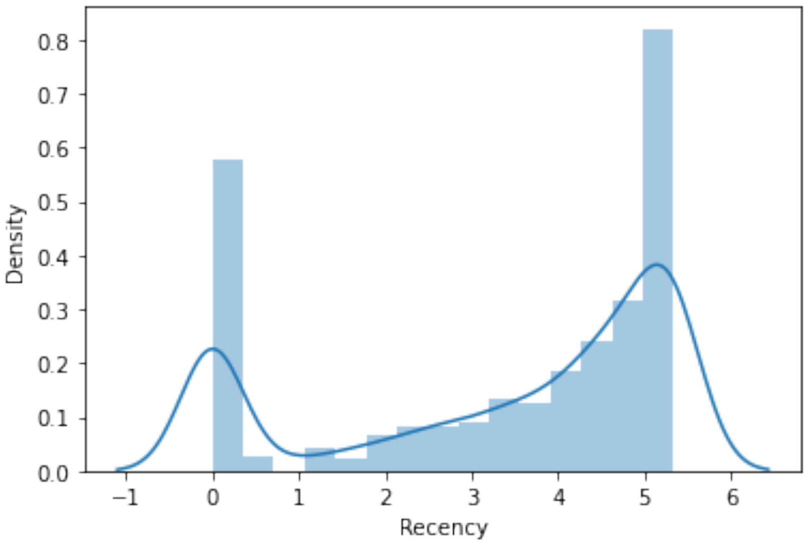


Segments

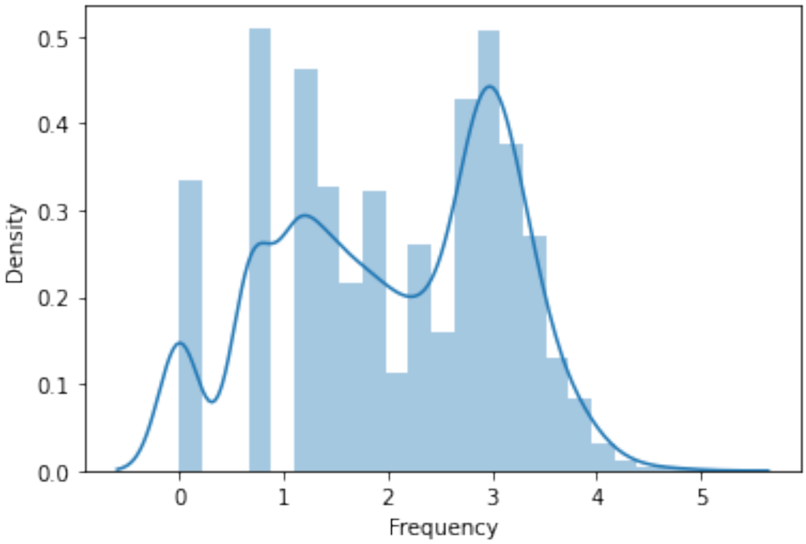
K-Means Clustering

In [35]:
```
#Handle negative and zero values so as to handle infinite numbe
def handle_neg_n_zero(num):
    if num <= 0:
        return 1
    else:
        return num
#Apply handle_neg_n_zero function to Recency and Monetary colum
RFMScores['Recency'] = [handle_neg_n_zero(x) for x in RFMScores
RFMScores['Monetary'] = [handle_neg_n_zero(x) for x in RFMScore

#Perform Log transformation to bring data into normal or near n
Log_Tfd_Data = RFMScores[['Recency', 'Frequency', 'Monetary']].
```

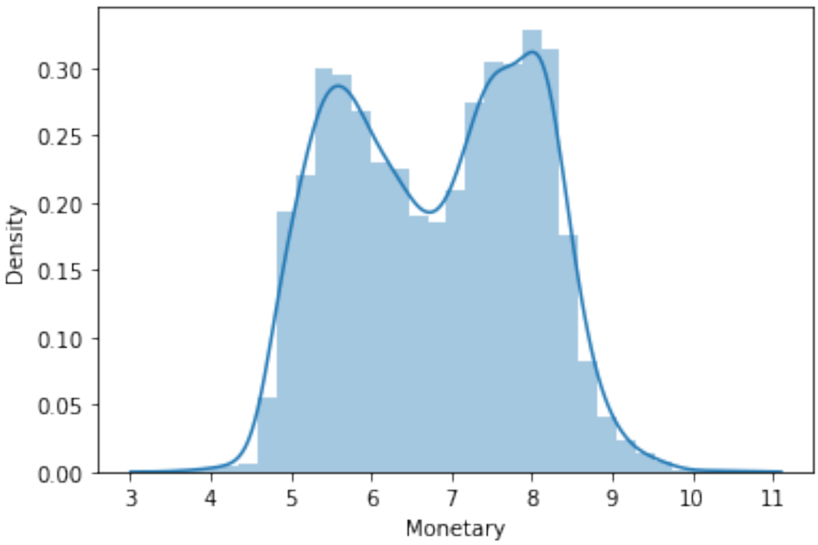
In [36]:
```
#Data distribution after data normalization for Recency
Recency_Plot = Log_Tfd_Data['Recency']
ax = sns.distplot(Recency_Plot)
```



In [37]:
```
#Data distribution after data normalization for Frequency
Frequency_Plot = Log_Tfd_Data.query('Frequency <100')['Frequenc
ax = sns.distplot(Frequency_Plot)
```



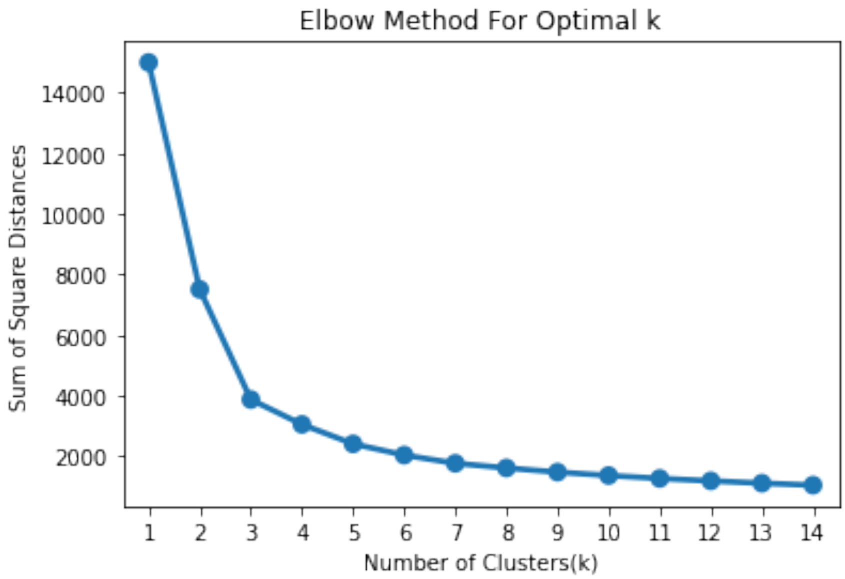
In [38]:
```
#Data distribution after data normalization for Monetary
Monetary_Plot = Log_Tfd_Data.query('Monetary < 2500')['Monetary
ax = sns.distplot(Monetary_Plot)
```



In [39]:
```
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 = RFMScores.index
```

In [40]:

```
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= 100
    km = km.fit(Scaled_Data)
    sum_of_sq_dist[k] = km.inertia_

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



In [41]:

```
#Perform K-Mean Clustering or build the K-Means clustering mode
KMean_clust = KMeans(n_clusters= 3, init= 'k-means++', max_iter
KMean_clust.fit(Scaled_Data)

#Find the clusters for the observation given in the dataset
RFMScores['Cluster'] = KMean_clust.labels_
RFMScores
```

Out[41]:

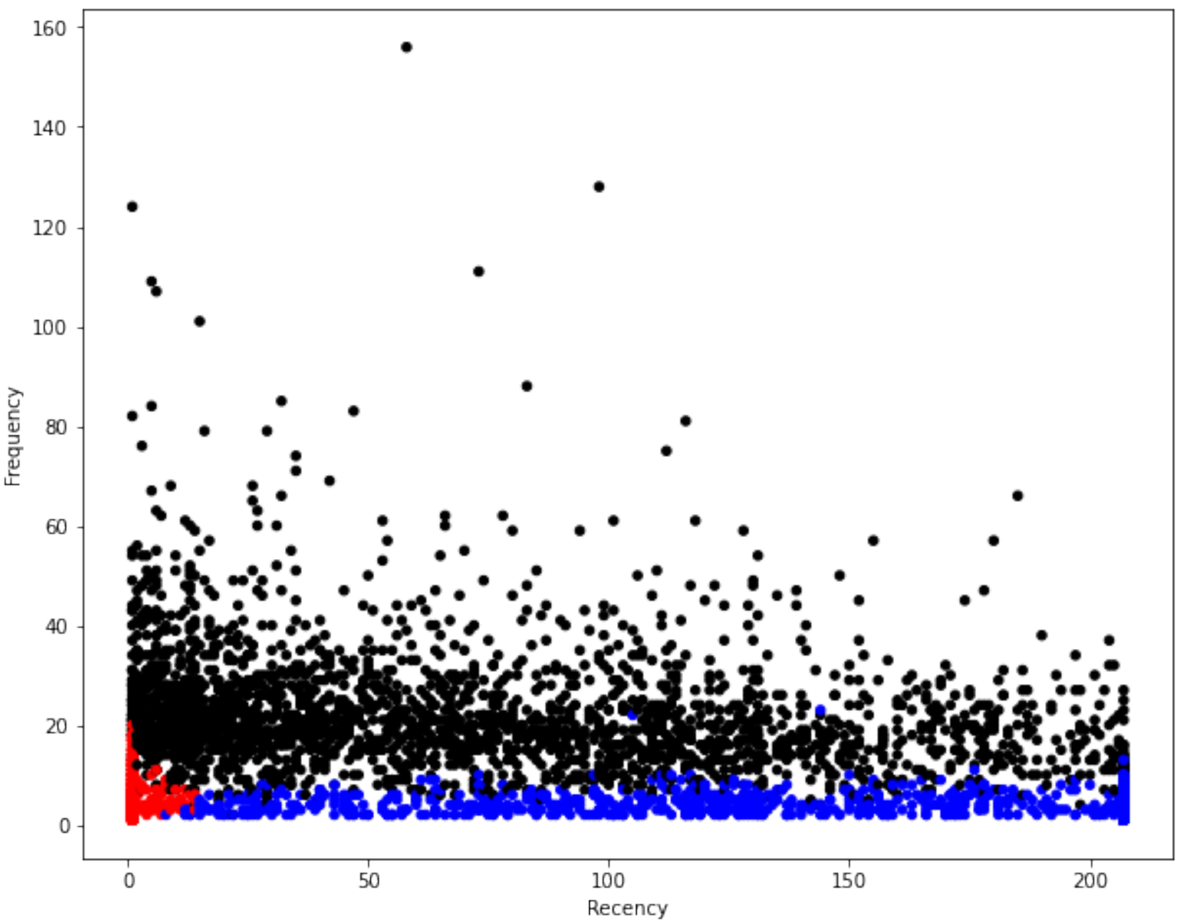
	CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore	RFM
0	22.0	1.0	124	11986.54	1	1	1	111	3	
1	29.0	1.0	82	11025.96	1	1	1	111	3	
2	83.0	1.0	43	7259.69	1	1	1	111	3	
3	95.0	1.0	44	6992.27	1	1	1	111	3	
4	124.0	1.0	55	6263.44	1	1	1	111	3	
...	...	...	...	...	...	...	...	...	...	
4995	173946.0	207.0	1	117.49	3	3	3	333	9	
4996	173987.0	207.0	1	117.49	3	3	3	333	9	
4997	174004.0	207.0	1	117.49	3	3	3	333	9	
4998	174038.0	207.0	1	117.49	3	3	3	333	9	
4999	200783.0	207.0	2	94.14	3	3	3	333	9	

5000 rows × 11 columns

In [42]:

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

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

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



In [43]:
```
RFMScores
```

Out[43]:

	CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore	RFM_
0	22.0	1.0	124	11986.54	1	1	1	111	3	
1	29.0	1.0	82	11025.96	1	1	1	111	3	
2	83.0	1.0	43	7259.69	1	1	1	111	3	
3	95.0	1.0	44	6992.27	1	1	1	111	3	
4	124.0	1.0	55	6263.44	1	1	1	111	3	
...	...	...	...	...	...	...	...	...	...	
4995	173946.0	207.0	1	117.49	3	3	3	333	9	
4996	173987.0	207.0	1	117.49	3	3	3	333	9	
4997	174004.0	207.0	1	117.49	3	3	3	333	9	
4998	174038.0	207.0	1	117.49	3	3	3	333	9	
4999	200783.0	207.0	2	94.14	3	3	3	333	9	

5000 rows × 12 columns

In [ ]:
```

```