In [1]:
```
# Data Analytics Mini Project
# Name : Supriya Ananda Kore
# MIS : 111907055
# Batch : C
```

Customer Segmentation is a methodology using which we can divide our customers into group of individual who are simmilar in terms of either gender,spending behavior,frequency to visit,age or other demographics.

1. Who are your most Loyal Customers?

To calculate Customer Segmentation:

1. RFM(Recency,Frequency,Monetary) Score for each customer
2. Create cluster using K-Means

Recency: The more the recent purchases better the score would be.

Frequency: How often customer purchases product ,the more frequent they purchase the better score would be for frequency

Monetary: How much the customers spends,the more the amount the better monetary score would be

In [2]:
```
#Import necessary libraries
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [3]:
```
#Import Online Retail Data containing transaction from 01/12/2010 and 09/12/2011
Rtl_data = pd.read_csv('CSV1.csv',encoding='unicode_escape')
Rtl_data.head()
```

Out[3]:

	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

In [4]:
```
#Cheack the shape (number of columns and rows) in the dataset
Rtl_data.shape
```

Out[4]: (541909, 8)

Data Preprocessing

In [5]:
```
#Customer distribution by country
country_cust_data=Rtl_data[['Country','CustomerID']].drop_duplicates()
country_cust_data.groupby(['Country'])['CustomerID'].aggregate('count').reset_index().sort_values('CustomerID',ascending
```

Out[5]:

	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

	Country	CustomerID
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 [6]:
```
#From above we can understand that majority customers are from United Kingdom
#So that we will take only those customers
```

In [7]:
```
#Keep only United Kingdom data
Rtl_data = Rtl_data.query("Country=='United Kingdom'").reset_index(drop=True)
```

In [8]:
```
#Check for missing values in the dataset
Rtl_data.isnull().sum(axis=0)
```

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

In [9]:
```
#Remove missing values from customerID column,can ignore missing values in description column
Rtl_data = Rtl_data[pd.notnull(Rtl_data['CustomerID'])]

#Validate if there are any negative values in Quantity column
Rtl_data.Quantity.min()
```

Out[9]:
```
-80995
```

In [10]:
```
#Validate if there are any negative values in UnitPrice column
Rtl_data.UnitPrice.min()
```

Out[10]:
```
0.0
```

In [11]:
```
#Filter out records with negative values
Rtl_data = Rtl_data[(Rtl_data['Quantity']>0)]
```

In [12]:
```
#Convert the string date field to datetime
Rtl_data['InvoiceDate'] = pd.to_datetime(Rtl_data['InvoiceDate'])
```

In [13]:
```
#Add new column depecting total amount
Rtl_data['TotalAmount'] = Rtl_data['Quantity']*Rtl_data['UnitPrice']
```

In [14]:
```
#Check the shape (number of columns and rows) in the dataset after cleaning
Rtl_data.shape
```

Out[14]: (354345, 9)

In [15]:
```
Rtl_data.head()
```

Out[15]:

	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

Recency,Frequency and Monetary Modelling

In [16]:
```
#Recency = Latest Date - Last Invoice Date ,
#Frequency = count of invoice number of transaction(s),
#Monetary = Sum of Total Amount per customer
import datetime as dt

#Set Latest date 2011-12-10 as last invoice date was 2011-12-09. This is to calculate the number of days from recent pur
Latest_Date = dt.datetime(2011,12,11)

#Create RFM Modelling scores for each customer
RFMScore = Rtl_data.groupby('CustomerID').agg({'InvoiceDate': lambda x: (Latest_Date - x.max()).days, 'InvoiceNo': lambd

#Convert Invoice Date into type int
RFMScore['InvoiceDate'] = RFMScore['InvoiceDate'].astype(int)

#Rename column names as Recency,Frequency and Monetary
RFMScore.rename(columns={'InvoiceDate':'Recency',
                         'InvoiceNo':'Frequency',
                         'TotalAmount':'Monetary'},inplace=True)

RFMScore.reset_index().head()
```

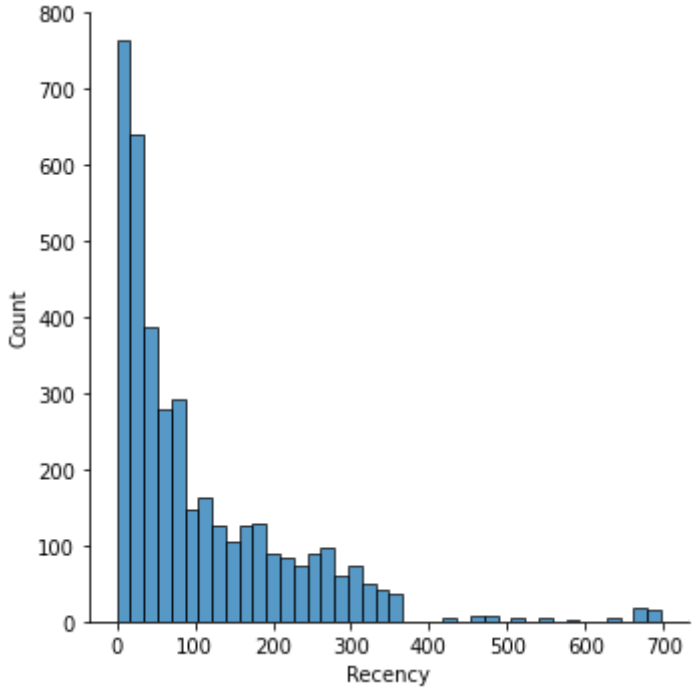
Out[16]:

	CustomerID	Recency	Frequency	Monetary
0	12346.0	326	1	77183.60
1	12747.0	23	103	4196.01
2	12748.0	5	4596	33719.73
3	12749.0	23	199	4090.88
4	12820.0	45	59	942.34

In [17]:
```
#Descriptive Statastics (Recency)
RFMScore.Recency.describe()
```

Out[17]:
```
count    3921.000000
mean      105.586585
std       115.044919
min         0.000000
25%        22.000000
50%        61.000000
75%       162.000000
max       697.000000
Name: Recency, dtype: float64
```

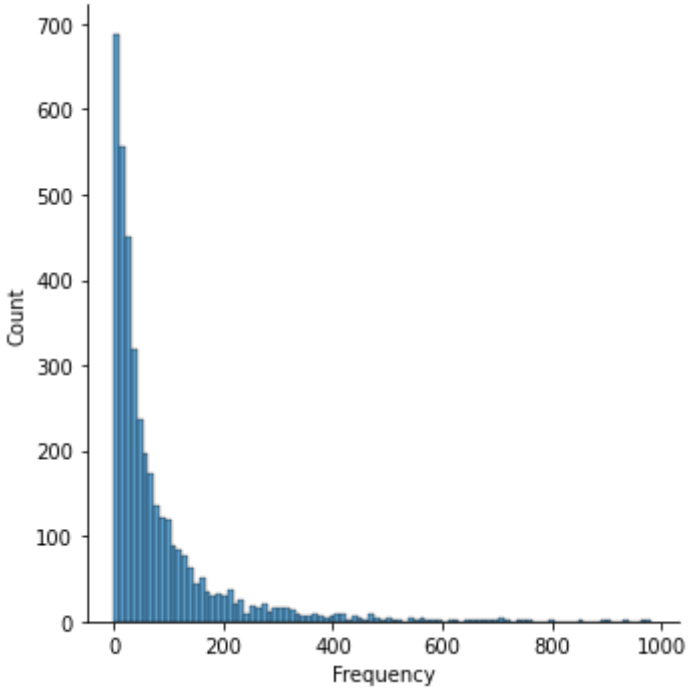
In [18]:
```
#Recency distribution plot
import seaborn as sns
x = RFMScore['Recency']

ax = sns.displot(x)
```



In [19]:
```
#Descriptive Statastics (Frequency)
RFMScore.Frequency.describe()
```

Out[19]:
```
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
```

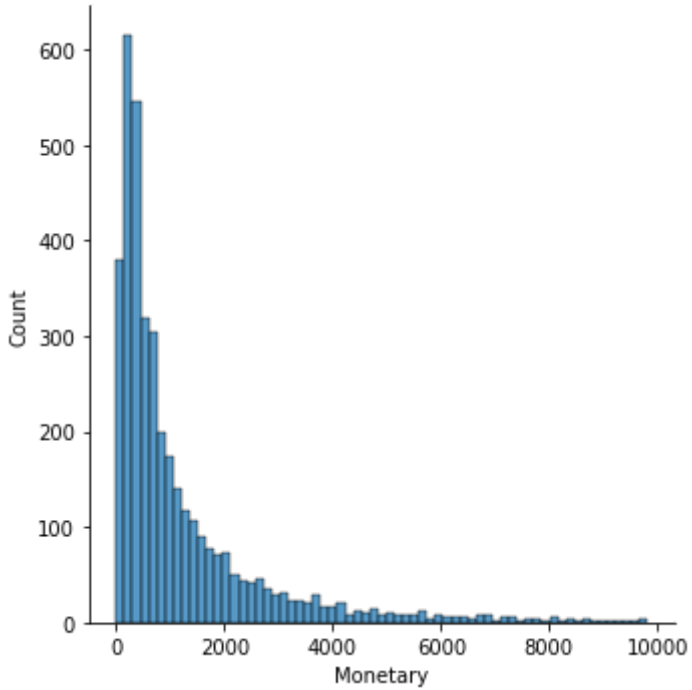
In [20]:
```
#Frequency distribution plot, taking observations which have frequency less than 1000
import seaborn as sns
x = RFMScore.query('Frequency<1000')['Frequency']

ax = sns.displot(x)
```



In [21]:
```
#Descriptive Statastics (Monetary)
RFMScore.Monetary.describe()
```

Out[21]:
```
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 [22]:
```
#Monetary distribution plot, taking observations which have monetary value less than 10000
import seaborn as sns
x = RFMScore.query('Monetary<10000')['Monetary']

ax = sns.displot(x)
```



In [23]:
```
#Split into four segment using quantiles
quantiles = RFMScore.quantile(q=[0.25,0.5,0.75])
quantiles = quantiles.to_dict()
```

In [24]:
```
quantiles
```

Out[24]:
```
{'Recency': {0.25: 22.0, 0.5: 61.0, 0.75: 162.0},
 'Frequency': {0.25: 17.0, 0.5: 41.0, 0.75: 99.0},
 'Monetary': {0.25: 300.03999999999996, 0.5: 651.8199999999999, 0.75: 1575.89}}
```

In [25]:
```
#Functions to create R,F and M segments
def RScoring(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 FnMScoring(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
```

In [26]:
```
#Calculate Add R, F and M segment value columns in the existing dataset to show R, F and M segment values
RFMScore['R'] = RFMScore['Recency'].apply(RScoring,args=('Recency',quantiles,))
RFMScore['F'] = RFMScore['Frequency'].apply(FnMScoring,args=('Frequency',quantiles,))
RFMScore['M'] = RFMScore['Monetary'].apply(FnMScoring,args=('Monetary',quantiles,))
RFMScore.head()
```

Out[26]:

CustomerID	Recency	Frequency	Monetary	R	F	M
12346.0	326	1	77183.60	4	4	1
12747.0	23	103	4196.01	2	1	1
12748.0	5	4596	33719.73	1	1	1
12749.0	23	199	4090.88	2	1	1
12820.0	45	59	942.34	2	2	2

In [27]:
```
#Calculate and Add RFMGroup value column showing combined concatenated score of RFM
RFMScore['RFMGroup'] = RFMScore.R.map(str) + RFMScore.F.map(str) + RFMScore.M.map(str)

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

Out[27]:

CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore_col
12346.0	326	1	77183.60	4	4	1	441	9
12747.0	23	103	4196.01	2	1	1	211	4
12748.0	5	4596	33719.73	1	1	1	111	3
12749.0	23	199	4090.88	2	1	1	211	4
12820.0	45	59	942.34	2	2	2	222	6

In [28]:
```
#Assign Loyalty Level to each customer
Loyalty_Level = ['Platinum','Gold','Silver']
Score_cuts = pd.qcut(RFMScore.RFMScore_col,q=3, labels = Loyalty_Level)
RFMScore['RFM_Loyalty_Level'] = Score_cuts.values
RFMScore.reset_index()
```

Out[28]:

	CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore_col	RFM_Loyalty_Level
0	12346.0	326	1	77183.60	4	4	1	441	9	Gold
1	12747.0	23	103	4196.01	2	1	1	211	4	Platinum
2	12748.0	5	4596	33719.73	1	1	1	111	3	Platinum
3	12749.0	23	199	4090.88	2	1	1	211	4	Platinum
4	12820.0	45	59	942.34	2	2	2	222	6	Platinum
...	...	...	...	...	...	...	...	...	...	...
3916	18280.0	160	10	180.60	3	4	4	344	11	Silver
3917	18281.0	4	7	80.82	1	4	4	144	9	Gold
3918	18282.0	216	12	178.05	4	4	4	444	12	Silver
3919	18283.0	10	756	2094.88	1	1	1	111	3	Platinum
3920	18287.0	0	70	1837.28	1	2	1	121	4	Platinum

3921 rows × 10 columns

In [29]:
```
#Validate the data for RFMGroup == 111
RFMScore[RFMScore['RFMGroup']== '111'].sort_values('Monetary',ascending=False).reset_index().head(10)
```

Out[29]:

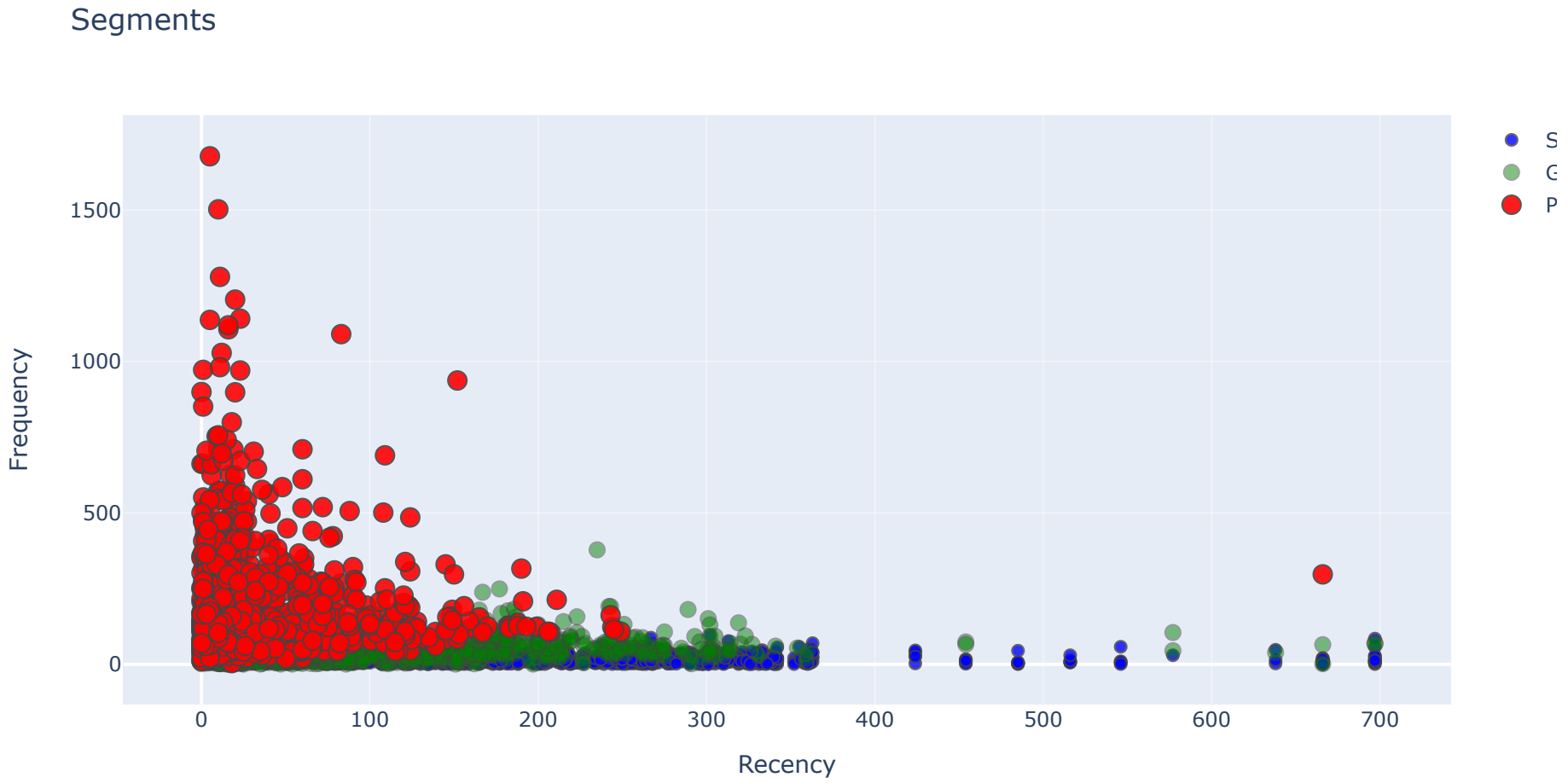
	CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore_col	RFM_Loyalty_Level
0	18102.0	12	431	259657.30	1	1	1	111	3	Platinum
1	17450.0	3	337	194550.79	1	1	1	111	3	Platinum
2	17511.0	6	963	91062.38	1	1	1	111	3	Platinum
3	16684.0	12	277	66653.56	1	1	1	111	3	Platinum
4	14096.0	12	5111	65164.79	1	1	1	111	3	Platinum
5	15311.0	0	2379	60767.90	1	1	1	111	3	Platinum
6	13089.0	6	1818	58825.83	1	1	1	111	3	Platinum
7	15061.0	5	403	54534.14	1	1	1	111	3	Platinum
8	14088.0	11	589	50491.81	1	1	1	111	3	Platinum
9	17841.0	0	7847	40991.57	1	1	1	111	3	Platinum

RFMGroup==111 are the best customers and we can try to cross sell other products of our brand and as well as we can encourage them to sign up for Loyalty programs to enjoy some elite experiences like priority access to newly launched products etc.

If RFMGroup==444 then we can give them reword or cuopon

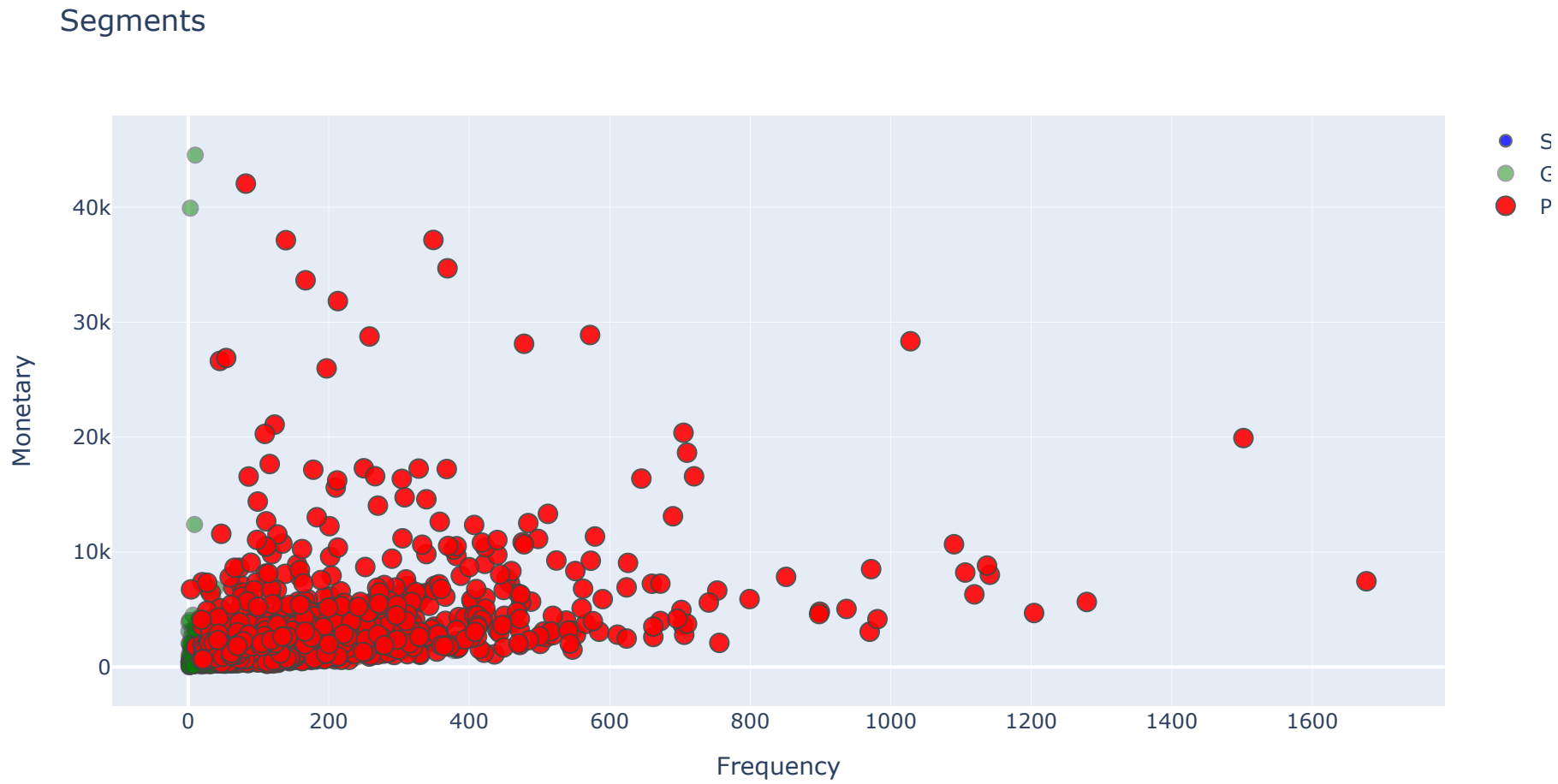
In [30]:
```
import chart_studio as cs
import plotly.offline as po
import plotly.graph_objs as gobj

#Recency vs Frequency
graph = RFMScore.query('Monetary < 50000 and Frequency < 2000')

plot_data = [
    gobj.Scatter(
    x=graph.query("RFM_Loyalty_Level == 'Silver'")['Recency'],
    y=graph.query("RFM_Loyalty_Level == 'Silver'")['Frequency'],
    mode='markers',
    name='Silver',
    marker=dict(size=7,
                line=dict(width=1),
                color='blue',
                opacity=0.8
                )
    ),
    gobj.Scatter(
    x=graph.query("RFM_Loyalty_Level == 'Gold'")['Recency'],
    y=graph.query("RFM_Loyalty_Level == 'Gold'")['Frequency'],
    mode='markers',
    name='Gold',
    marker=dict(size=9,
                line=dict(width=1),
                color='green',
                opacity=0.5
                )
    ),
    gobj.Scatter(
    x=graph.query("RFM_Loyalty_Level == 'Platinum'")['Recency'],
    y=graph.query("RFM_Loyalty_Level == 'Platinum'")['Frequency'],
    mode='markers',
    name='Platinum',
    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)
```



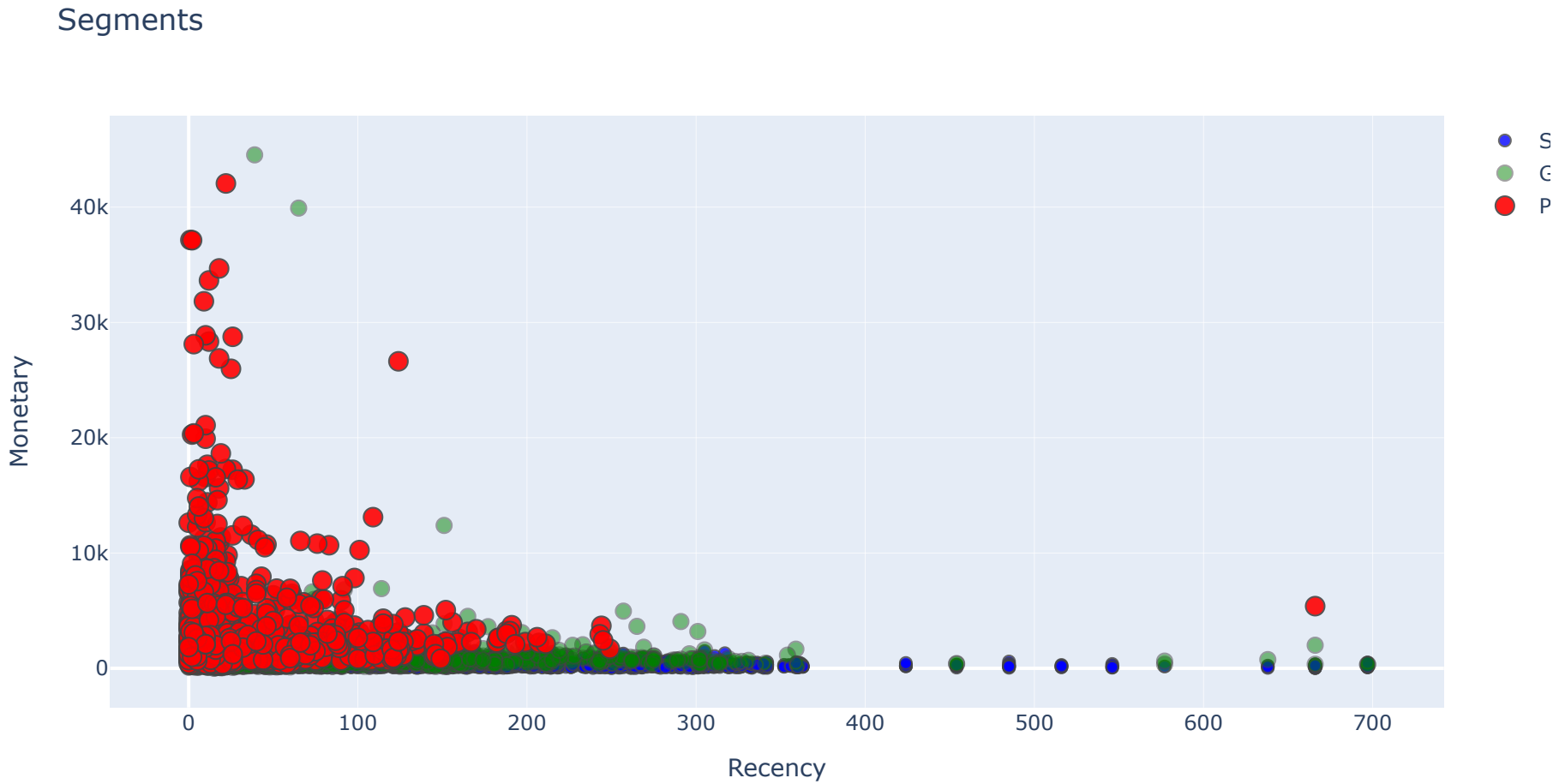
In [31]:
```
import chart_studio as cs
import plotly.offline as po
import plotly.graph_objs as gobj

#Frequency vs Monetary
graph = RFMScore.query('Monetary < 50000 and Frequency < 2000')

plot_data = [
    gobj.Scatter(
    x=graph.query("RFM_Loyalty_Level == 'Silver'")['Frequency'],
    y=graph.query("RFM_Loyalty_Level == 'Silver'")['Monetary'],
    mode='markers',
    name='Silver',
    marker=dict(size=7,
                line=dict(width=1),
                color='blue',
                opacity=0.8
                )
    ),
    gobj.Scatter(
    x=graph.query("RFM_Loyalty_Level == 'Gold'")['Frequency'],
    y=graph.query("RFM_Loyalty_Level == 'Gold'")['Monetary'],
    mode='markers',
    name='Gold',
    marker=dict(size=9,
                line=dict(width=1),
                color='green',
                opacity=0.5
                )
    ),
    gobj.Scatter(
    x=graph.query("RFM_Loyalty_Level == 'Platinum'")['Frequency'],
    y=graph.query("RFM_Loyalty_Level == 'Platinum'")['Monetary'],
    mode='markers',
    name='Platinum',
    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)
```



In [32]:
```
import chart_studio as cs
import plotly.offline as po
import plotly.graph_objs as gobj

#Recency vs Monetary
graph = RFMScore.query('Monetary < 50000 and Frequency < 2000')

plot_data = [
    gobj.Scatter(
    x=graph.query("RFM_Loyalty_Level == 'Silver'")['Recency'],
    y=graph.query("RFM_Loyalty_Level == 'Silver'")['Monetary'],
    mode='markers',
    name='Silver',
    marker=dict(size=7,
                line=dict(width=1),
                color='blue',
                opacity=0.8
                )
    ),
    gobj.Scatter(
    x=graph.query("RFM_Loyalty_Level == 'Gold'")['Recency'],
    y=graph.query("RFM_Loyalty_Level == 'Gold'")['Monetary'],
    mode='markers',
    name='Gold',
    marker=dict(size=9,
                line=dict(width=1),
                color='green',
                opacity=0.5
                )
    ),
    gobj.Scatter(
    x=graph.query("RFM_Loyalty_Level == 'Platinum'")['Recency'],
    y=graph.query("RFM_Loyalty_Level == 'Platinum'")['Monetary'],
    mode='markers',
    name='Platinum',
    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)
```

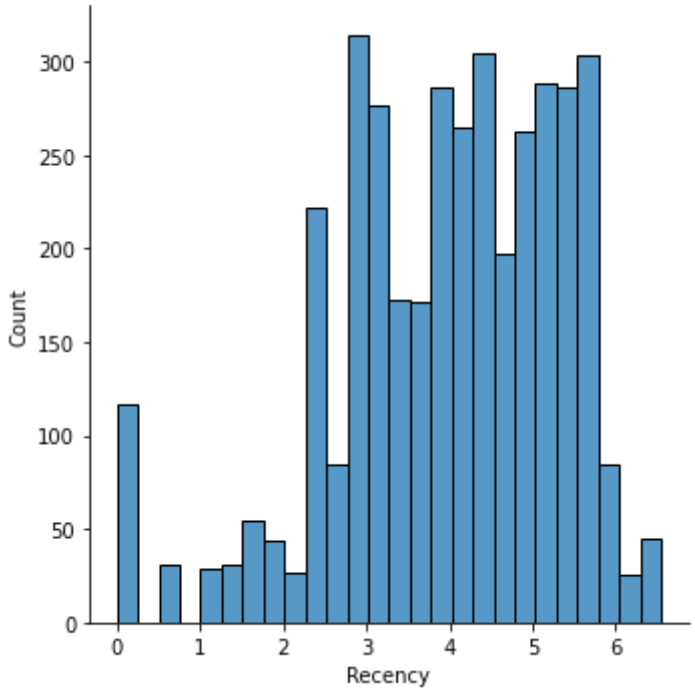


K-Means Clustering

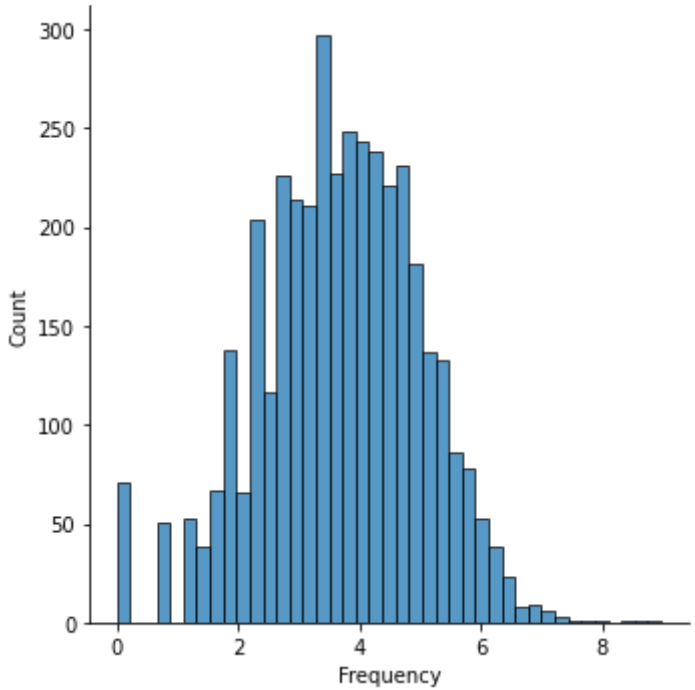
K-Means Clustering is an unsupervised learning algorithm which makes clusters or a group of a data points based on the distance between them that is we divide the data into groups based on the patterns in the data such that all the data points in a cluster should be in a similar to each other .

In [33]:
```
#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 normalized scale or near normal distribution
Log_Tfd_Data = RFMScore[['Recency','Frequency','Monetary']].apply(np.log,axis=1).round(3)
```

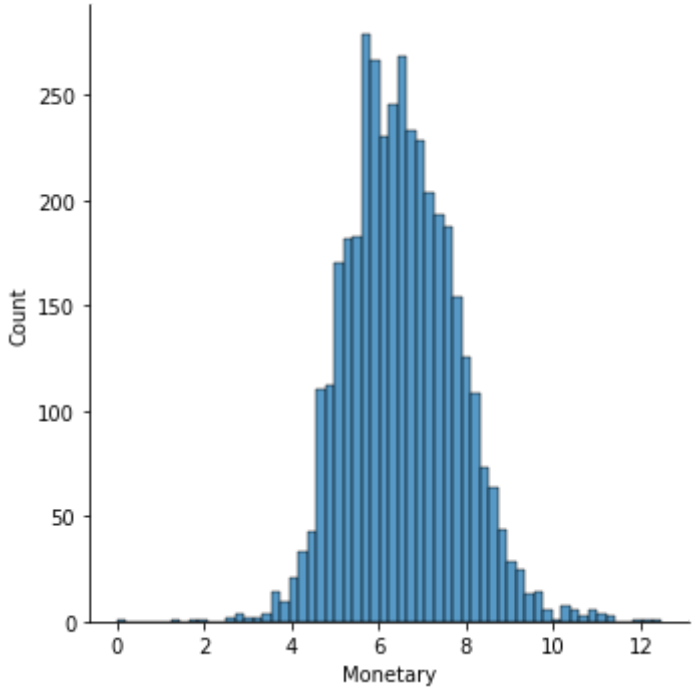
In [34]:
```
#Data distribution after data normalization for Recency
Recency_Plot = Log_Tfd_Data['Recency']
ax = sns.displot(Recency_Plot)
```



In [35]:
```
#Data distribution after data normalization for Frequency
Frequency_Plot = Log_Tfd_Data['Frequency']
ax = sns.displot(Frequency_Plot)
```

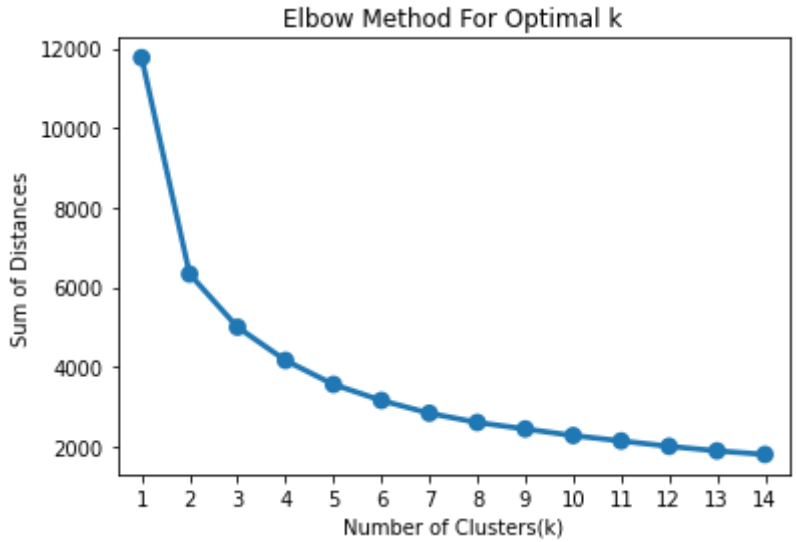


In [36]:
```
#Data distribution after data normalization for Monetary
Monetary_Plot = Log_Tfd_Data['Monetary']
ax = sns.displot(Monetary_Plot)
```



In [37]:
```
from sklearn.preprocessing import StandardScaler

#Bring the data on the 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 [38]:
```
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 Distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



Build K-Means Clustering Model

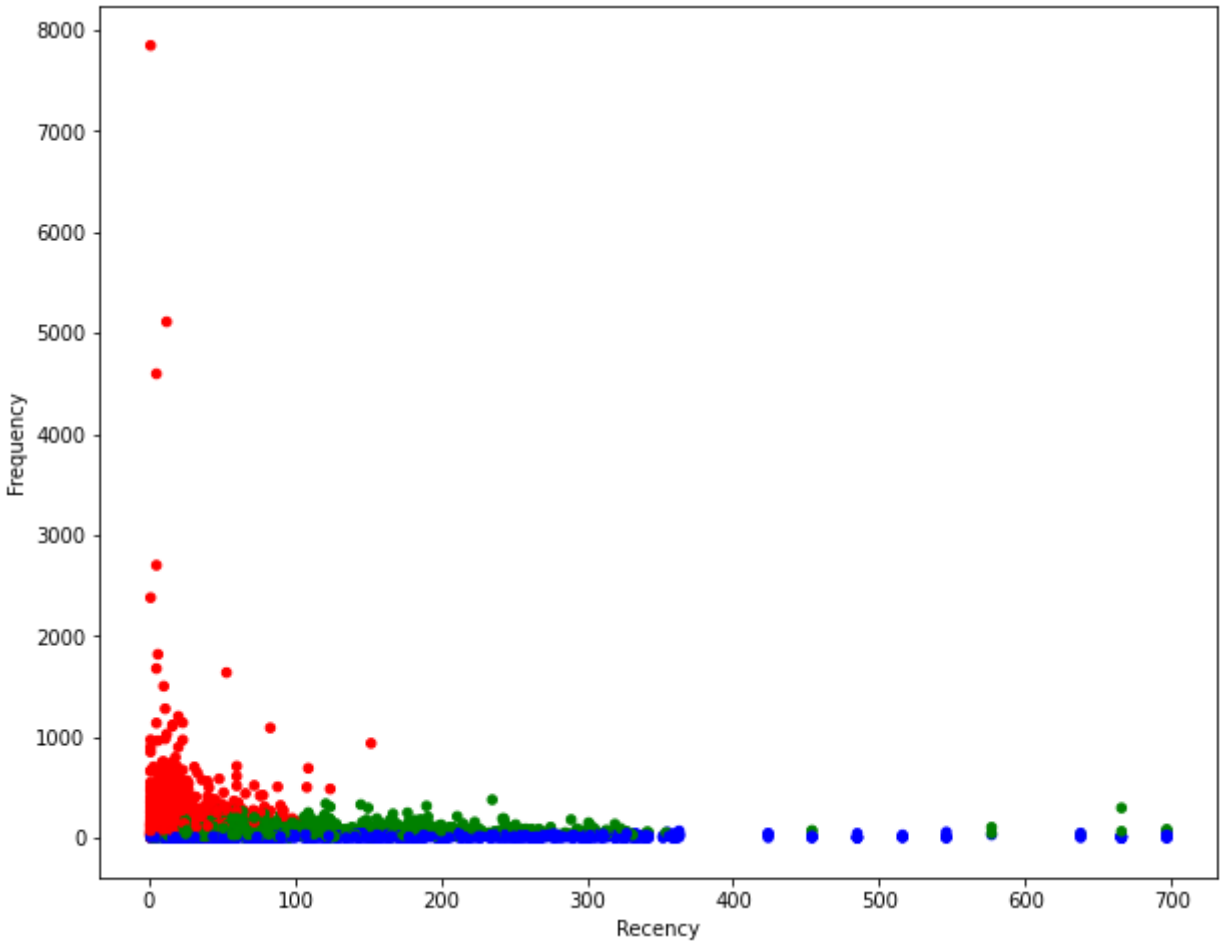
In [39]:
```
#Perform K-Means Clustering or build the K-Means clustering model
KMean_clust = KMeans(n_clusters=3, init= "k-means++",max_iter=1000)
KMean_clust.fit(Scaled_Data)

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

Out[39]:

CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore_col	RFM_Loyalty_Level	Cluster
12346.0	326	1	77183.60	4	4	1	441	9	Gold	1
12747.0	23	103	4196.01	2	1	1	211	4	Platinum	0
12748.0	5	4596	33719.73	1	1	1	111	3	Platinum	0
12749.0	23	199	4090.88	2	1	1	211	4	Platinum	0
12820.0	45	59	942.34	2	2	2	222	6	Platinum	1

In [40]:
```
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=(10,8),
     c = RFMScore['Color']
)
```

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


8000
7000
6000
5000
4000
3000
2000
1000
0
Frequency
0
100
200
300
400
500
600
700
Recency

In [41]:
```
RFMScore.head()
```

Out[41]:

CustomerID	Recency	Frequency	Monetary	R	F	M	RFMGroup	RFMScore_col	RFM_Loyalty_Level	Cluster	Color
12346.0	326	1	77183.60	4	4	1	441	9	Gold	1	green
12747.0	23	103	4196.01	2	1	1	211	4	Platinum	0	red
12748.0	5	4596	33719.73	1	1	1	111	3	Platinum	0	red
12749.0	23	199	4090.88	2	1	1	211	4	Platinum	0	red
12820.0	45	59	942.34	2	2	2	222	6	Platinum	1	green