Data Modelling Approach

- **Step 1: Importing the Relevant Libraries**
- **Step 2: Data Inspection and Cleaning**
- **Step 3: Data Exploration**
- **Step 4: RFM Modelling**
- **Step 5: find out the customers who are 'champions', 'Potential customers' and 'need attention**
- **Step 6: Data Visualization After RFM Analysis**
- **Step 7: K-Means Clustering Technique**

Step 1: Importing the Relevant Libraries

In [1]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
from matplotlib.gridspec import GridSpec
import seaborn as sns
import plotly.express as px
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler as ss
```

Step 2: Data Inspection and Cleaning

In [2]:
```
data = pd.read_excel("C:/Users/AkshayPharande/Downloads/Round 1 Assignment/Round 1 Assignment/sales_data.xlsx")
data.head()
```

Out[2]:

	CustomerID	TOTAL_ORDERS	REVENUE	AVERAGE_ORDER_VALUE	CARRIAGE_REVENUE	AVERAGESHIPPING	FIRST_ORDER_DATE	LATEST_ORDER_DATE	AVGDAYSBETWEENORDERS	DAYSSINCELASTORDER	...	WEEK3_DAY16_DAY23_REVENUE	WEE
0	22	124	11986.54	96.67	529.59	4.27	2016-12-30	2021-10-24	14.19	1	...	2592.18	
1	29	82	11025.96	134.46	97.92	1.19	2018-03-31	2021-10-24	15.89	1	...	2807.66	
2	83	43	7259.69	168.83	171.69	3.99	2017-11-30	2021-10-24	33.12	1	...	713.94	
3	95	44	6992.27	158.92	92.82	2.11	2019-04-09	2021-10-24	21.11	1	...	997.02	
4	124	55	6263.44	113.88	179.04	3.26	2020-10-23	2021-10-24	6.65	1	...	2725.66	

5 rows × 40 columns

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

Out[3]:
```
(5000, 40)
```

- **We have 5000 rows and 40 columns**

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

Out[4]:
```
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
```

- **NO missing values present in data and this is a clean dataset.**

Step 3: Data Exploration

In [5]:
```
data.describe()
```

Out[5]:

	CustomerID	TOTAL_ORDERS	REVENUE	AVERAGE_ORDER_VALUE	CARRIAGE_REVENUE	AVERAGESHIPPING	AVGDAYSBETWEENORDERS	DAYSSINCELASTORDER	MONDAY_ORDERS	TUESDAY_ORDERS	...	WEEK3_DAY16_DAY23_REVENUE
count	5000.000000	5000.00000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.00000	...	5000.000000
mean	40709.227800	12.87040	1681.523840	136.537378	46.036376	3.592574	163.159618	87.420000	1.629000	1.75440	...	421.826908
std	49949.848017	12.67988	1998.618678	91.651569	47.879226	2.021360	259.699496	80.156513	2.236506	2.43394	...	643.449120
min	1.000000	1.00000	38.500000	10.680000	0.000000	0.000000	0.000000	1.000000	0.000000	0.00000	...	0.000000
25%	1687.500000	3.00000	315.097500	83.025000	9.980000	2.500000	21.670000	7.000000	0.000000	0.00000	...	0.000000
50%	13765.000000	8.00000	966.725000	113.160000	24.985000	3.660000	57.635000	68.000000	1.000000	1.00000	...	194.990000
75%	71891.500000	20.00000	2493.072500	160.272500	76.862500	4.790000	170.357500	171.250000	2.000000	3.00000	...	604.085000
max	277160.000000	156.00000	34847.400000	1578.880000	529.590000	35.990000	1409.500000	207.000000	19.000000	23.00000	...	12946.220000

8 rows × 38 columns

In [6]:
```
AVG_Daily_Orders = [data[['MONDAY_ORDERS','TUESDAY_ORDERS','WEDNESDAY_ORDERS','THURSDAY_ORDERS','FRIDAY_ORDERS','SATURDAY_ORDERS','SUNDAY_ORDERS']].mean()]
AVG_Daily_Revenue = [data[['MONDAY_REVENUE','TUESDAY_REVENUE','WEDNESDAY_REVENUE','THURSDAY_REVENUE','FRIDAY_REVENUE','SATURDAY_REVENUE','SUNDAY_REVENUE']].mean()]
```

In [7]:
```
AVG_Daily_Orders, AVG_Daily_Revenue
```

Out[7]:
```
([MONDAY_ORDERS       1.6290
  TUESDAY_ORDERS      1.7544
  WEDNESDAY_ORDERS    1.7980
  THURSDAY_ORDERS     2.1340
  FRIDAY_ORDERS       1.9462
  SATURDAY_ORDERS     1.6834
  SUNDAY_ORDERS       1.9254
  dtype: float64],
 [MONDAY_REVENUE       215.208336
  TUESDAY_REVENUE      233.510430
  WEDNESDAY_REVENUE    235.689294
  THURSDAY_REVENUE     265.949796
  FRIDAY_REVENUE       250.580554
  SATURDAY_REVENUE     219.642100
  SUNDAY_REVENUE       260.943330
  dtype: float64])
```

According to the data above, Thursday, Friday, and Sunday have higher daily average customer orders than other days. Additionally, the average revenue generated on these days is higher than on other days..

In [8]:
```
AVG_Weekly_Orders = [data[['WEEK1_DAY01_DAY07_ORDERS','WEEK2_DAY08_DAY15_ORDERS','WEEK3_DAY16_DAY23_ORDERS','WEEK4_DAY24_DAY31_ORDERS']].mean()]
AVG_Weekely_Revenue = [data[['WEEK1_DAY01_DAY07_REVENUE','WEEK2_DAY08_DAY15_REVENUE','WEEK3_DAY16_DAY23_REVENUE','WEEK4_DAY24_DAY31_REVENUE']].mean()]
```

In [9]:
```
AVG_Weekly_Orders, AVG_Weekely_Revenue
```

Out[9]:
```
([WEEK1_DAY01_DAY07_ORDERS    2.9978
  WEEK2_DAY08_DAY15_ORDERS    3.0626
  WEEK3_DAY16_DAY23_ORDERS    3.2300
  WEEK4_DAY24_DAY31_ORDERS    3.5800
  dtype: float64],
 [WEEK1_DAY01_DAY07_REVENUE    378.638346
  WEEK2_DAY08_DAY15_REVENUE    406.595734
  WEEK3_DAY16_DAY23_REVENUE    421.826908
  WEEK4_DAY24_DAY31_REVENUE    474.462852
  dtype: float64])
```

According to the graph above, average revenue and orders increase from week 1 to week 4, and most customers prefer to buy products in the last week of the month.

Step 4: RFM Modeling Technique :

Here we will calculate the Recency, Frequency and Monetary for the customers and those are defined as ;

- Recency : How much time has elapsed since a customer's last activity or transaction with the brand? i.e DAYSSINCELASTORDER
- Frequency : How often has a customer transacted or interacted? i.e. TOTAL_ORDERS
- Monetary : How much a customer has spent with the brand during a particular period of time? i.e.REVENUE

  Therefore, We need DAYSSINCELASTORDER, TOTAL_ORDERS and REVENUE columns to do RFM Modelling.

In [10]:
```
#RFM factors calculation:
#converting the names of the columns
RFM_data = data.rename(columns = {'DAYSSINCELASTORDER' : 'Recency',
                          'TOTAL_ORDERS' : "Frequency",
                          'REVENUE' : "Monetary"},inplace = False)
RFM_data.head()
```

Out[10]:

	CustomerID	Frequency	Monetary	AVERAGE_ORDER_VALUE	CARRIAGE_REVENUE	AVERAGESHIPPING	FIRST_ORDER_DATE	LATEST_ORDER_DATE	AVGDAYSBETWEENORDERS	Recency	...	WEEK3_DAY16_DAY23_REVENUE	WEEK4_DAY24_DAY31_RI
0	22	124	11986.54	96.67	529.59	4.27	2016-12-30	2021-10-24	14.19	1	...	2592.18	
1	29	82	11025.96	134.46	97.92	1.19	2018-03-31	2021-10-24	15.89	1	...	2807.66	
2	83	43	7259.69	168.83	171.69	3.99	2017-11-30	2021-10-24	33.12	1	...	713.94	
3	95	44	6992.27	158.92	92.82	2.11	2019-04-09	2021-10-24	21.11	1	...	997.02	
4	124	55	6263.44	113.88	179.04	3.26	2020-10-23	2021-10-24	6.65	1	...	2725.66	

5 rows × 40 columns

In [11]:
```
RFM_data_1 = RFM_data[['CustomerID','Recency','Frequency','Monetary']]
```

In [12]:
```
RFM_data_1.head()
```

Out[12]:

	CustomerID	Recency	Frequency	Monetary
0	22	1	124	11986.54
1	29	1	82	11025.96
2	83	1	43	7259.69
3	95	1	44	6992.27
4	124	1	55	6263.44

In [13]:
```
# RFM_data Description/ Summary
RFM_data_1.iloc[:,1:4].describe()
```

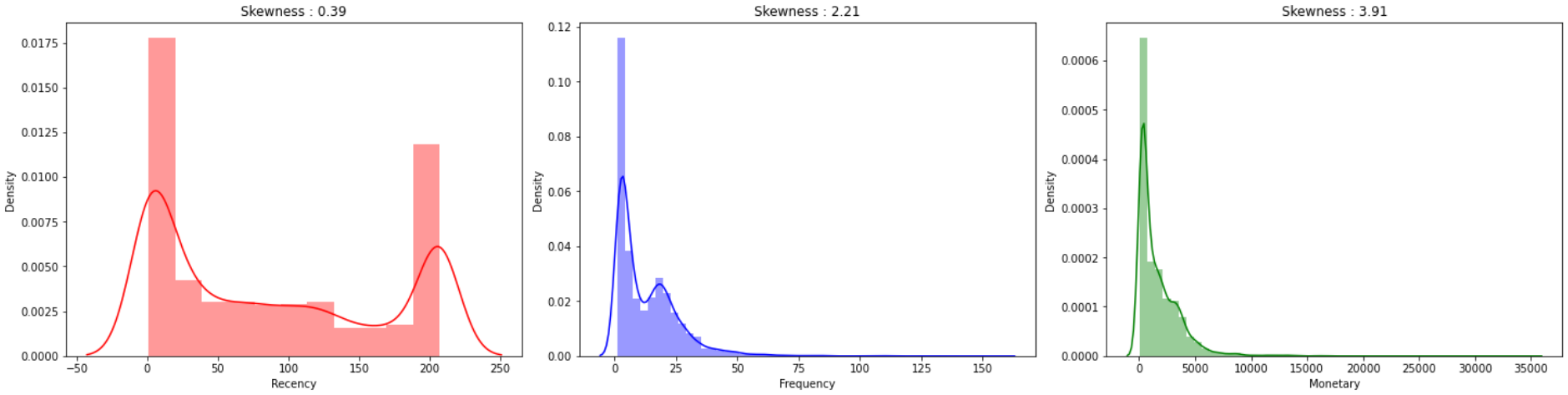
Out[13]:

	Recency	Frequency	Monetary
count	5000.000000	5000.00000	5000.000000
mean	87.420000	12.87040	1681.523840
std	80.156513	12.67988	1998.618678
min	1.000000	1.00000	38.500000
25%	7.000000	3.00000	315.097500
50%	68.000000	8.00000	966.725000
75%	171.250000	20.00000	2493.072500
max	207.000000	156.00000	34847.400000

From above result, we can observe that average recency of the customers are 87 days (approx), an average customer are purchasing the product 13 times and spending an average 1681.52 unitprice.

In [14]:
```
#Visualizing the Recency, Frequency and Monetary distributions.
i = 0
fig = plt.figure(constrained_layout = True,figsize = (20,5))
gs = GridSpec(1, 3, figure=fig)

col = ['red','blue','green']
for var in list(RFM_data_1.columns[1:4]):
    plt.subplot(gs[0,i])
    sns.distplot(RFM_data_1[var],color= col[i])
    plt.title('Skewness ' + ': ' + round(RFM_data_1[var].skew(),2).astype(str))
    i= i+1
```

```
C:\Users\AkshayPharande\Conda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\AkshayPharande\Conda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\AkshayPharande\Conda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



From above distribution plots we can observe that all of these three metrics are positively skewed or right skewed. And also as per skewness coefficient values indicating same.

Segmentation :

In [15]:
```
#Here, we will divide the data set into 4 parts based on the quantiles.
quantiles = RFM_data_1.drop('CustomerID',axis = 1).quantile(q = [0.33,0.67])
quantiles.to_dict()
```

Out[15]:
```
{'Recency': {0.33: 19.0, 0.67: 128.0},
 'Frequency': {0.33: 4.0, 0.67: 17.0},
 'Monetary': {0.33: 427.09400000000005, 0.67: 1905.1959}}
```

In [16]:
```
#Creating the R,F and M scoring/segement function
#[1] Recency scoring (Negative Impact : Higher the value, less valuable)
def R_score(var,p,d):
    if var <= d[p][0.33]:
        return 1
    elif var <= d[p][0.67]:
        return 2
    else:
        return 3

#[2] Frequency and Monetary (Positive Impact : Higher the value, better the customer)
def FM_score(var,p,d):
    if var <= d[p][0.33]:
        return 3
    elif var <= d[p][0.67]:
        return 2
    else:
        return 1


#Scoring:
RFM_data_1['R_score'] = RFM_data_1['Recency'].apply(R_score,args = ('Recency',quantiles,))
RFM_data_1['F_score'] = RFM_data_1['Frequency'].apply(FM_score,args = ('Frequency',quantiles,))
RFM_data_1['M_score'] = RFM_data_1['Monetary'].apply(FM_score,args = ('Monetary',quantiles,))
RFM_data_1.head()
```

```
C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\612544624.py:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  RFM_data_1['R_score'] = RFM_data_1['Recency'].apply(R_score,args = ('Recency',quantiles,))
C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\612544624.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  RFM_data_1['F_score'] = RFM_data_1['Frequency'].apply(FM_score,args = ('Frequency',quantiles,))
C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\612544624.py:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  RFM_data_1['M_score'] = RFM_data_1['Monetary'].apply(FM_score,args = ('Monetary',quantiles,))
```

Out[16]:

	CustomerID	Recency	Frequency	Monetary	R_score	F_score	M_score
0	22	1	124	11986.54	1	1	1
1	29	1	82	11025.96	1	1	1
2	83	1	43	7259.69	1	1	1
3	95	1	44	6992.27	1	1	1
4	124	1	55	6263.44	1	1	1

In [17]:
```
#Now we will create : RFMGroup and RFMScore
RFM_data_1['RFM_Group'] = RFM_data_1['R_score'].astype(str) + RFM_data_1['F_score'].astype(str) + RFM_data_1['M_score'].astype(str)

#Score
RFM_data_1['RFM_Score'] = RFM_data_1[['R_score','F_score','M_score']].sum(axis = 1)
RFM_data_1.head()
```

```
C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\1189733458.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  RFM_data_1['RFM_Group'] = RFM_data_1['R_score'].astype(str) + RFM_data_1['F_score'].astype(str) + RFM_data_1['M_score'].astype(str)
C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\1189733458.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  RFM_data_1['RFM_Score'] = RFM_data_1[['R_score','F_score','M_score']].sum(axis = 1)
```

Out[17]:

	CustomerID	Recency	Frequency	Monetary	R_score	F_score	M_score	RFM_Group	RFM_Score
0	22	1	124	11986.54	1	1	1	111	3
1	29	1	82	11025.96	1	1	1	111	3
2	83	1	43	7259.69	1	1	1	111	3
3	95	1	44	6992.27	1	1	1	111	3
4	124	1	55	6263.44	1	1	1	111	3

RFM Scores have been calculated now we will use this score to make segments of the customers and define level of loyality.

Step 5: find out the customers who are 'champions', 'Potential customers' and 'need attention'

In [18]:
```
#Creating the Customer segments/ Loyality_level
Segment = ['champions','Potential customers','need attention']
cuts = pd.qcut(RFM_data_1['RFM_Score'],q = 3,labels=Segment)
RFM_data_1['Segment'] = cuts.values
RFM_data_1.tail(15)
```

```
C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\3162852954.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  RFM_data_1['Segment'] = cuts.values
```

Out[18]:

	CustomerID	Recency	Frequency	Monetary	R_score	F_score	M_score	RFM_Group	RFM_Score	Segment
4985	173166	207	3	118.68	3	3	3	333	9	need attention
4986	173176	207	4	118.59	3	3	3	333	9	need attention
4987	173219	207	1	118.49	3	3	3	333	9	need attention
4988	173315	207	1	118.49	3	3	3	333	9	need attention
4989	173503	207	2	118.08	3	3	3	333	9	need attention
4990	173766	207	4	117.79	3	3	3	333	9	need attention
4991	173792	207	1	117.59	3	3	3	333	9	need attention
4992	173842	207	1	117.49	3	3	3	333	9	need attention
4993	173857	207	1	117.49	3	3	3	333	9	need attention
4994	173944	207	1	117.49	3	3	3	333	9	need attention
4995	173946	207	1	117.49	3	3	3	333	9	need attention
4996	173987	207	1	117.49	3	3	3	333	9	need attention
4997	174004	207	1	117.49	3	3	3	333	9	need attention
4998	174038	207	1	117.49	3	3	3	333	9	need attention
4999	200783	207	2	94.14	3	3	3	333	9	need attention

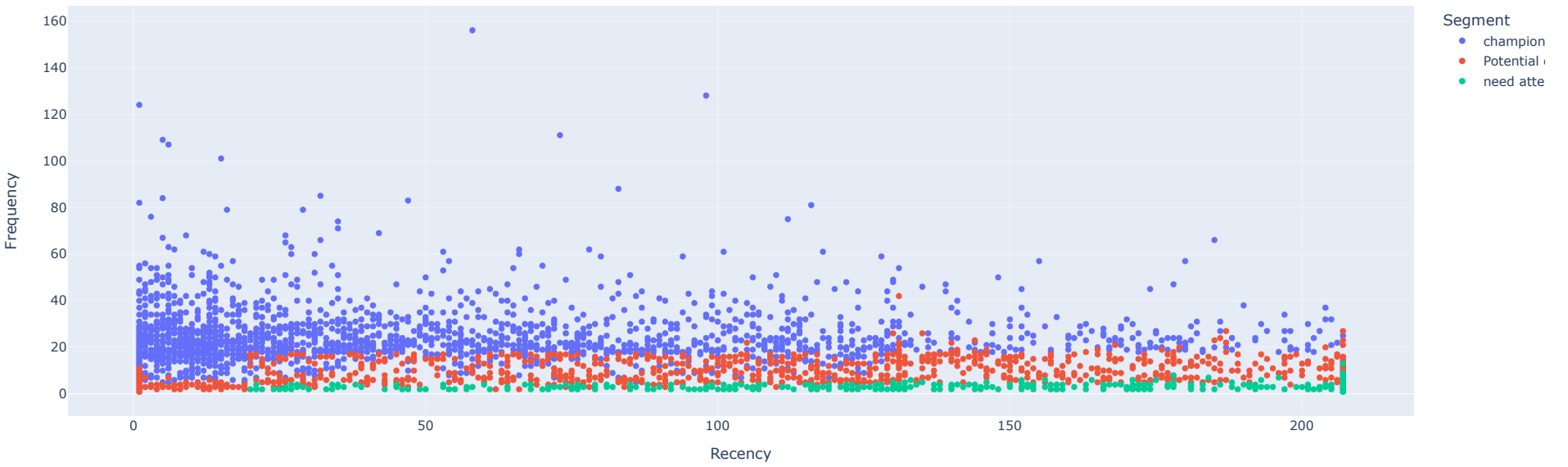
We have classified our customer into four segments based on their R,F and M scores.

Step 6: Data Visualization

_Visualization for Recency, Frequency and Monetary : RFMSegment

1.Recency V/s Frequency

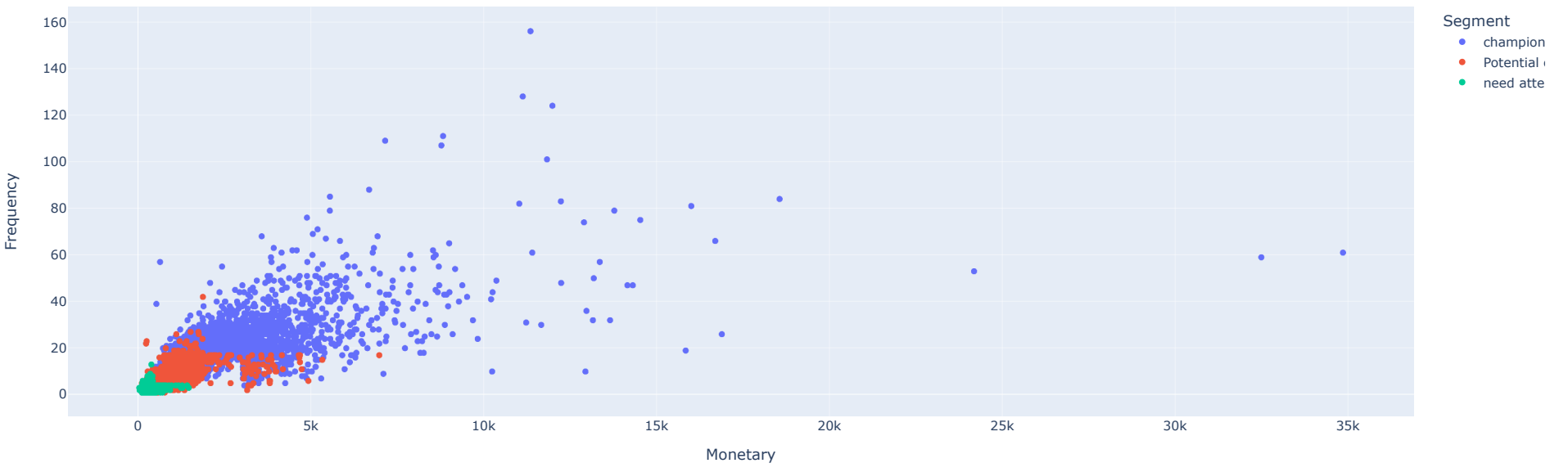
In [19]:
```
fig = px.scatter(RFM_data_1,x = "Recency", y = "Frequency",color = "Segment")
fig.show()
```



We can see the customers whose Recency is less than two month have high Frequency i.e the customers buying more when their recency is less.

2.Frequency V/s Monetary

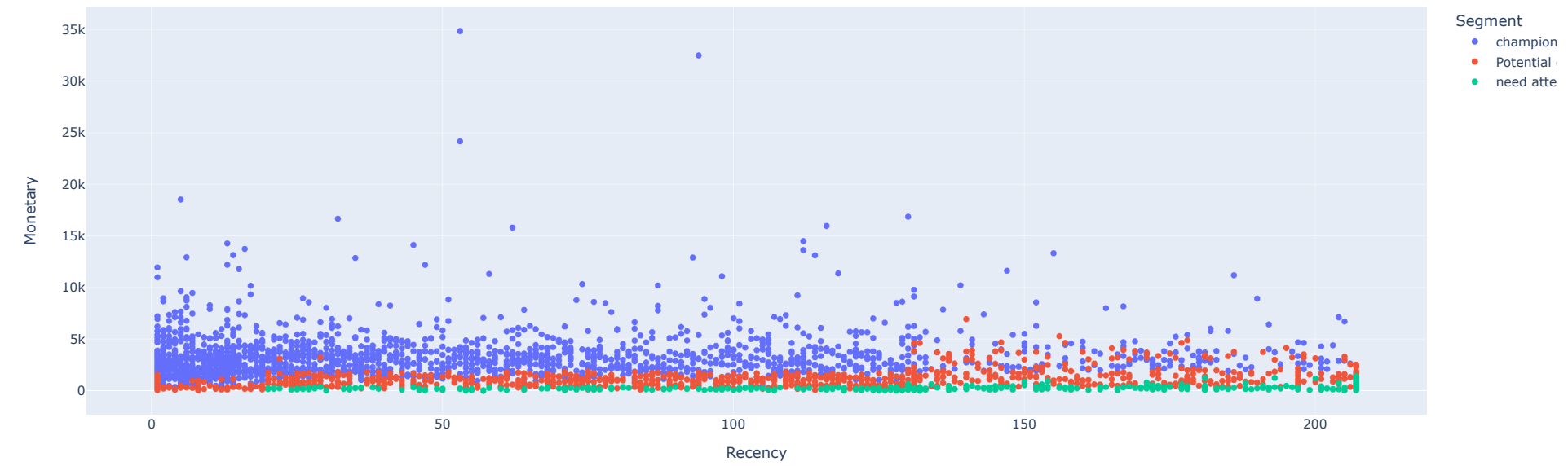
In [20]:
```
fig = px.scatter(RFM_data_1,x = "Monetary", y = "Frequency",color = "Segment")
fig.show()
```



We can see, customers buying frequently are spending less amount.

3. Recency V/s Monetary

In [21]:
```
fig = px.scatter(RFM_data_1,x = "Recency", y = "Monetary",color = "Segment")
fig.show()
```



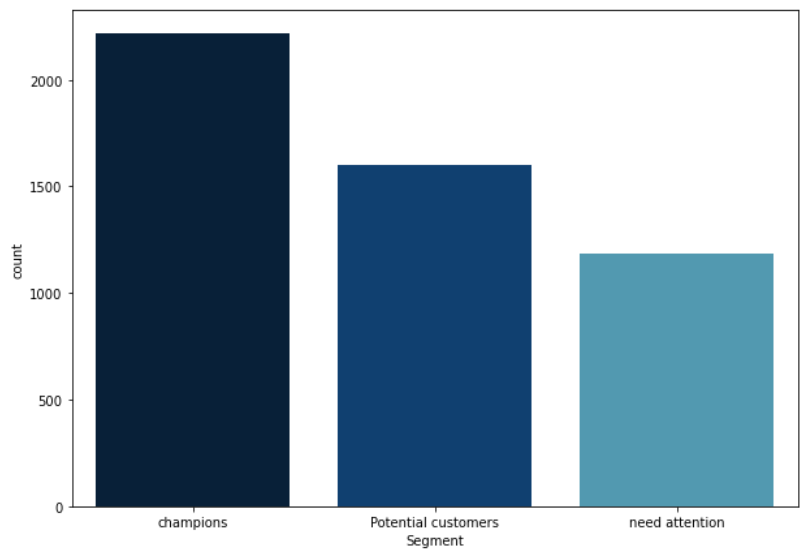
We can see the customers whose Recency is less than two month have high Monetary i.e the customers spending more when their recency is less.

4. RFM_Segment

In [22]:
```
plt.figure(figsize=(10,7))
sns.countplot('Segment',data=RFM_data_1,palette='ocean')
```

```
C:\Users\AkshayPharande\Conda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misin
terpretation.
```

Out[22]:
```
<AxesSubplot:xlabel='Segment', ylabel='count'>
```



From Above chart It is clear that almost 50% customers are spend time and buy products and around 25-30% customers are potential customers and 20-25% customers need attention.

Step 7: K-Means Clustering Technique :

How the K-means algorithm works ?

Clustering is the process of dividing the entire data into groups (also known as clusters) based on the patterns in the data.

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids.

To create the customer segementation based on the K-Means Clustering based on the R, F, and M Scores: Before that we will bring them into same scale and normalise them.

In [23]:
```
# First will focus on the negativ and zero before the transformation.
def right_treat(var):
    if var <= 0:
        return 1
    else:
        return var

# Describing the data
RFM_data_1.describe()
```

Out[23]:

	CustomerID	Recency	Frequency	Monetary	R_score	F_score	M_score	RFM_Score
count	5000.000000	5000.000000	5000.00000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	40709.227800	87.420000	12.87040	1681.523840	1.995400	2.047200	2.000000	6.042600
std	49949.848017	80.156513	12.67988	1998.618678	0.812595	0.817744	0.812485	1.880929
min	1.000000	1.000000	1.00000	38.500000	1.000000	1.000000	1.000000	3.000000
25%	1687.500000	7.000000	3.00000	315.097500	1.000000	1.000000	1.000000	4.000000
50%	13765.000000	68.000000	8.00000	966.725000	2.000000	2.000000	2.000000	6.000000
75%	71891.500000	171.250000	20.00000	2493.072500	3.000000	3.000000	3.000000	7.000000
max	277160.000000	207.000000	156.00000	34847.400000	3.000000	3.000000	3.000000	9.000000

From above we can see that there is no 0 values present in Frequency , Monetary and Recency.

In [24]:
```
#Applying on the data.
RFM_data_1['Recency'] = RFM_data_1['Recency'].apply(Lambda x : right_treat(x))
RFM_data_1['Frequency'] = RFM_data_1['Frequency'].apply(Lambda x : right_treat(x))
RFM_data_1['Monetary'] = RFM_data_1['Monetary'].apply(Lambda x : right_treat(x))

#Checking the Skewness of R, F and M
print('Recency Skewness : ' + RFM_data_1['Recency'].skew().astype(str))
print('Frequency Skewness : ' + RFM_data_1['Frequency'].skew().astype(str))
print('Monetary Skewness : ' + RFM_data_1['Monetary'].skew().astype(str))
```

```
Recency Skewness : 0.390002913835805
Frequency Skewness : 2.205765054664766
Monetary Skewness : 3.9057746675131164
```

```
C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\3514597734.py:2: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\3514597734.py:3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\3514597734.py:4: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

All the variables are right skewed, so will make log transformation of it.

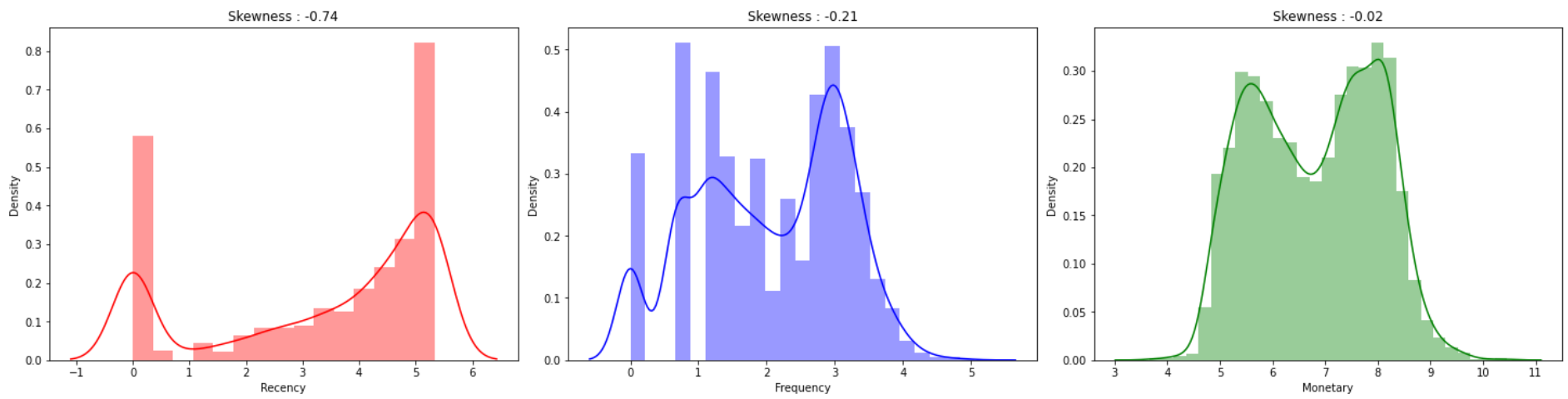
In [25]:
```
#Log Transformation
log_RFM_data = RFM_data_1[['Recency','Frequency','Monetary']].apply(np.log,axis = 1).round(3)
```

In [26]:
```
i = 0
fig = plt.figure(constrained_layout = True,figsize = (20,5))
gs = GridSpec(1, 3, figure=fig)

col = ['red','blue','green']
for var in list(log_RFM_data.columns[0:3]):
    plt.subplot(gs[0,i])
    sns.distplot(log_RFM_data[var],color= col[i])
    plt.title('Skewness ' + ': ' + round(log_RFM_data[var].skew(),2).astype(str))
    i= i+1
log_RFM_data.describe()
```

```
C:\Users\AkshayPharande\Conda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:

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

C:\Users\AkshayPharande\Conda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:

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

C:\Users\AkshayPharande\Conda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:

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

Out[26]:

	Recency	Frequency	Monetary
count	5000.000000	5000.000000	5000.000000
mean	3.380851	2.041230	6.807619
std	2.008518	1.093141	1.173694
min	0.000000	0.000000	3.651000
25%	1.946000	1.099000	5.753000
50%	4.220000	2.079000	6.874000
75%	5.143250	2.996000	7.821250
max	5.333000	5.050000	10.459000

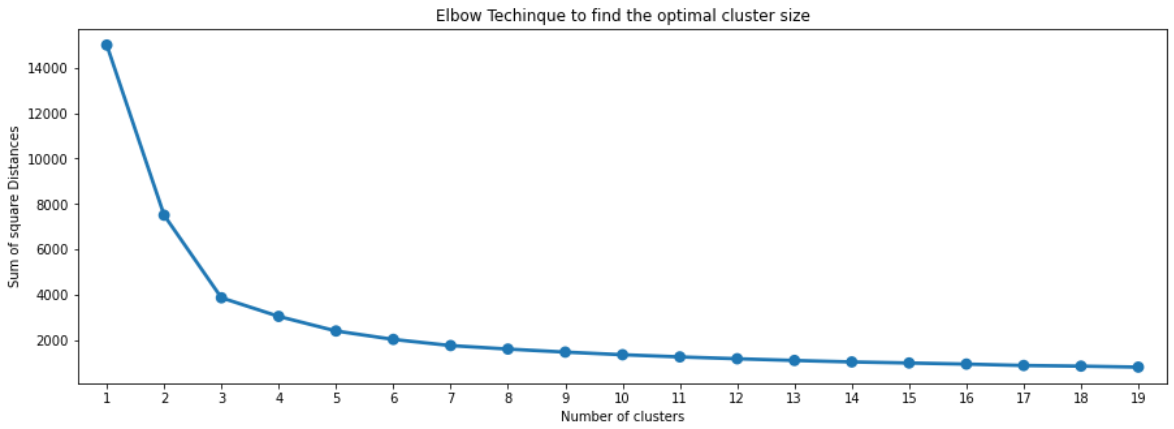


Therefore all the variables are now approximately normally distributed. Will make all of them on the same scale as Monetary is little large in values.

In [27]:
```
#Scaling the data
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
Scaled_RFM_data = ss.fit_transform(log_RFM_data)
Scaled_RFM_data = pd.DataFrame(Scaled_RFM_data,columns=log_RFM_data.columns,index=log_RFM_data.index)
```

In [28]:
```
# Will search the optimal number of cluster based on the Elbow Method as below:
SS_distance = {}
for k in range(1,20):
    mod = KMeans(n_clusters= k, max_iter=1000,init = 'k-means++')
    mod = mod.fit(Scaled_RFM_data)
    SS_distance[k] = mod.inertia_

#Plotting the sum of square distance values and numbers of clusters
plt.figure(figsize = (15,5))
sns.pointplot(x = list(SS_distance.keys()), y = list(SS_distance.values()))
plt.xlabel("Number of clusters")
plt.ylabel("Sum of square Distances")
plt.title("Elbow Techinque to find the optimal cluster size")
```

Out[28]:
```
Text(0.5, 1.0, 'Elbow Techinque to find the optimal cluster size')
```



We can observe that as the number of cluster increases the sum of square distance are becoming lesser. And will take the count of cluster where this elbow is bending. In our cases, sum of square distance is dramatically decreasing at K = 3, so this is optimal value to choose for no of clusters.

In [29]:
```
# Now we will perform K- means clustering on the data set.
KM_clust = KMeans(n_clusters= 3, init = 'k-means++',max_iter = 1000)
KM_clust.fit(Scaled_RFM_data)

# Mapping on the data
RFM_data_1['Cluster'] = KM_clust.labels_
RFM_data_1['Cluster'] = 'Cluster' + RFM_data_1['Cluster'].astype(str)
RFM_data_1.head()
```

```
C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\821865125.py:6: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\AkshayPharande\AppData\Local\Temp\ipykernel_3856\821865125.py:7: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

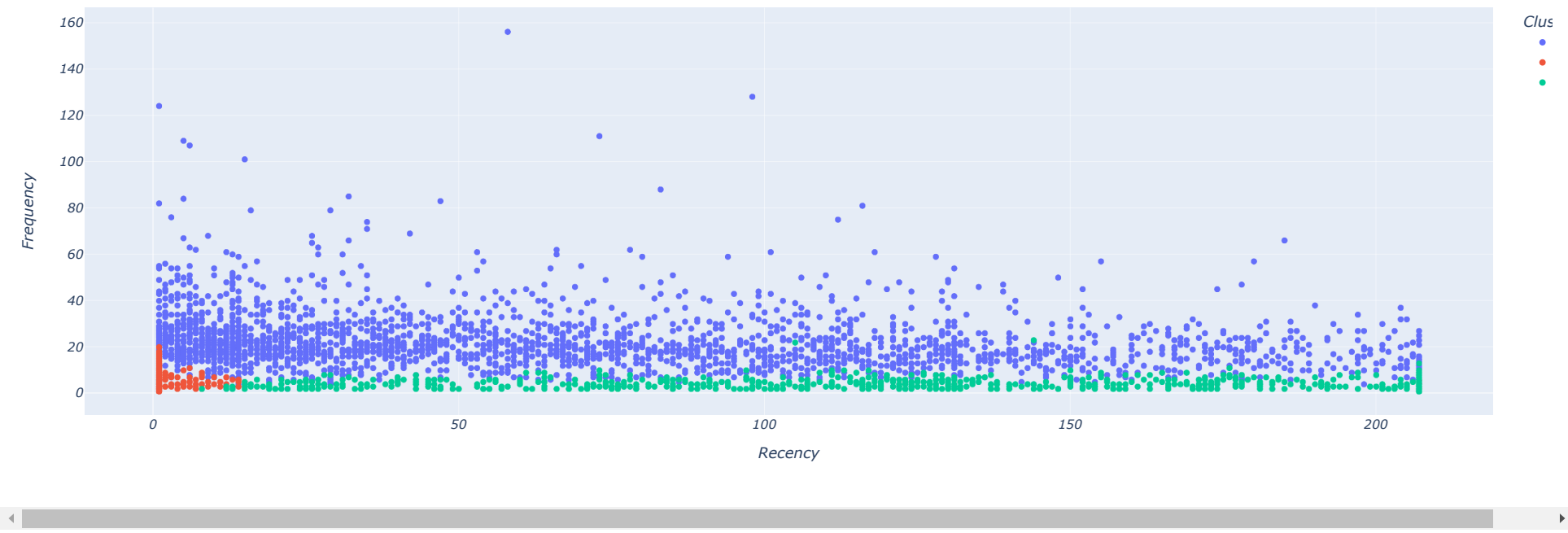
Out[29]:

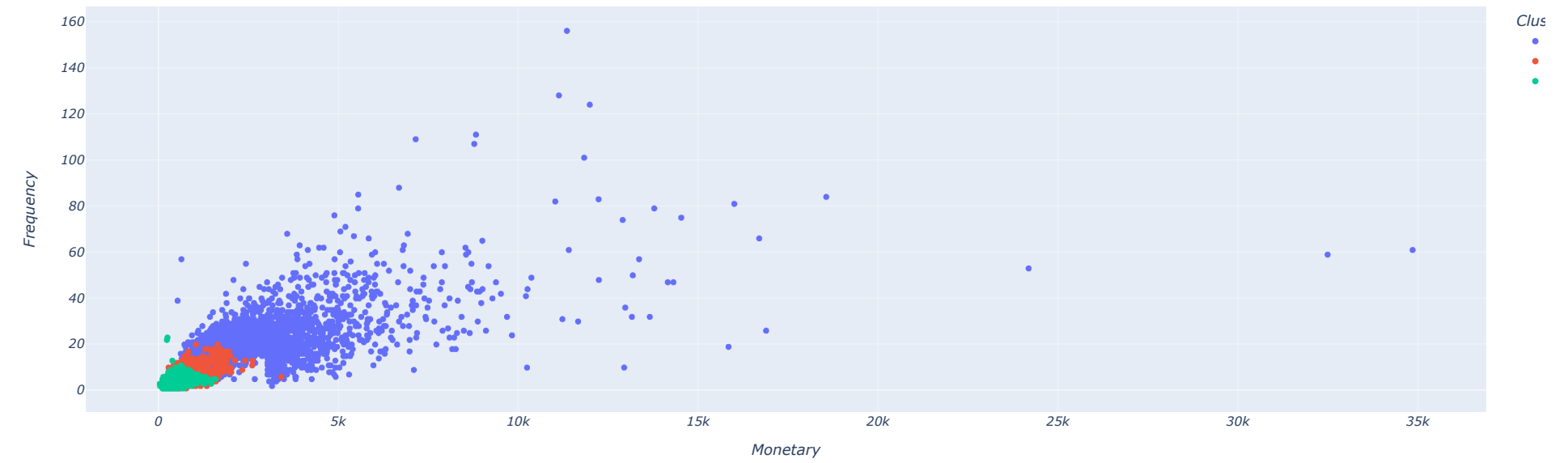
	CustomerID	Recency	Frequency	Monetary	R_score	F_score	M_score	RFM_Group	RFM_Score	Segment	Cluster
0	22	1	124	11986.54	1	1	1	111	3	champions	Cluster1
1	29	1	82	11025.96	1	1	1	111	3	champions	Cluster1
2	83	1	43	7259.69	1	1	1	111	3	champions	Cluster1
3	95	1	44	6992.27	1	1	1	111	3	champions	Cluster1
4	124	1	55	6263.44	1	1	1	111	3	champions	Cluster1

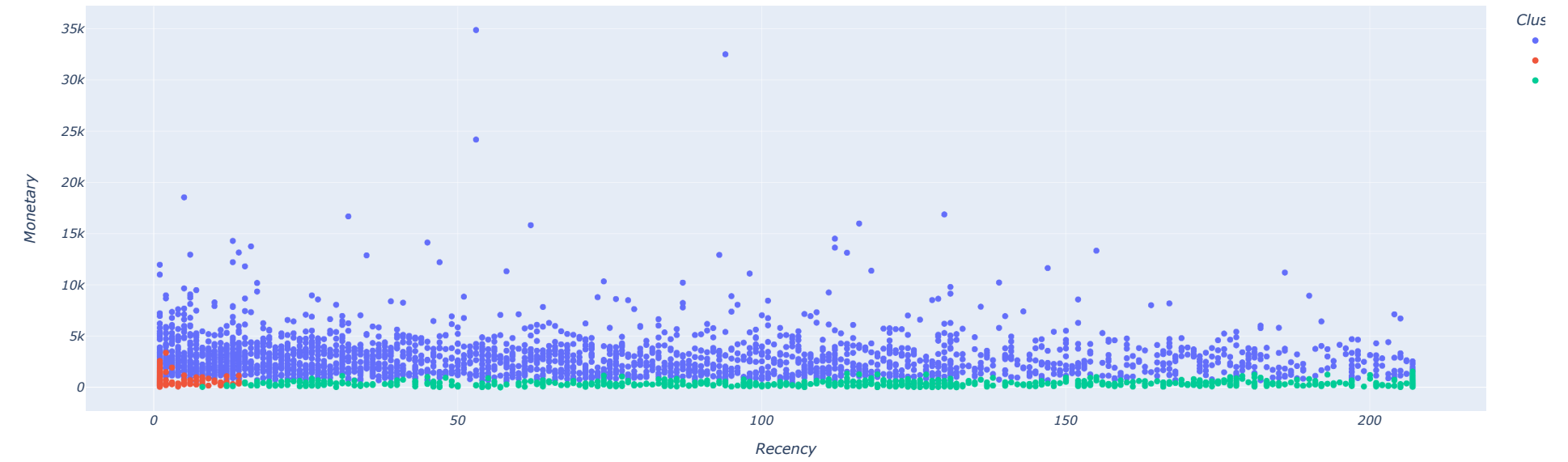
Clusters have been created based on the values of recency, frequency and monetary with the help of K-Means Clustering.

Visualization for Recency, Frequency and Monetary : Cluster Groups

In [30]:
```
# Recency V/s Frequency
fig = px.scatter(RFM_data_1,x = 'Recency',y = 'Frequency', color = 'Cluster')
fig.show()

# Frequency V/s Monetary
fig = px.scatter(RFM_data_1,x = 'Monetary',y = 'Frequency', color = 'Cluster')
fig.show()

# Recency V/s Monetary
fig = px.scatter(RFM_data_1,x = 'Recency',y = 'Monetary', color = 'Cluster')
fig.show()
```







- ***Plotting Frequency and Recency: here, the blue group frequently purchases products and they are the most recency one. The green ones tried to purchase recently but they are not frequent buyers which we can determine that they are the new customers.***
- ***Plotting Frequency and Monetary: even here the blue group of customers tried to purchase more and frequently whereas the green group is very little frequency and spends very little.***
- ***Plotting Recency and monetary: from the graph we can say that the blue group the ones who like to spend more and they are the recent customers.***

Step 8: Findings

- ***1: Most of the customer orders placed during last week of the month***
- ***2: Thursday, Friday, and Sunday are the most prominent days for customers to purchase products.***
- ***3: average recency of the customers are 87 days (approx), an average customer are purchasing the product 13 times and spending an average 1681.52 unitprice***
- ***4: The customers whose Recency is less than two month have high Frequency i.e the customers buying more when their recency is less***
- ***5: customers buying frequently are spending less amount.***
- ***6: the customers whose Recency is less than two month have high Monetory i.e the customers spending more when their recency is less.***
- ***7: almost 50% customers are spend time and buy products and around 25-30% customers are potential customers and 20-25% customers need attention.***

Step 8: Suggestions

I. Based on the above R-F-M score, we can give some Recommendations.

- ***1: Champions: We can Reward them for their multiples purchases. They can be early adopters to very new products. Suggest them "Refer a friend". Also, they can be the most loyal customers that have the habit to order.***
- ***2: Potential customers: Create loyalty cards in which they can gain points each time of purchasing and these points could transfer into a discount***
- ***3: Need attention: Send them personalized emails/messages/notifications to encourage them to order. Also Notify them about the discounts to keep them spending more and more money on your products.***

II. Based On the visualization of data :

- ***1. Allow discounts on products at the beginning of the month so that customers can start spending from week one..***
- ***2. Take survey from need attention customers about service, offers and what they required from us.***

THANK YOU...

In [ ]: