Capstone_project

December 14, 2022

[1]:
```
# import important libraries

import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.colors as mcolors


import warnings
warnings.filterwarnings('ignore')

import datetime as dt

from operator import attrgetter

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

[2]:
```
# Read excel file
df=pd.read_excel('Online Retail.xlsx')
```

[3]:
```
df.head()
```

[3]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice  CustomerID         Country
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
```

```
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
```

[4]:
```
# Directly delete the duplicates as they are not in the scope of our topic
df.drop_duplicates(inplace=True)
```

[5]:
```
#checking null values
df.isnull().sum()
```

[5]:
```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135037
Country             0
dtype: int64
```

[6]:
```
#Deleting null values - since we are here to seggregate customer and if␣
↪information of customer is missing thn we dont need tht customer hence␣
↪delete these rows

df=df.dropna()
df.isnull().sum()
```

[6]:
```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

[7]:
```
# Descriptive Analysis
df.describe()
```

[7]:

	Quantity	UnitPrice	CustomerID
count	401604.000000	401604.000000	401604.000000
mean	12.183273	3.474064	15281.160818
std	250.283037	69.764035	1714.006089
min	-80995.000000	0.000000	12346.000000
25%	2.000000	1.250000	13939.000000
50%	5.000000	1.950000	15145.000000
75%	12.000000	3.750000	16784.000000
max	80995.000000	38970.000000	18287.000000

There some negative values in the Quantity and UnitPrice features. It can't be possible. So I will filter the data greater than zero.

[8]:
```
df = df[(df['Quantity'] > 0) & (df['UnitPrice'] > 0)]
```

0.1 Data Preparation

For cohort analysis, we need three labels. These are cohort,order month,period number or payment period, cohort group and cohort period/index.

[9]:
```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], format='%m/%d/%Y %H:%M')␣
↪#convert string date field to datetime
```

Now, we need to create the cohort and order_month variables. The first one indicates the monthly cohort based on the first purchase date and the second one is the truncated month of the purchase or Invoice date.

[10]:
```
df['cohort'] = df.groupby('CustomerID')['InvoiceDate'].transform('min').dt.
↪to_period('M')
df['order_month'] = df['InvoiceDate'].dt.to_period('M')  # dt.month can also be␣
↪use
```

[11]:
```
df.head()
```

[11]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice  CustomerID         Country   cohort  \
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom  2010-12
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom  2010-12
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom  2010-12
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom  2010-12
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom  2010-12

  order_month
0     2010-12
1     2010-12
2     2010-12
3     2010-12
4     2010-12
```

Then, we aggregate the data per cohort and order_month and count the number of unique customers in each group.

[12]:
```python
df_cohort = df.groupby(['cohort', 'order_month']).agg(n_customers=('CustomerID', 'nunique')).reset_index(drop=False)
```

[13]:
```python
df_cohort['period_number'] = (df_cohort['order_month'] - df_cohort['cohort']).apply(attrgetter('n'))
```

[14]:
```python
df_cohort.head()
```

[14]:
```
    cohort order_month  n_customers  period_number
0  2010-12     2010-12          885              0
1  2010-12     2011-01          324              1
2  2010-12     2011-02          286              2
3  2010-12     2011-03          340              3
4  2010-12     2011-04          321              4
```

Then, we aggregate the data per cohort and order_month and count the number of unique customers in each group.

[15]:
```python
cohort_pivot = df_cohort.pivot_table(index='cohort', columns='period_number', values='n_customers')
```

[16]:
```python
cohort_pivot
```

[16]:
```
period_number     0      1      2      3      4      5      6      7      8   \
cohort
2010-12        885.0  324.0  286.0  340.0  321.0  352.0  321.0  309.0  313.0
2011-01        417.0   92.0  111.0   96.0  134.0  120.0  103.0  101.0  125.0
2011-02        380.0   71.0   71.0  108.0  103.0   94.0   96.0  106.0   94.0
2011-03        452.0   68.0  114.0   90.0  101.0   76.0  121.0  104.0  126.0
2011-04        300.0   64.0   61.0   63.0   59.0   68.0   65.0   78.0   22.0
2011-05        284.0   54.0   49.0   49.0   59.0   66.0   75.0   27.0    NaN
2011-06        242.0   42.0   38.0   64.0   56.0   81.0   23.0    NaN    NaN
2011-07        188.0   34.0   39.0   42.0   51.0   21.0    NaN    NaN    NaN
2011-08        169.0   35.0   42.0   41.0   21.0    NaN    NaN    NaN    NaN
2011-09        299.0   70.0   90.0   34.0    NaN    NaN    NaN    NaN    NaN
2011-10        358.0   86.0   41.0    NaN    NaN    NaN    NaN    NaN    NaN
2011-11        323.0   36.0    NaN    NaN    NaN    NaN    NaN    NaN    NaN
2011-12         41.0    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN

period_number     9      10     11     12
cohort
2010-12        350.0  331.0  445.0  235.0
2011-01        136.0  152.0   49.0    NaN
2011-02        116.0   26.0    NaN    NaN
2011-03         39.0    NaN    NaN    NaN
2011-04          NaN    NaN    NaN    NaN
2011-05          NaN    NaN    NaN    NaN
```

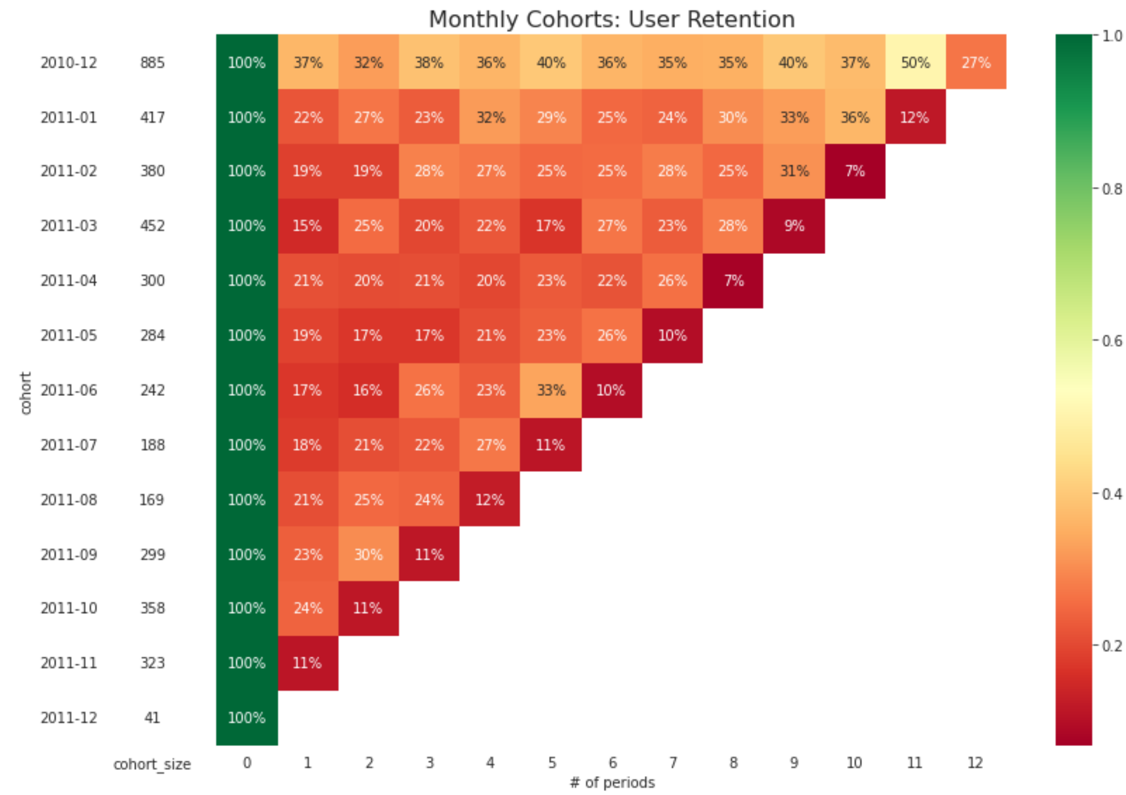
```
2011-06          NaN    NaN    NaN    NaN
2011-07          NaN    NaN    NaN    NaN
2011-08          NaN    NaN    NaN    NaN
2011-09          NaN    NaN    NaN    NaN
2011-10          NaN    NaN    NaN    NaN
2011-11          NaN    NaN    NaN    NaN
2011-12          NaN    NaN    NaN    NaN
```

[17]:
```python
cohort_size = cohort_pivot.iloc[:, 0]
```

[18]:
```python
retention_matrix = cohort_pivot.divide(cohort_size, axis=0)
```

Lastly, we plot the retention matrix as a heatmap. Additionally, we wanted to include extra information regarding the cohort size. That is why we in fact created two heatmaps, where the one indicating the cohort size is using a white only colormap — no coloring at all.

[19]:
```python
with sns.axes_style("white"):
    fig, ax = plt.subplots(1, 2, figsize=(12, 8), sharey=True,␣
 ↪gridspec_kw={'width_ratios': [1, 11]})

    # retention matrix
    sns.heatmap(retention_matrix,
                mask=retention_matrix.isnull(),
                annot=True,
                fmt='.0%',
                cmap='RdYlGn',
                ax=ax[1])
    ax[1].set_title('Monthly Cohorts: User Retention', fontsize=16)
    ax[1].set(xlabel='# of periods',
              ylabel='')

     # cohort size
    cohort_size_df = pd.DataFrame(cohort_size).rename(columns={0:␣
 ↪'cohort_size'})
    white_cmap = mcolors.ListedColormap(['white'])
    sns.heatmap(cohort_size_df,
                annot=True,
                cbar=False,
                fmt='g',
                cmap=white_cmap,
                ax=ax[0])

    fig.tight_layout()
```



In the image, we can see that there is a sharp drop-off in the second month (indexed as 1) already, on average around 80% of customers do not make any purchase in the second month. The first cohort (2010–12) seems to be an exception and performs surprisingly well as compared to the other ones. A year after the first purchase, there is a 50% retention. This might be a cohort of dedicated customers, who first joined the platform based on some already-existing connections with the retailer. However, from data alone, that is very hard to accurately explain.

Throughout the matrix, we can see fluctuations in retention over time. This might be caused by the characteristics of the business, where clients do periodic purchases, followed by periods of inactivity.

0.2 Prepare data for modelling

. RECENCY= No. of days since last purchase

. FREQUENCY= No. of transaction

. MONETARY= Total amount of transaction (i.e. Revenue)

[20]:
```
# Create column indicating revenue per customer

df['Revenue']=df['Quantity']*df['UnitPrice']
```

```
df.head()
```

[20]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice  CustomerID         Country   cohort  \
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom  2010-12
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom  2010-12
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom  2010-12
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom  2010-12
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom  2010-12

  order_month  Revenue
0     2010-12    15.30
1     2010-12    20.34
2     2010-12    22.00
3     2010-12    20.34
4     2010-12    20.34
```

[21]:
```
revenue_contribution=df.groupby('CustomerID')['Revenue'].sum()
revenue_contribution=revenue_contribution.reset_index()
revenue_contribution.head()
```

[21]:
```
   CustomerID   Revenue
0     12346.0  77183.60
1     12347.0   4310.00
2     12348.0   1797.24
3     12349.0   1757.55
4     12350.0    334.40
```

[22]:
```
Frequency=df.groupby('CustomerID')['InvoiceNo'].count()
Frequency=Frequency.reset_index()
Frequency.head()
```

[22]:
```
   CustomerID  InvoiceNo
0     12346.0          1
1     12347.0        182
2     12348.0         31
3     12349.0         73
4     12350.0         17
```

[23]:
```
df['duration']=df['InvoiceDate'].max()-df['InvoiceDate']
df['duration'].head()
```

[23]:
```
0   373 days 04:24:00
1   373 days 04:24:00
2   373 days 04:24:00
3   373 days 04:24:00
4   373 days 04:24:00
Name: duration, dtype: timedelta64[ns]
```

[24]:
```
recency=df.groupby('CustomerID')['duration'].min()
recency=recency.reset_index()
recency.head()
```

[24]:

	CustomerID	duration
0	12346.0	325 days 02:49:00
1	12347.0	1 days 20:58:00
2	12348.0	74 days 23:37:00
3	12349.0	18 days 02:59:00
4	12350.0	309 days 20:49:00

[25]:
```
# Merging data to Calculate RFM metrics.

rf=pd.merge(recency,Frequency, on='CustomerID', how='inner')
rfm=pd.merge(rf,revenue_contribution, on='CustomerID', how='inner')
rfm.columns=['CustomerID','Recency','Frequency','Monetary']
rfm['Recency']=rfm['Recency'].dt.days
rfm.head()
```

[25]:

	CustomerID	Recency	Frequency	Monetary
0	12346.0	325	1	77183.60
1	12347.0	1	182	4310.00
2	12348.0	74	31	1797.24
3	12349.0	18	73	1757.55
4	12350.0	309	17	334.40

[26]:
```
# Split customers into four segments using Quantiles
#Build RFM Segments. Give recency, frequency, and monetary scores individually␣
↪by dividing them into quartiles.

quantiles=rfm.quantile(q=[0.25,0.50,0.75])
quantiles=quantiles.to_dict()
quantiles
```

[26]:
```
{'CustomerID': {0.25: 13813.25, 0.5: 15299.5, 0.75: 16778.75},
 'Recency': {0.25: 17.0, 0.5: 50.0, 0.75: 141.0},
 'Frequency': {0.25: 17.0, 0.5: 41.0, 0.75: 98.0},
 'Monetary': {0.25: 306.48249999999996,
  0.5: 668.5699999999999,
  0.75: 1660.5975000000012}}
```

[27]:
```
#function to create R,F,M segments

def R_score(x,p,d):
    if x<=d[p][0.25]:
        return 1
    elif x<=d[p][0.50]:
        return 2
    elif x<=d[p][0.75]:
        return 3
    else:
        return 4

def FnM_score(x,p,d):
    if x<=d[p][0.25]:
        return 4
    elif x<=d[p][0.50]:
        return 3
    elif x<=d[p][0.75]:
        return 2
    else:
        return 1
```

[28]:
```
# Calculate and add R,F,M segment value columns in existing dataset to show␣
↪R,F,M values

rfm['R']=rfm['Recency'].apply(R_score, args=('Recency',quantiles))
rfm['F']=rfm['Frequency'].apply(FnM_score, args=('Frequency',quantiles))
rfm['M']=rfm['Monetary'].apply(FnM_score, args=('Monetary',quantiles))
rfm.head()
```

[28]:

	CustomerID	Recency	Frequency	Monetary	R	F	M
0	12346.0	325	1	77183.60	4	4	1
1	12347.0	1	182	4310.00	1	1	1
2	12348.0	74	31	1797.24	3	3	1
3	12349.0	18	73	1757.55	2	2	1
4	12350.0	309	17	334.40	4	4	3

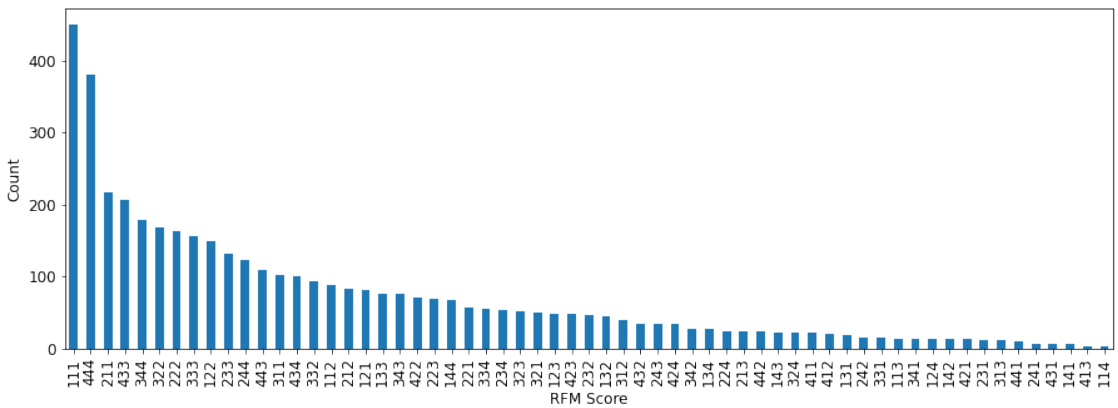
[29]:
```
#calculate and add RFM_group value column showing combined concatenated score␣
↪of RFM
#Combine three ratings to get a RFM segment (as strings).


rfm['RFM_group']=rfm ['R'].map(str)+rfm ['F'].map(str)+rfm ['M'].map(str) #␣
↪map(str) can be replaced with astype(str)

#calculate and add RFM_score value column showing total sum of RFM_group values
#Get the RFM score by adding up the three ratings.
```

```
rfm['RFM_score']=rfm[['R','F','M']].sum(axis =1)
rfm.head()
```

[29]:

	CustomerID	Recency	Frequency	Monetary	R	F	M	RFM_group	RFM_score
0	12346.0	325	1	77183.60	4	4	1	441	9
1	12347.0	1	182	4310.00	1	1	1	111	3
2	12348.0	74	31	1797.24	3	3	1	331	7
3	12349.0	18	73	1757.55	2	2	1	221	5
4	12350.0	309	17	334.40	4	4	3	443	11

[30]:
```
ax = rfm['RFM_group'].value_counts().plot(kind='bar', figsize=(15, 5),␣
↪fontsize=12)
ax.set_xlabel("RFM Score", fontsize=12)
ax.set_ylabel("Count", fontsize=12)
plt.show()
```

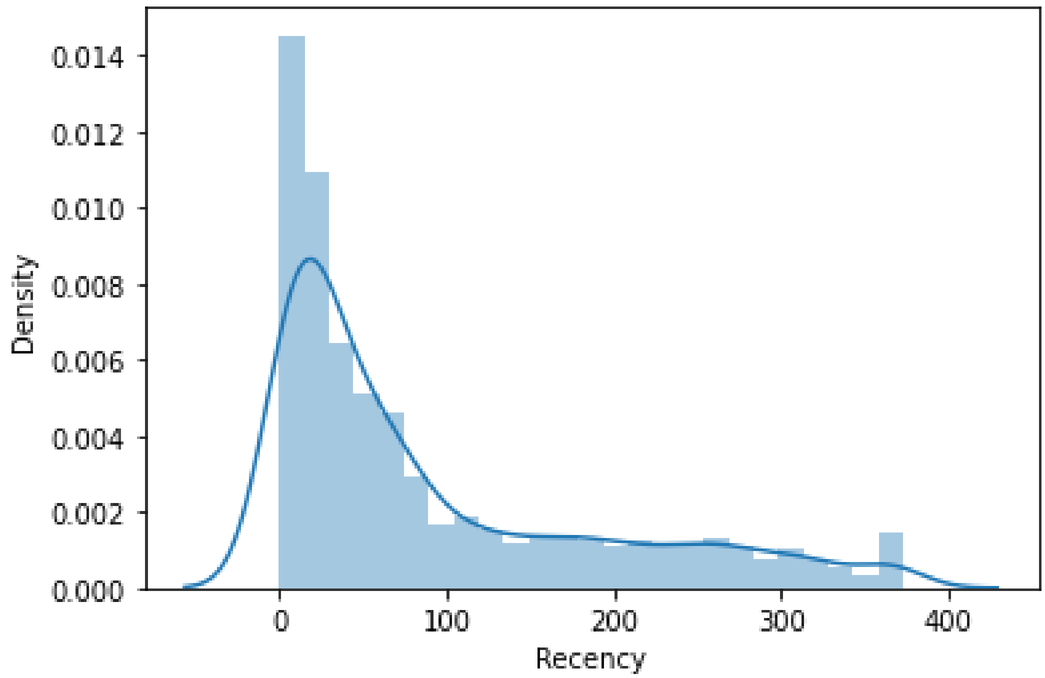


We can see that our largest segment is made up of our most valuable customers. However, our next largest segment is made up of our least valuable customers.

0.3 Check for skewness of graph

[31]:
```
sns.distplot(rfm['Recency'])
```

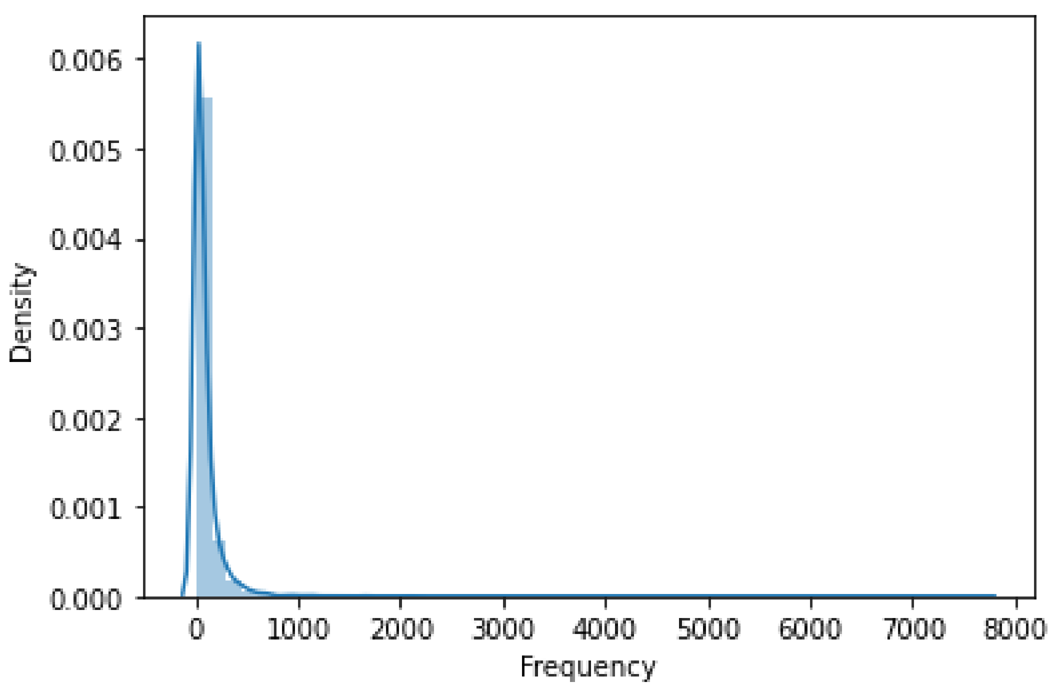
[31]: <AxesSubplot:xlabel='Recency', ylabel='Density'>



It shows tht graph is right-skewed.

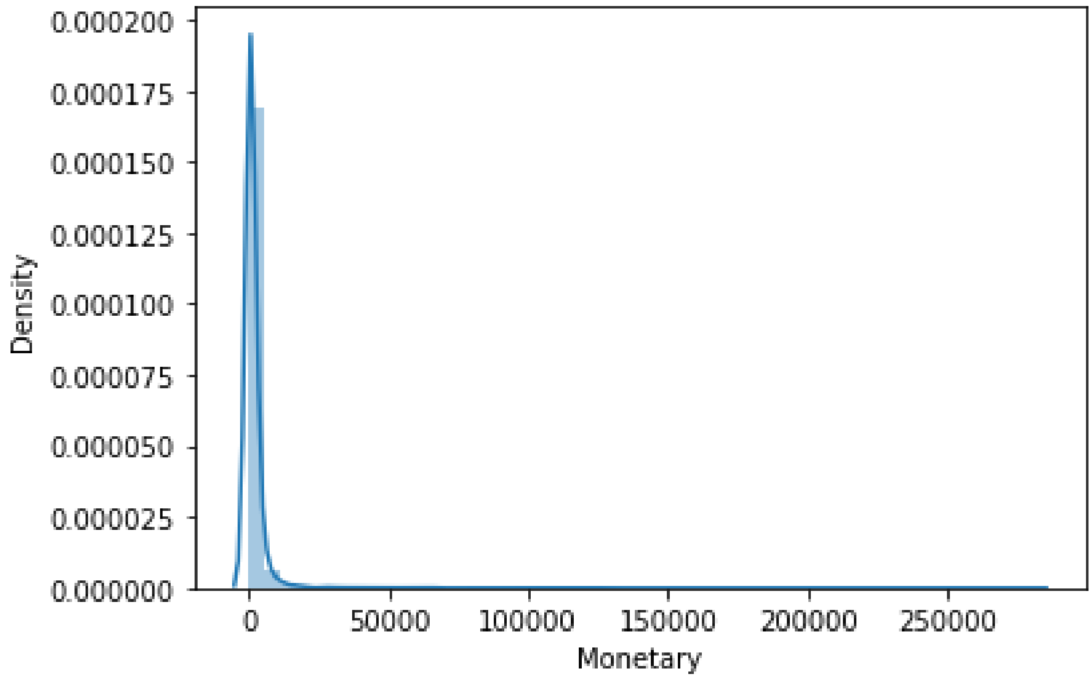
[32]:
```
sns.distplot(rfm['Frequency'])
```

[32]: <AxesSubplot:xlabel='Frequency', ylabel='Density'>



[33]:
```
sns.distplot(rfm['Monetary'])
```

[33]: <AxesSubplot:xlabel='Monetary', ylabel='Density'>



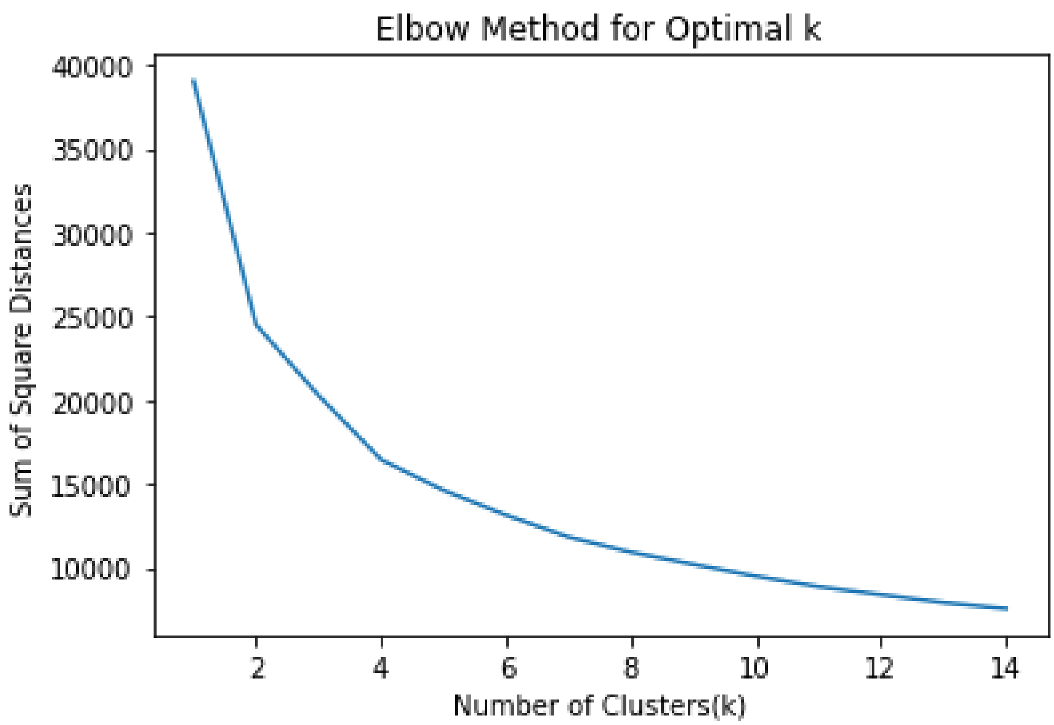
0.4 Scaling of data

[34]:
```
# Bring the data on the same scale
scaleobj = StandardScaler()
scaled_data = scaleobj.fit_transform(rfm)

#Transform it back to dataframe
scaled_data = pd.DataFrame(scaled_data, index=rfm.index,columns=rfm.columns)
```

0.5 Elbow Method for finding optimal no. of clusters

[35]:
```
sum_of_sq_dist=[]
for k in range(1,15):
    km=KMeans(n_clusters=k, init="k-means++", max_iter=1000,random_state=0)
    km=km.fit(scaled_data)
    sum_of_sq_dist.append(km.inertia_)
```

```
# Plot the graph for the sum of square distance values and number of Clusters
plt.plot(range(1,15),sum_of_sq_dist)
plt.xlabel('Number of Clusters(k)')
plt.ylabel('Sum of Square Distances')
plt.title('Elbow Method for Optimal k')
plt.show()
```



we can see from the elbow plot that your optimum cluster value is 4 , so let's build our model on that.

0.6 K-Means Clustering

[47]:
```
# Perform K-Means clustering
kmean_clust = KMeans(n_clusters=4, init="k-means++",n_init=10, max_iter=1000,␣
↪random_state=48)
kmean_clust.fit(scaled_data)
```

[47]:
```
KMeans(max_iter=1000, n_clusters=4, random_state=48)
```

[54]:
```
labels=kmean_clust.labels_
centroids=kmean_clust.cluster_centers_
```

```python
print(labels)
```

```
[3 0 1 … 1 0 0]
```

[49]:
```python
# find the clusters of observation given in data set
rfm['Cluster'] = labels
rfm.head()
```

[49]:
```
   CustomerID  Recency  Frequency  Monetary  R  F  M RFM_group  RFM_score  \
0     12346.0      325          1  77183.60  4  4  1       441          9
1     12347.0        1        182   4310.00  1  1  1       111          3
2     12348.0       74         31   1797.24  3  3  1       331          7
3     12349.0       18         73   1757.55  2  2  1       221          5
4     12350.0      309         17    334.40  4  4  3       443         11

   Cluster  Colors
0        3  Yellow
1        0     Red
2        1    Blue
3        0     Red
4        3  Yellow
```

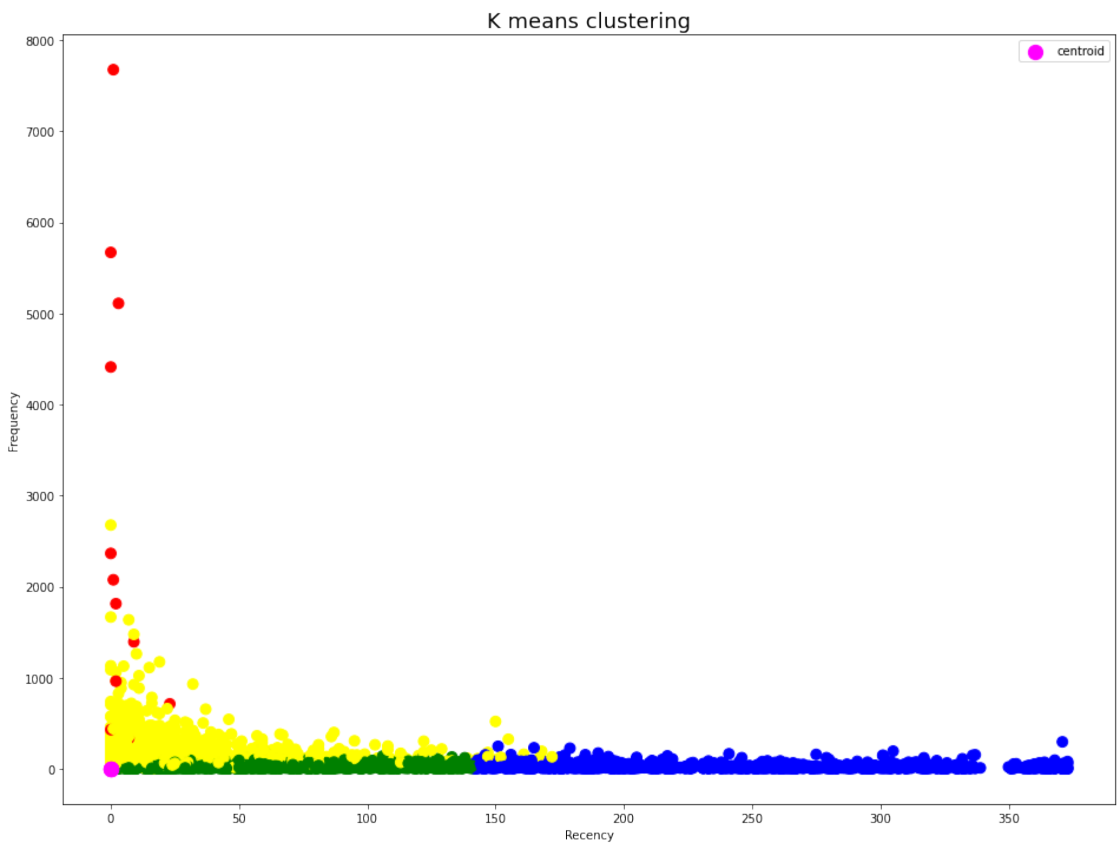
[50]:
```python
# Mapping of clusters with different colors
colors=['Yellow','Green','Red','Blue']
rfm['Colors']=rfm['Cluster'].map(lambda x:colors[x])
rfm.head()
```

[50]:
```
   CustomerID  Recency  Frequency  Monetary  R  F  M RFM_group  RFM_score  \
0     12346.0      325          1  77183.60  4  4  1       441          9
1     12347.0        1        182   4310.00  1  1  1       111          3
2     12348.0       74         31   1797.24  3  3  1       331          7
3     12349.0       18         73   1757.55  2  2  1       221          5
4     12350.0      309         17    334.40  4  4  3       443         11

   Cluster  Colors
0        3    Blue
1        0  Yellow
2        1   Green
3        0  Yellow
4        3    Blue
```

0.7 2D Scatter Plot

[55]:
```python
# Scatter plot for Frequency v/s Recency

rfm.plot(kind='scatter' ,y='Frequency', x='Recency',figsize=(16,12),s=80,␣
↪c=rfm['Colors'])
plt.scatter(centroids[:,0],centroids[:,1],s=150,c='magenta',label='centroid')
plt.title('K means clustering',fontsize=18)
plt.legend()
```

[55]: <matplotlib.legend.Legend at 0x7f2185beff90>

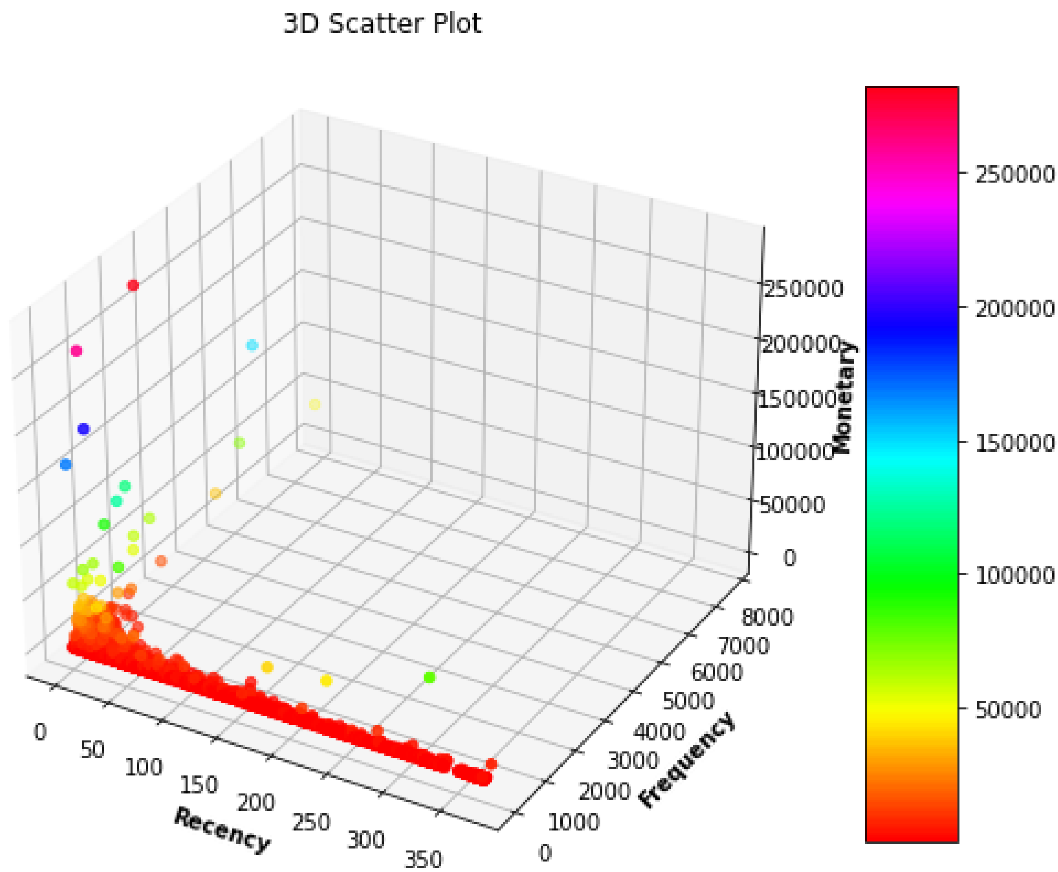


0.8 3D Scatter Plot

[82]:
```python
import plotly.express as px
fig=px.scatter_3d(rfm, x='Recency',
                  y='Frequency',
                  z='Monetary',
                  color='Cluster')
```

```
fig.show()
```

[79]:
```
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt

x=rfm['Recency']
y=rfm['Frequency']
z=rfm['Monetary']
my_cmap=plt.get_cmap('hsv')

fig=plt.figure(figsize=(10,7))

ax=plt.axes(projection='3d')
sctt=ax.scatter3D( x, y, z,c=(x+y+z),cmap=my_cmap)

plt.title('3D Scatter Plot')
ax.set_xlabel('Recency',fontweight ='bold')
ax.set_ylabel('Frequency',fontweight ='bold')
ax.set_zlabel('Monetary',fontweight ='bold')
fig.colorbar(sctt,ax=ax, shrink=0.9, aspect=8)
plt.show()
```


3D Scatter Plot
Recency
Frequency
Monetary
0
50
100
150
200
250
300
350
1000
2000
3000
4000
5000
6000
7000
8000
50000
100000
150000
200000
250000

[ ]:

[ ]: