```
In [1]: # importing the libraries
```

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [3]: # Importing the data
```

```
In [4]: retail_df= pd.read_excel("Online Retail.xlsx")
```

```
In [5]: retail_df.head(10)
```

Out[5]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01 08:26:00	7.65	17850.0	United Kingdom
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	2010-12-01 08:26:00	4.25	17850.0	United Kingdom
7	536366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:28:00	1.85	17850.0	United Kingdom
8	536366	22632	HAND WARMER RED POLKA DOT	6	2010-12-01 08:28:00	1.85	17850.0	United Kingdom
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	2010-12-01 08:34:00	1.69	13047.0	United Kingdom

1. Data Inspection and data cleaning

```
In [6]: # checking the shape of the data
retail_df.shape

Out[6]: (541909, 8)
```

```
In [7]: # data info
retail_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

```
In [8]: # Checking the null values for missing value treatment

retail_df.isnull().sum()

Out[8]: InvoiceNo          0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID    135080
Country            0
dtype: int64
```

```
In [15]: # Checking the percentage of the customer ID
(retail_df["CustomerID"].isnull().sum()/len(retail_df)*100)

Out[15]: 24.926694334288598
```

The description column can be dropped as it has no use in the model

```
In [16]: retail_df=retail_df.drop(["Description"],axis=1)
```

```
In [17]: retail_df.head(5)
```

Out[17]:

	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

```
In [23]: # Missing value treatment for CustomerID
# The customer Id is an unique class and hence cannot be replaced with mean/mode/median.The null values needs t
```

```
In [28]: retail_df=(retail_df.dropna())
```

```
In [29]: retail_df.isnull().sum()

Out[29]: InvoiceNo      0
StockCode      0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

```
In [30]: # Dropping the duplicate entries in the dataset

retail_df=retail_df.drop_duplicates()
retail_df.shape

Out[30]: (401602, 7)
```

Performing Descriptive Analysis

```
In [32]: # Customer Id is of float type and this needs to changed to string type

retail_df["CustomerID"]=retail_df["CustomerID"].astype(str)
```

```
In [34]: retail_df.describe(datetime_is_numeric=True)
```

Out[34]:

	Quantity	InvoiceDate	UnitPrice
count	401602.000000	401602	401602.000000
mean	12.182579	2011-07-10 12:08:08.129839872	3.474064
min	-80995.000000	2010-12-01 08:26:00	0.000000
25%	2.000000	2011-04-06 15:02:00	1.250000
50%	5.000000	2011-07-29 15:40:00	1.950000
75%	12.000000	2011-10-20 11:58:00	3.750000
max	80995.000000	2011-12-09 12:50:00	38970.000000
std	250.283248	NaN	69.764209

Inferences

1. The minimum quantity is in negative which implies returns
2. The invoice date ranges from 2010-12-01 to 2010-12-01
3. Average price of each product is 3.47

```
In [39]: retail_df.describe(include=['O'])
```

Out[39]:

	InvoiceNo	StockCode	CustomerID	Country
count	401602	401602	401602	401602
unique	22190	3684	4372	37
top	576339	85123A	17841.0	United Kingdom
freq	542	2065	7812	356726

Inferences

1. The count of invoice numbers are 401602 while unqiue transactions are 22190.
2. The invoice no with maximum frequency of 542 is "576339"
3. The stock code of 85123A is of max frequency is 2065.
4. The country with max transaction is UK and there are total 37 countries

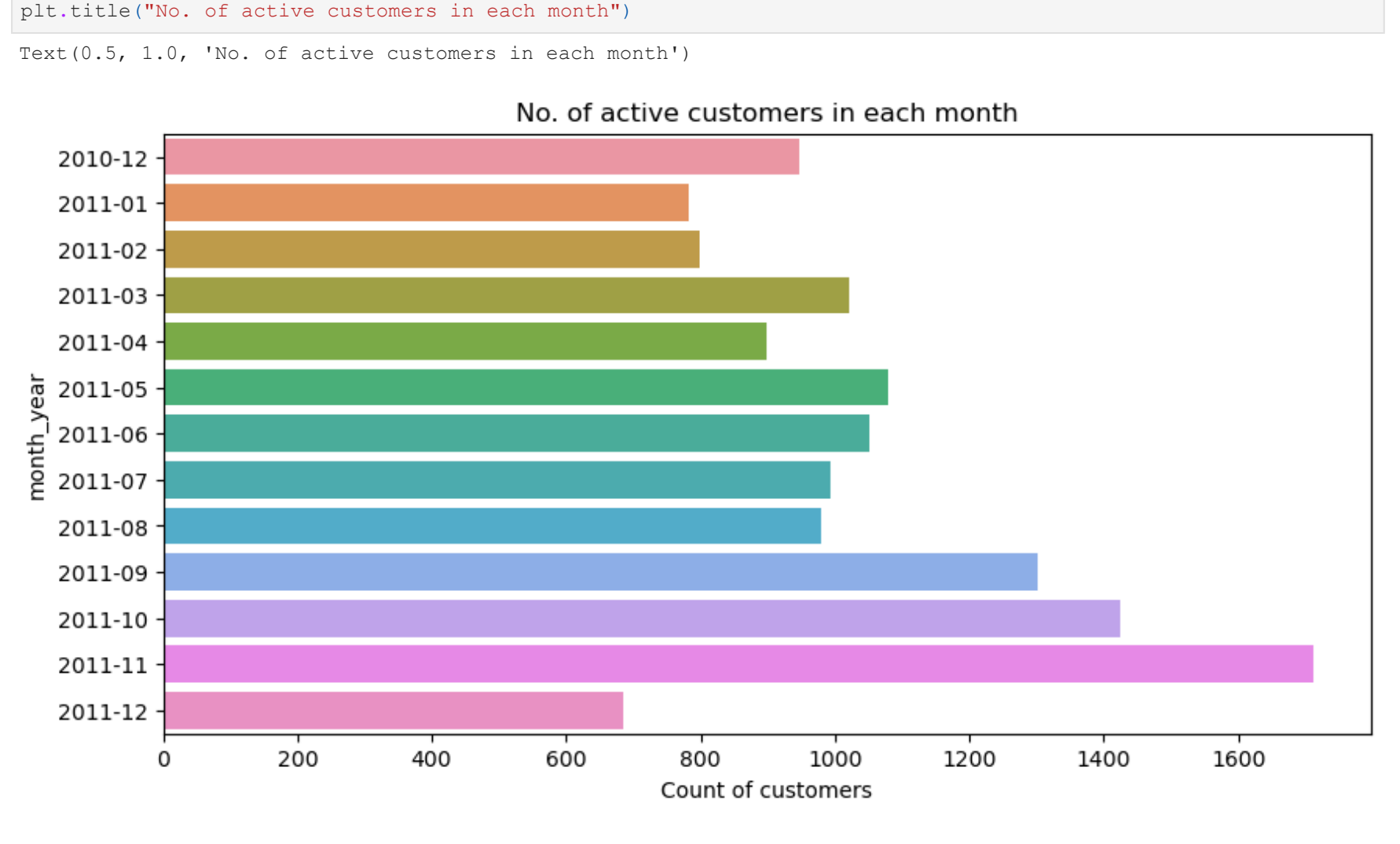
Data Transformation

Cohort Analysis **(a) Create month cohort of customers and analyze active customers in each cohort:**

```
In [41]: retail_df['month_year'] = retail_df['InvoiceDate'].dt.to_period('M')
retail_df['month_year'].nunique()

Out[41]: 13
```

```
In [42]: month_cohort = retail_df.groupby('month_year')['CustomerID'].nunique()
month_cohort

Out[42]: month_year
2010-12     948
2011-01     783
2011-02     798
2011-03    1020
2011-04     899
2011-05    1079
2011-06    1051
2011-07     993
2011-08     980
2011-09    1302
2011-10    1425
2011-11    1711
2011-12     686
Freq: M, Name: CustomerID, dtype: int64
```

```
In [43]: plt.figure(figsize=(10,5))
sns.barplot(y = month_cohort.index, x = month_cohort.values);
plt.xlabel("Count of customers")
plt.title("No. of active customers in each month")

Out[43]: Text(0.5, 1.0, 'No. of active customers in each month')
```

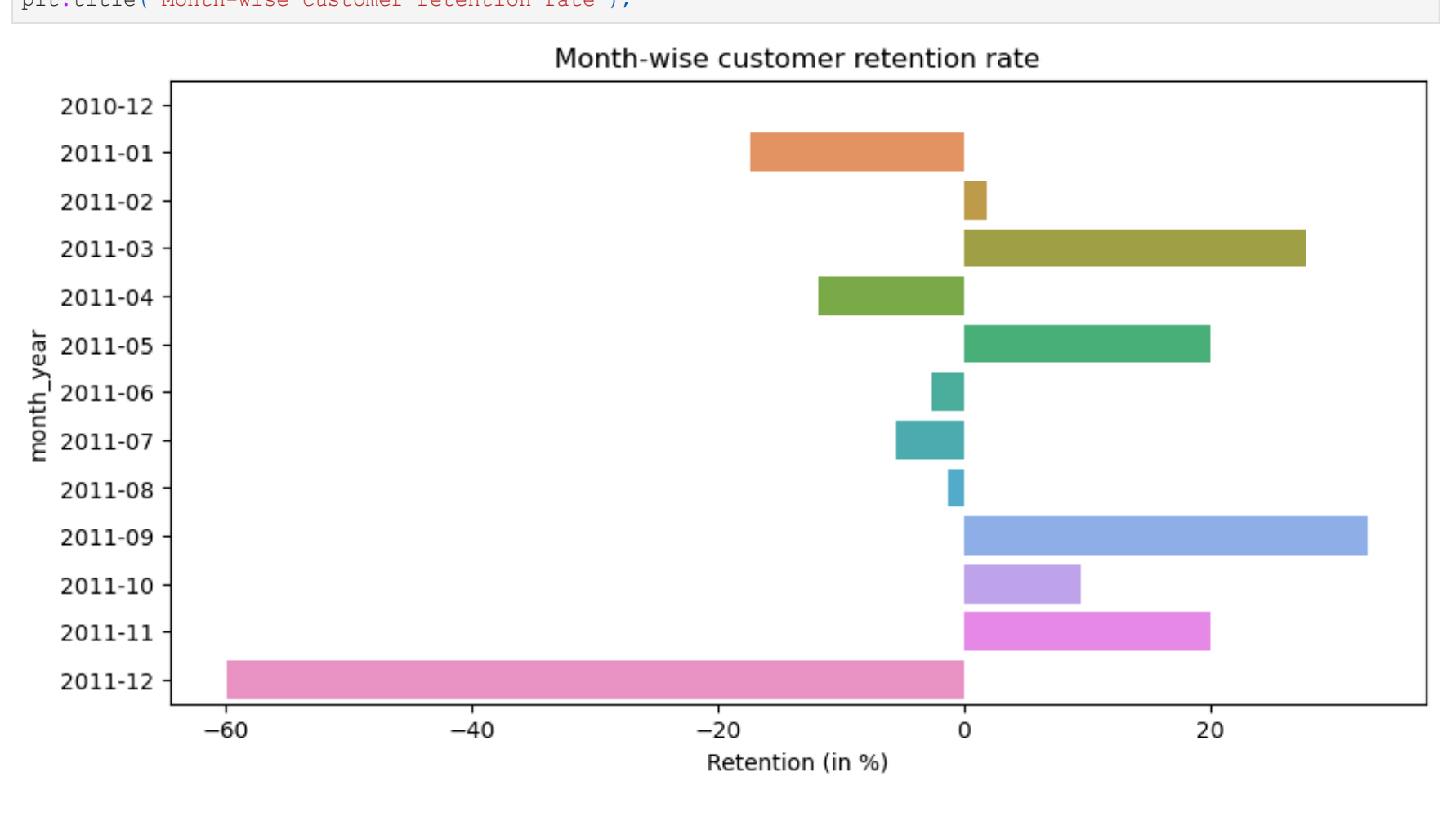


(b) Analyze the retention rate of customers:

```
In [45]: month_cohort - month_cohort.shift()

Out[45]: month_year
2010-12       NaN
2011-01    -165.0
2011-02      15.0
2011-03     222.0
2011-04    -121.0
2011-05     180.0
2011-06     -28.0
2011-07     -58.0
2011-08     -13.0
2011-09     322.0
2011-10     123.0
2011-11     286.0
2011-12   -1025.0
Freq: M, Name: CustomerID, dtype: float64
```

```
In [46]: retention_rate = round(month_cohort.pct_change(periods=1)*100,2)
retention_rate

Out[46]: month_year
2010-12      NaN
2011-01   -17.41
2011-02     1.92
2011-03    27.82
2011-04   -11.86
2011-05    20.02
2011-06    -2.59
2011-07    -5.52
2011-08    -1.31
2011-09    32.86
2011-10     9.45
2011-11    20.07
2011-12   -59.91
Freq: M, Name: CustomerID, dtype: float64
```

```
In [47]: plt.figure(figsize=(10,5))
sns.barplot(y = retention_rate.index, x = retention_rate.values);
plt.xlabel("Retention (in %)")
plt.title("Month-wise customer retention rate");
```



RFM ANALYSIS - RECENCY FREQUENCY MONETORY ANALYSIS

```
In [ ]: # MONETORY ANALYSIS
```

```
In [50]: retail_df['amount'] = retail_df['Quantity']*retail_df['UnitPrice']
retail_df.head()
```

Out[50]:

	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	month_year	amount
0	536365	85123A	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12	15.30
1	536365	71053	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12	20.34
2	536365	84406B	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010-12	22.00
3	536365	84029G	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12	20.34
4	536365	84029E	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12	20.34

```
In [51]: df_monetary = retail_df.groupby('CustomerID').sum()['amount'].reset_index()
df_monetary
```

Out[51]:

	CustomerID	amount
0	12346.0	0.00
1	12347.0	4310.00
2	12348.0	1797.24
3	12349.0	1757.55
4	12350.0	334.40
...	...	...
4367	18280.0	180.60
4368	18281.0	80.82
4369	18282.0	176.60
4370	18283.0	2045.53
4371	18287.0	1837.28

4372 rows × 2 columns

```
In [ ]: # FREQUENCY ANALYSIS
```

```
In [53]: df_frequency = retail_df.groupby('CustomerID').nunique()['InvoiceNo'].reset_index()
df_frequency
```

Out[53]:

	CustomerID	InvoiceNo
0	12346.0	2
1	12347.0	7
2	12348.0	4
3	12349.0	1
4	12350.0	1
...	...	...
4367	18280.0	1
4368	18281.0	1
4369	18282.0	3
4370	18283.0	16
4371	18287.0	3

4372 rows × 2 columns

```
In [48]: # RECENCY ANALYSIS
```

```
In [55]: # We will fix reference date for calculating recency as last transaction day in data + 1 day
from datetime import timedelta
ref_day = max(retail_df['InvoiceDate']) + timedelta(days=1)
retail_df['days_to_last_order'] = (ref_day - retail_df['InvoiceDate']).dt.days
retail_df.head()
```

Out[55]:

	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	month_year	amount	days_to_last_order
0	536365	85123A	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12	15.30	374
1	536365	71053	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12	20.34	374
2	536365	84406B	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010-12	22.00	374
3	536365	84029G	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12	20.34	374
4	536365	84029E	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12	20.34	374

```
In [61]: df_recency = retail_df.groupby('CustomerID')['days_to_last_order'].min().reset_index()
df_recency
```

Out[61]:

	CustomerID	days_to_last_order
0	12346.0	326
1	12347.0	2
2	12348.0	75
3	12349.0	19
4	12350.0	310
...	...	...
4367	18280.0	278
4368	18281.0	181
4369	18282.0	8
4370	18283.0	4
4371	18287.0	43

4372 rows × 2 columns

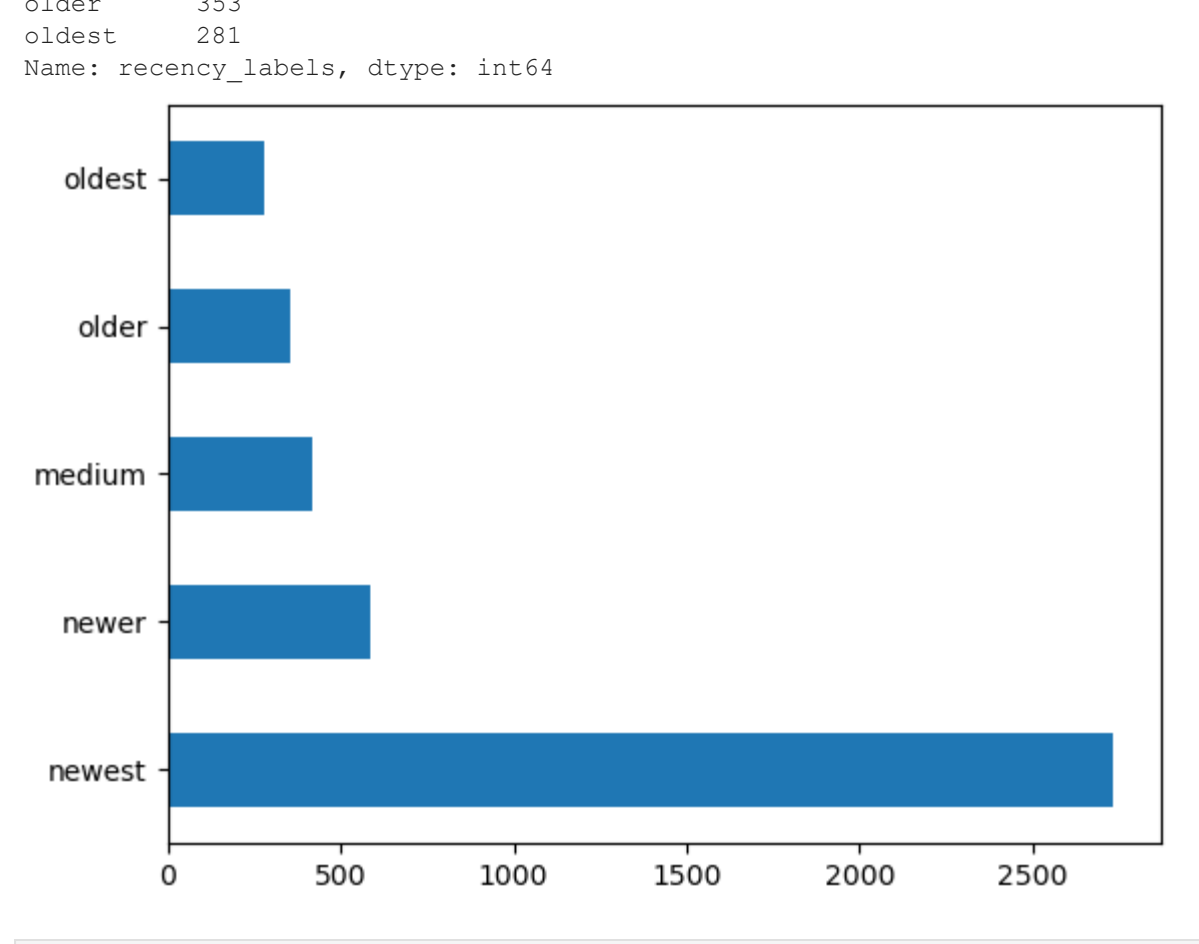
Calculating RFM Matrix

```
In [63]: df_rf = pd.merge(df_recency, df_frequency,  on='CustomerID', how='inner')
df_rfm = pd.merge(df_rf, df_monetary, on='CustomerID', how='inner')
df_rfm.columns = ['CustomerID', 'Recency', 'Frequency', 'Monetary']
df_rfm.head()
```

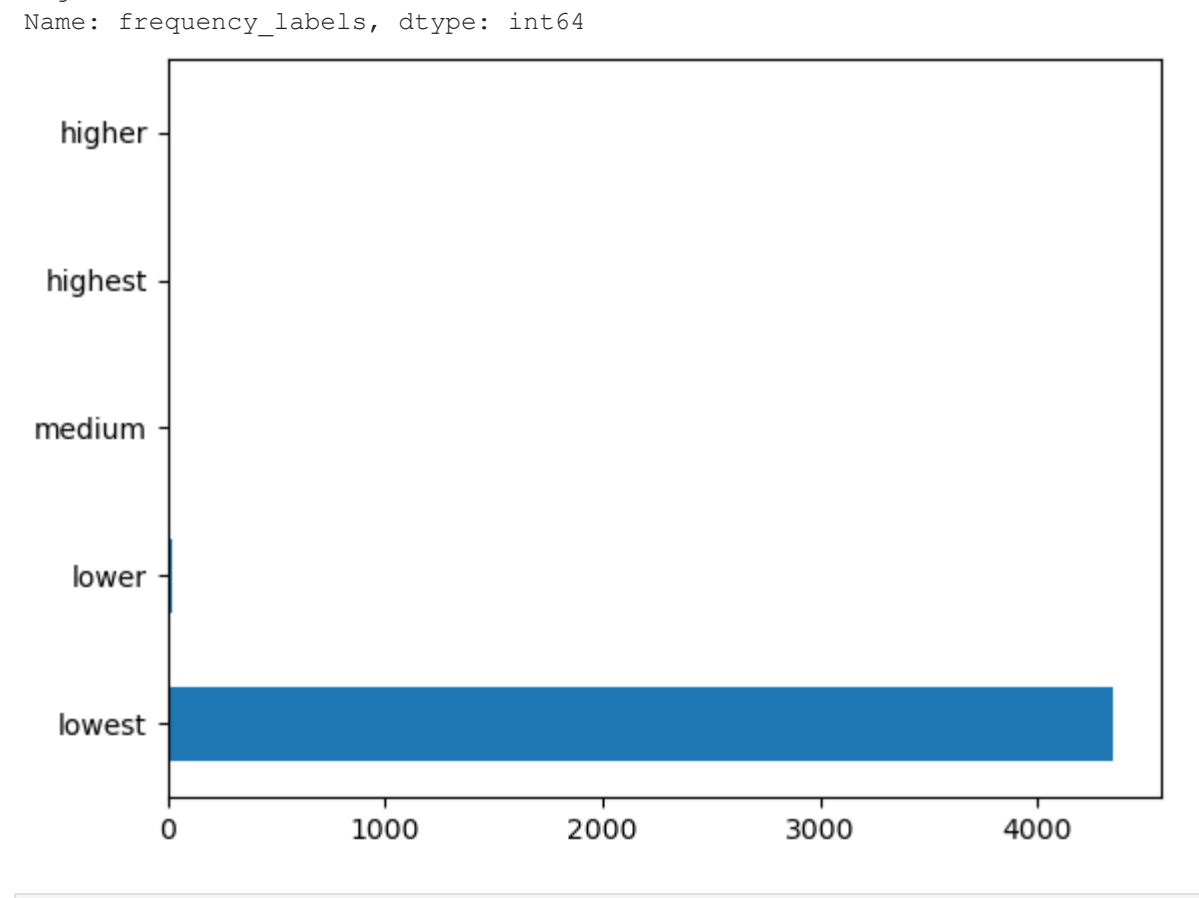
Out[63]:

	CustomerID	Recency	Frequency	Monetary
0	12346.0	326	2	0.00
1	12347.0	2	7	4310.00
2	12348.0	75	4	1797.24
3	12349.0	19	1	1757.55
4	12350.0	310	1	334.40

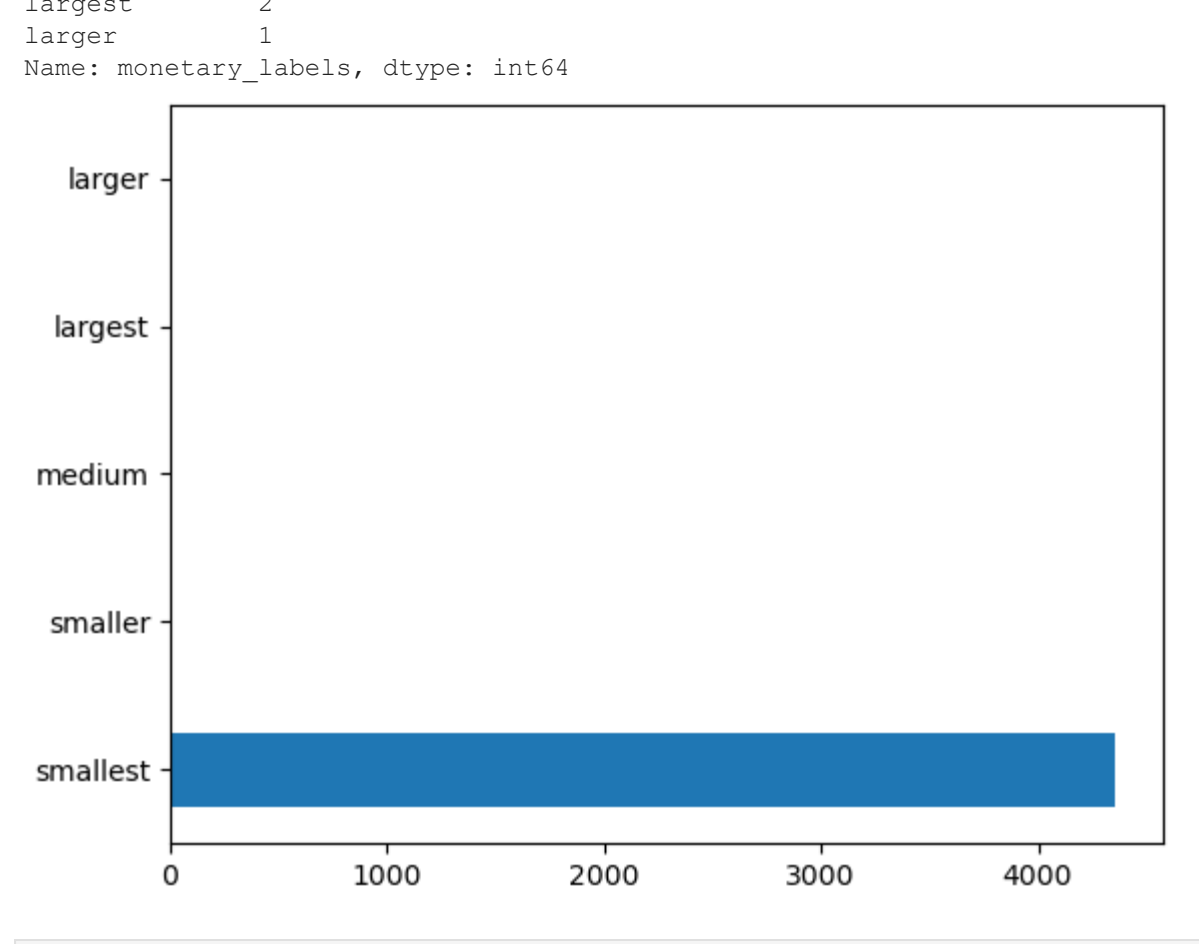
```
In [64]: df_rfm['recency_labels'] = pd.cut(df_rfm['Recency'], bins=5,
                                  labels=['newest', 'newer', 'medium', 'older', 'oldest'])
df_rfm['recency_labels'].value_counts().plot(kind='barh');
df_rfm['recency_labels'].value_counts()

Out[64]: newest    2734
newer      588
medium     416
older      353
oldest     281
Name: recency_labels, dtype: int64
```




```
In [65]: df_rfm['frequency_labels'] = pd.cut(df_rfm['Frequency'], bins=5, labels=['lowest', 'lower', 'medium', 'higher',
df_rfm['frequency_labels'].value_counts().plot(kind='barh');
df_rfm['frequency_labels'].value_counts()

Out[65]: lowest     4348
lower        18
medium        3
highest       2
higher        1
Name: frequency_labels, dtype: int64
```




```
In [66]: df_rfm['monetary_labels'] = pd.cut(df_rfm['Monetary'], bins=5, labels=['smallest', 'smaller', 'medium', 'larger
df_rfm['monetary_labels'].value_counts().plot(kind='barh');
df_rfm['monetary_labels'].value_counts()

Out[66]: smallest    4357
smaller        9
medium         3
largest        2
larger         1
Name: monetary_labels, dtype: int64
```




```
In [67]: df_rfm['rfm_segment'] = df_rfm[['recency_labels','frequency_labels','monetary_labels']].agg('-'.join, axis=1)
df_rfm.head()
```

Out[67]:

	CustomerID	Recency	Frequency	Monetary	recency_labels	frequency_labels	monetary_labels	rfm_segment
0	12346.0	326	2	0.00	oldest	lowest	smallest	oldest-lowest-smallest
1	12347.0	2	7	4310.00	newest	lowest	smallest	newest-lowest-smallest
2	12348.0	75	4	1797.24	newest	lowest	smallest	newest-lowest-smallest
3	12349.0	19	1	1757.55	newest	lowest	smallest	newest-lowest-smallest
4	12350.0	310	1	334.40	oldest	lowest	smallest	oldest-lowest-smallest

RFM SCORE

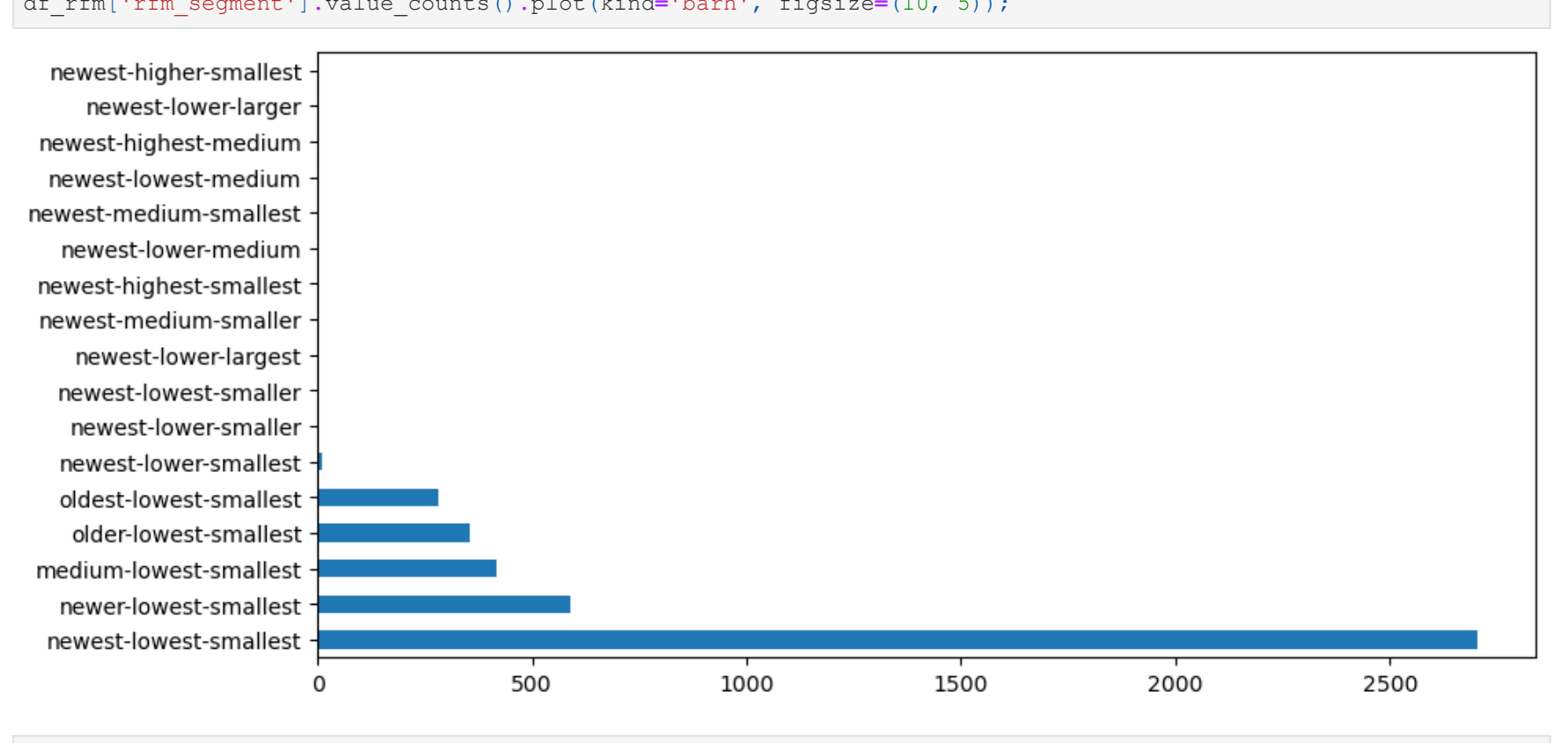
```
In [69]: recency_dict = {'newest': 5, 'newer':4, 'medium': 3, 'older':2, 'oldest':1}
frequency_dict = {'lowest':1, 'lower':2, 'medium': 3, 'higher':4, 'highest':5}
monetary_dict = {'smallest':1, 'smaller':2, 'medium': 3, 'larger':4, 'largest':5}

df_rfm['rfm_score'] = df_rfm['recency_labels'].map(recency_dict).astype(int)+ df_rfm['frequency_labels'].map(fr
df_rfm.head(10)
```

Out[69]:

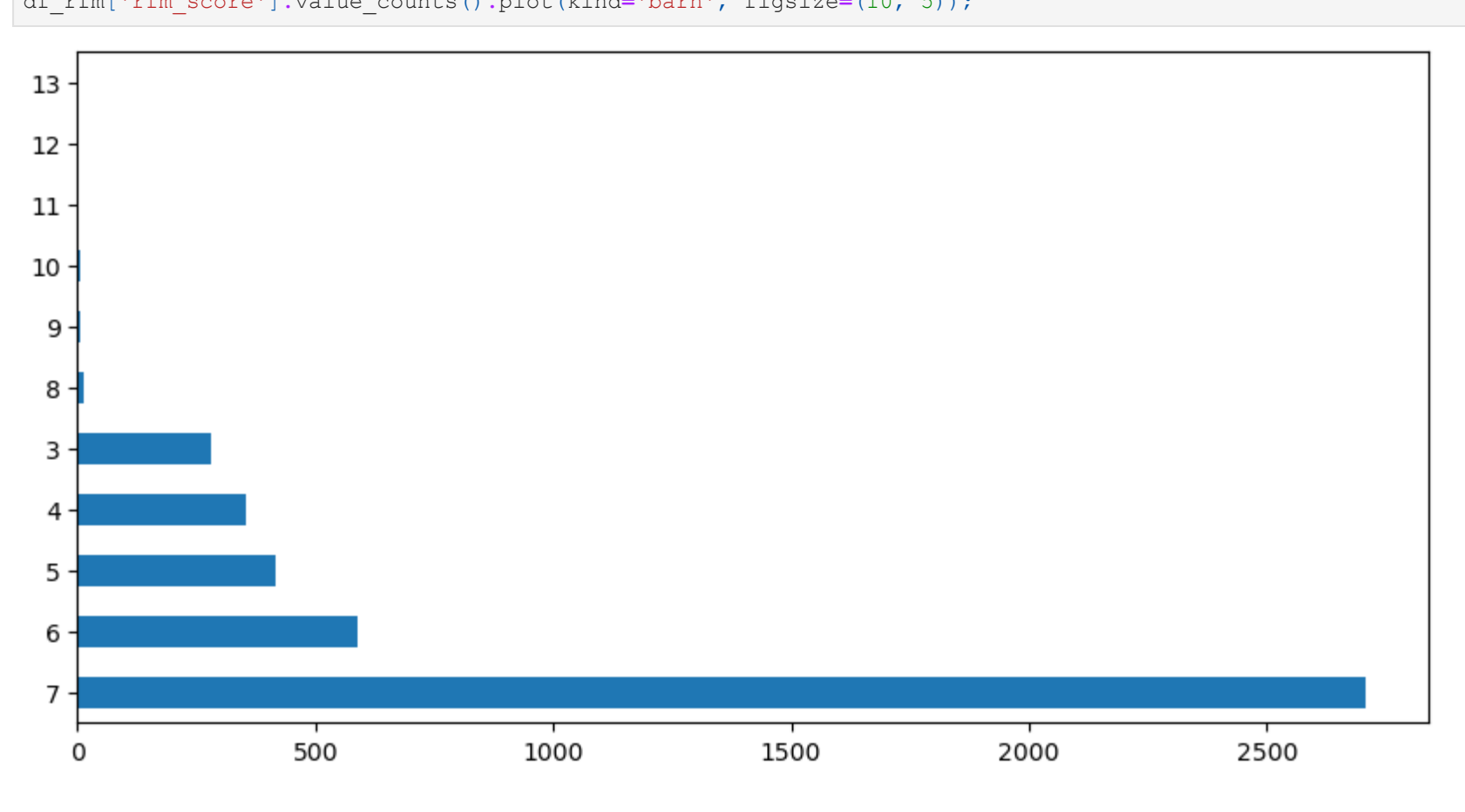
	CustomerID	Recency	Frequency	Monetary	recency_labels	frequency_labels	monetary_labels	rfm_segment	rfm_score
0	12346.0	326	2	0.00	oldest	lowest	smallest	oldest-lowest-smallest	3
1	12347.0	2	7	4310.00	newest	lowest	smallest	newest-lowest-smallest	7
2	12348.0	75	4	1797.24	newest	lowest	smallest	newest-lowest-smallest	7
3	12349.0	19	1	1757.55	newest	lowest	smallest	newest-lowest-smallest	7
4	12350.0	310	1	334.40	oldest	lowest	smallest	oldest-lowest-smallest	3
5	12352.0	36	11	1545.41	newest	lowest	smallest	newest-lowest-smallest	7
6	12353.0	204	1	89.00	medium	lowest	smallest	medium-lowest-smallest	5
7	12354.0	232	1	1079.40	older	lowest	smallest	older-lowest-smallest	4
8	12355.0	214	1	459.40	medium	lowest	smallest	medium-lowest-smallest	5
9	12356.0	23	3	2811.43	newest	lowest	smallest	newest-lowest-smallest	7

Analyze RFM Segment and Score:

```
In [71]: df_rfm['rfm_segment'].value_counts().plot(kind='barh', figsize=(10, 5));
```




```
In [72]: df_rfm['rfm_score'].value_counts().plot(kind='barh', figsize=(10, 5));
```



Data Modeling:

Create clusters using k-means clustering algorithm.

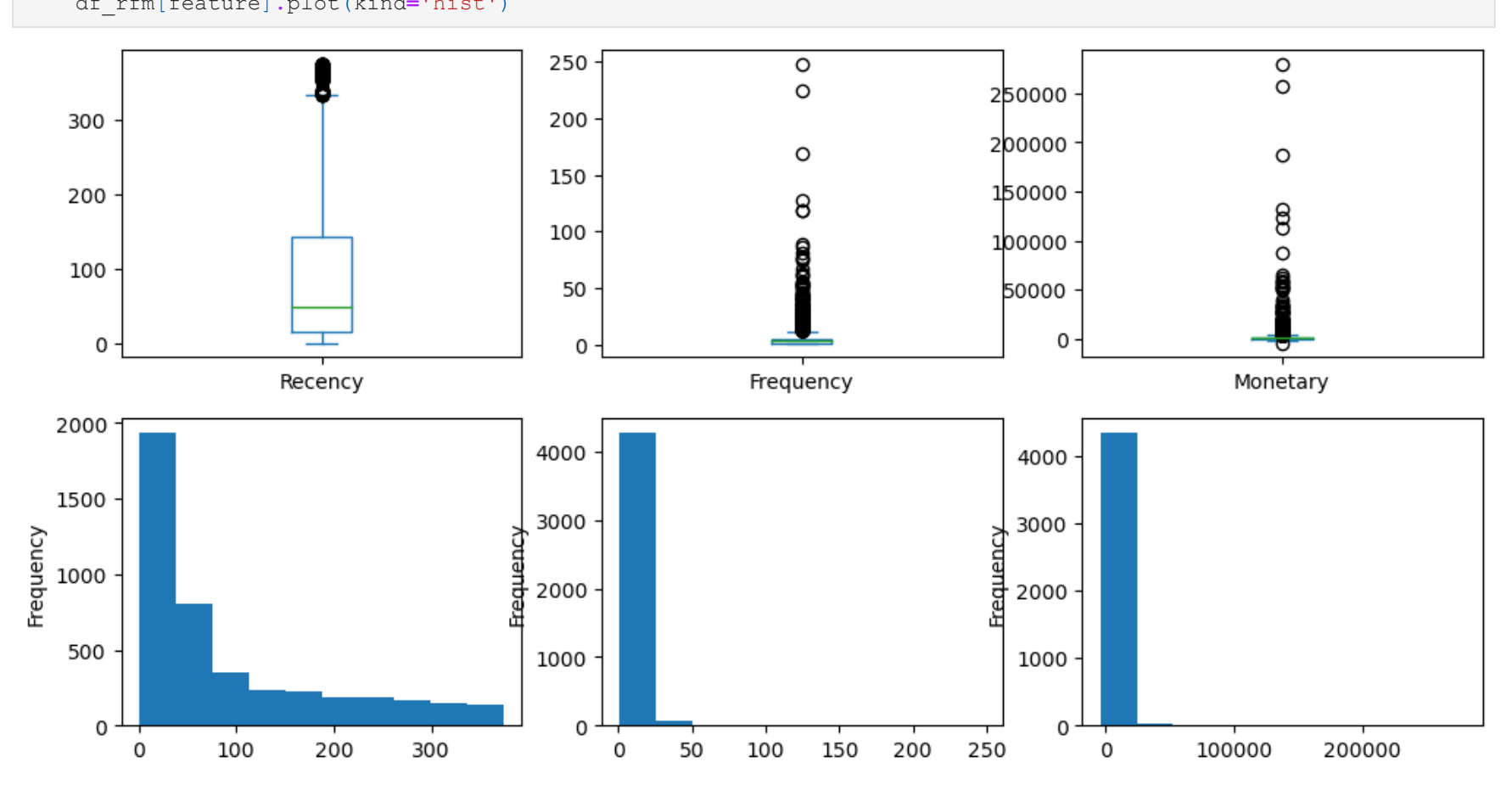
a. Prepare the data for the algorithm. If the data is asymmetrically distributed, manage the skewness with appropriate transformation. Standardize the data.

```
In [73]: print(df_rfm.shape)
df_rfm.head()

(4372, 9)
```

Out[73]:

	CustomerID	Recency	Frequency	Monetary	recency_labels	frequency_labels	monetary_labels	rfm_segment	rfm_score
0	12346.0	326	2	0.00	oldest	lowest	smallest	oldest-lowest-smallest	3
1	12347.0	2	7	4310.00	newest	lowest	smallest	newest-lowest-smallest	7
2	12348.0	75	4	1797.24	newest	lowest	smallest	newest-lowest-smallest	7
3	12349.0	19	1	1757.55	newest	lowest	smallest	newest-lowest-smallest	7
4	12350.0	310	1	334.40	oldest	lowest	smallest	oldest-lowest-smallest	3

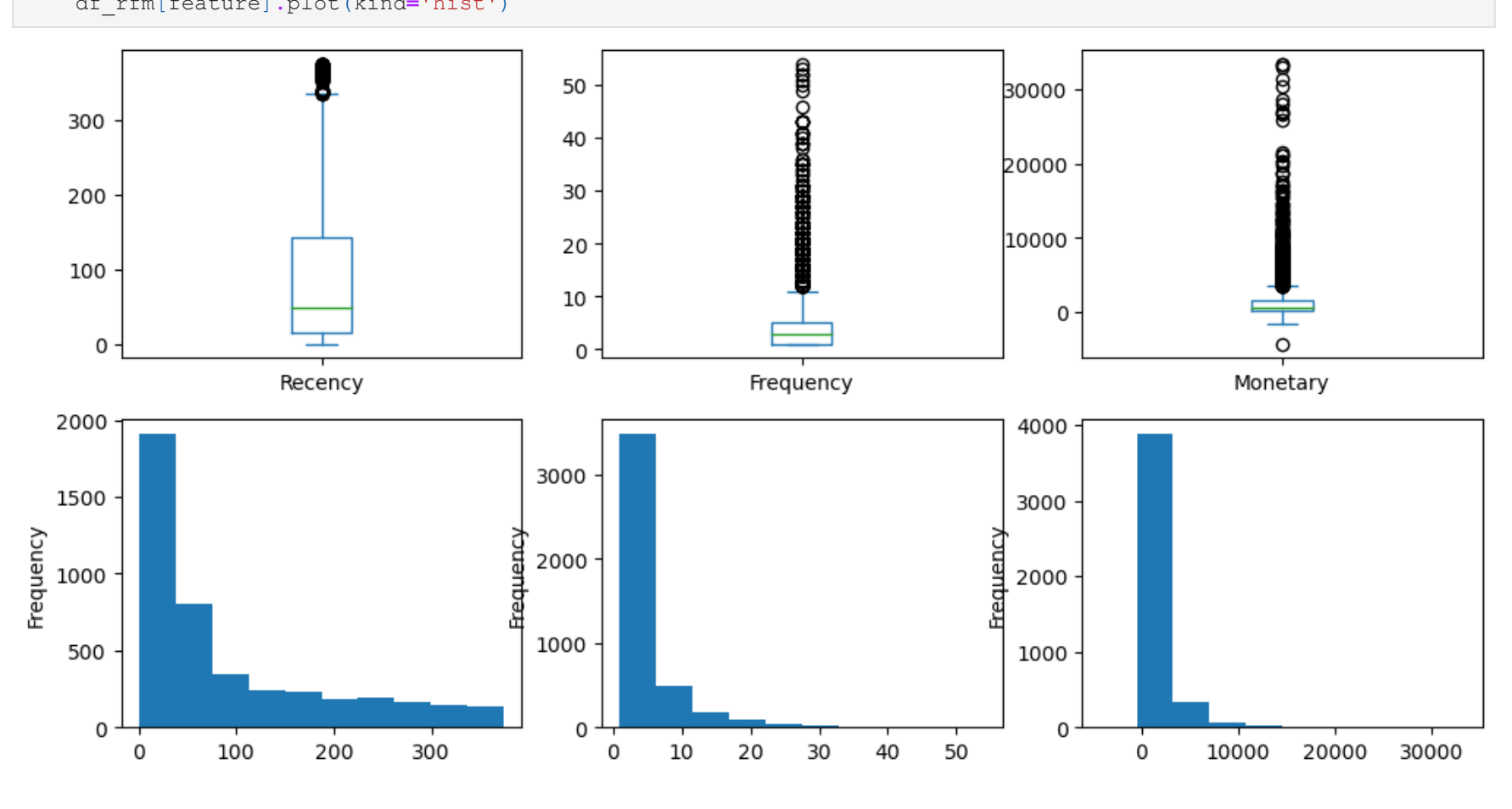
```
In [74]: plt.figure(figsize=(12,6))

for i, feature in enumerate(['Recency', 'Frequency', 'Monetary']):
    plt.subplot(2,3,i+1)
    df_rfm[feature].plot(kind='box')
    plt.subplot(2,3,i+1+3)
    df_rfm[feature].plot(kind='hist')
```



Outliers: Frequency and Monetary features in above data seem to have lot of outliers. They should be dropped.

```
In [75]: df_rfm = df_rfm[(df_rfm['Frequency']<60) & (df_rfm['Monetary']<40000)]
df_rfm.shape

Out[75]: (4346, 9)
```

```
In [76]: plt.figure(figsize=(12,6))

for i, feature in enumerate(['Recency', 'Frequency', 'Monetary']):
    plt.subplot(2,3,i+1)
    df_rfm[feature].plot(kind='box')
    plt.subplot(2,3,i+1+3)
    df_rfm[feature].plot(kind='hist')
```



Standard scalar Transformation

```
In [83]: from sklearn.preprocessing import StandardScaler
```

```
In [85]: scaler = StandardScaler()
```

```
In [86]: df_rfm_scaled = scaler.fit_transform(df_rfm_log_trans[['Recency', 'Frequency', 'Monetary']])
df_rfm_scaled

df_rfm_scaled = pd.DataFrame(df_rfm_scaled)
df_rfm_scaled.columns = ['Recency', 'Frequency', 'Monetary']
df_rfm_scaled.head()
```

Out[86]:

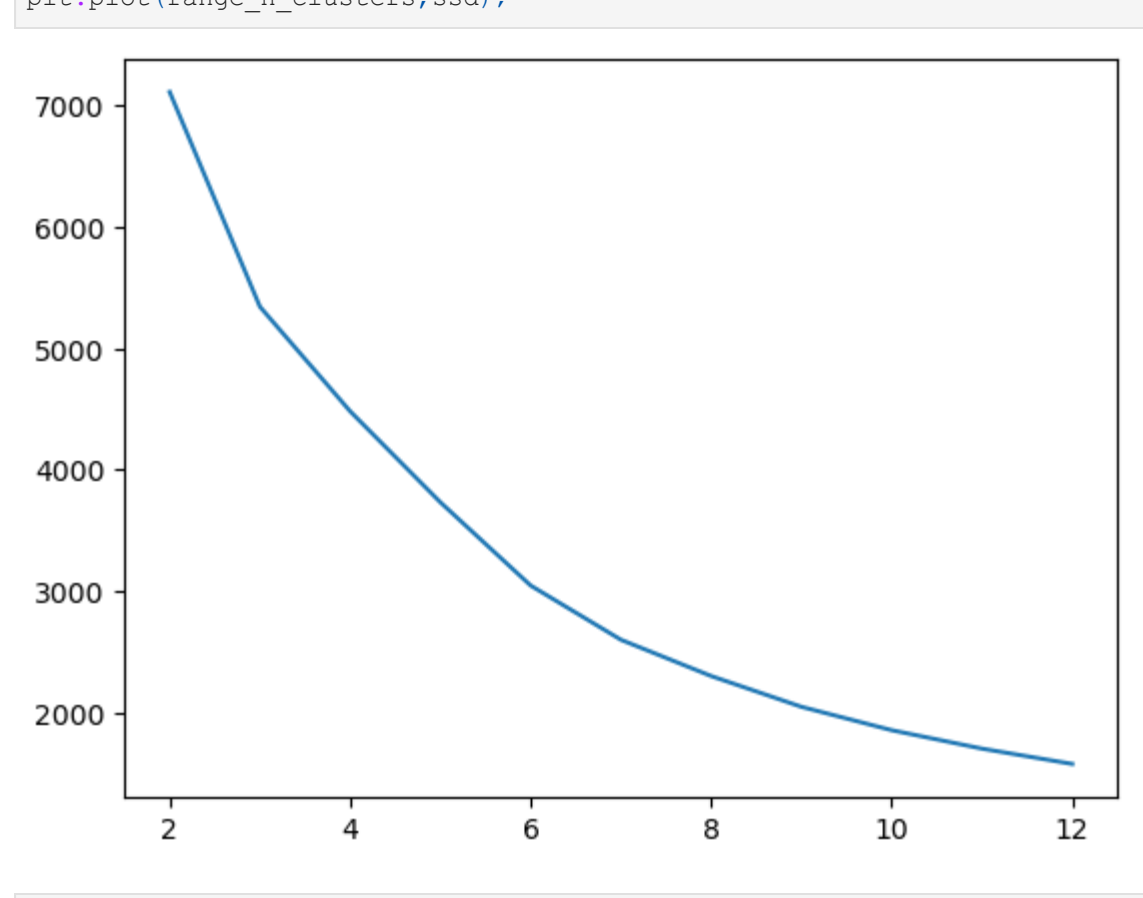
	Recency	Frequency	Monetary
0	1.402988	-0.388507	-0.770922
1	-2.100874	0.967301	1.485132
2	0.392218	0.361655	0.364190
3	-0.552268	-1.138669	0.342970
4	1.368370	-1.138669	-0.527416

b. Build K-Means Clustering Model and Decide the optimum number of clusters to be formed.

```
In [89]: # k-means with some arbitrary k
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
kmeans = KMeans(n_clusters=3, max_iter=50)
kmeans.fit(df_rfm_scaled)

Out[89]: KMeans(max_iter=50, n_clusters=3)
```

```
In [90]: kmeans.labels_

Out[90]: array([1, 2, 0, ..., 0, 2, 0])
```

```
In [91]: # Finding the Optimal Number of Clusters with the help of Elbow Curve/ SSD
ssd = []
range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=100)
    kmeans.fit(df_rfm_scaled)

    ssd.append(kmeans.inertia_)

# plot the SSDs for each n_clusters
plt.plot(range_n_clusters,ssd);
```



```
In [92]: # Creating dataframe for exporting to create visualization in tableau later
df_inertia = pd.DataFrame(list(zip(range_n_clusters, ssd)), columns=['clusters', 'intertia'])
df_inertia
```

Out[92]:

	clusters	intertia
0	2	7113.097396
1	3	5343.153295
2	4	4480.998412
3	5	3730.755397
4	6	3044.935139
5	7	2598.297835
6	8	2299.260226
7	9	2045.630814
8	10	1852.933027
9	11	1701.069737
10	12	1574.969301

In [93]:

```
# Finding the Optimal Number of Clusters with the help of Silhouette Analysis
range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10]

for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(df_rfm_scaled)

    cluster_labels = kmeans.labels_

    silhouette_avg = silhouette_score(df_rfm_scaled, cluster_labels)
    print("For n_clusters={0}, the silhouette score is {1}".format(num_clusters, silhouette_avg))
```

```
For n_clusters=2, the silhouette score is 0.44132753537785846
For n_clusters=3, the silhouette score is 0.3802926901795938
For n_clusters=4, the silhouette score is 0.3623606426972478
For n_clusters=5, the silhouette score is 0.3648879010841238
For n_clusters=6, the silhouette score is 0.3442420226114096
For n_clusters=7, the silhouette score is 0.34297910328413955
For n_clusters=8, the silhouette score is 0.3356899635770755
For n_clusters=9, the silhouette score is 0.3471822122915281
For n_clusters=10, the silhouette score is 0.35608282484485043
```

In [94]:

```
# Final model with k=3
kmeans = KMeans(n_clusters=3, max_iter=50)
kmeans.fit(df_rfm_scaled)
```

Out[94]:

```
KMeans(max_iter=50, n_clusters=3)
```

c. Analyze these clusters and comment on the results.

In [95]:

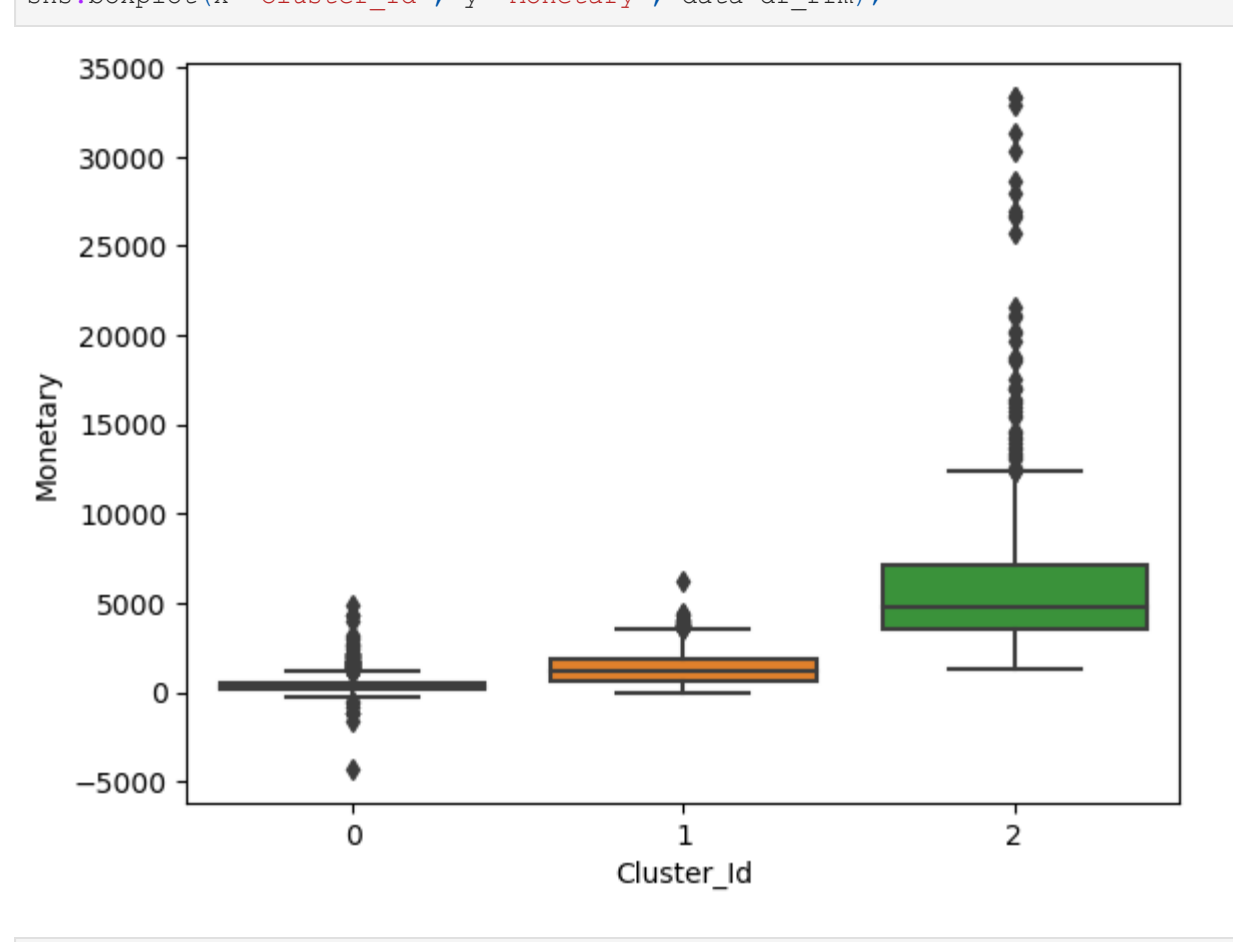
```
# assign the label
df_rfm['Cluster_Id'] = kmeans.labels_
df_rfm.head()
```

Out[95]:

	CustomerID	Recency	Frequency	Monetary	recency_labels	frequency_labels	monetary_labels	rfm_segment	rfm_score	Cluster_Id
0	12346.0	326	2	0.00	oldest	lowest	smallest	oldest-lowest-smallest	3	0
1	12347.0	2	7	4310.00	newest	lowest	smallest	newest-lowest-smallest	7	2
2	12348.0	75	4	1797.24	newest	lowest	smallest	newest-lowest-smallest	7	1
3	12349.0	19	1	1757.55	newest	lowest	smallest	newest-lowest-smallest	7	0
4	12350.0	310	1	334.40	oldest	lowest	smallest	oldest-lowest-smallest	3	0

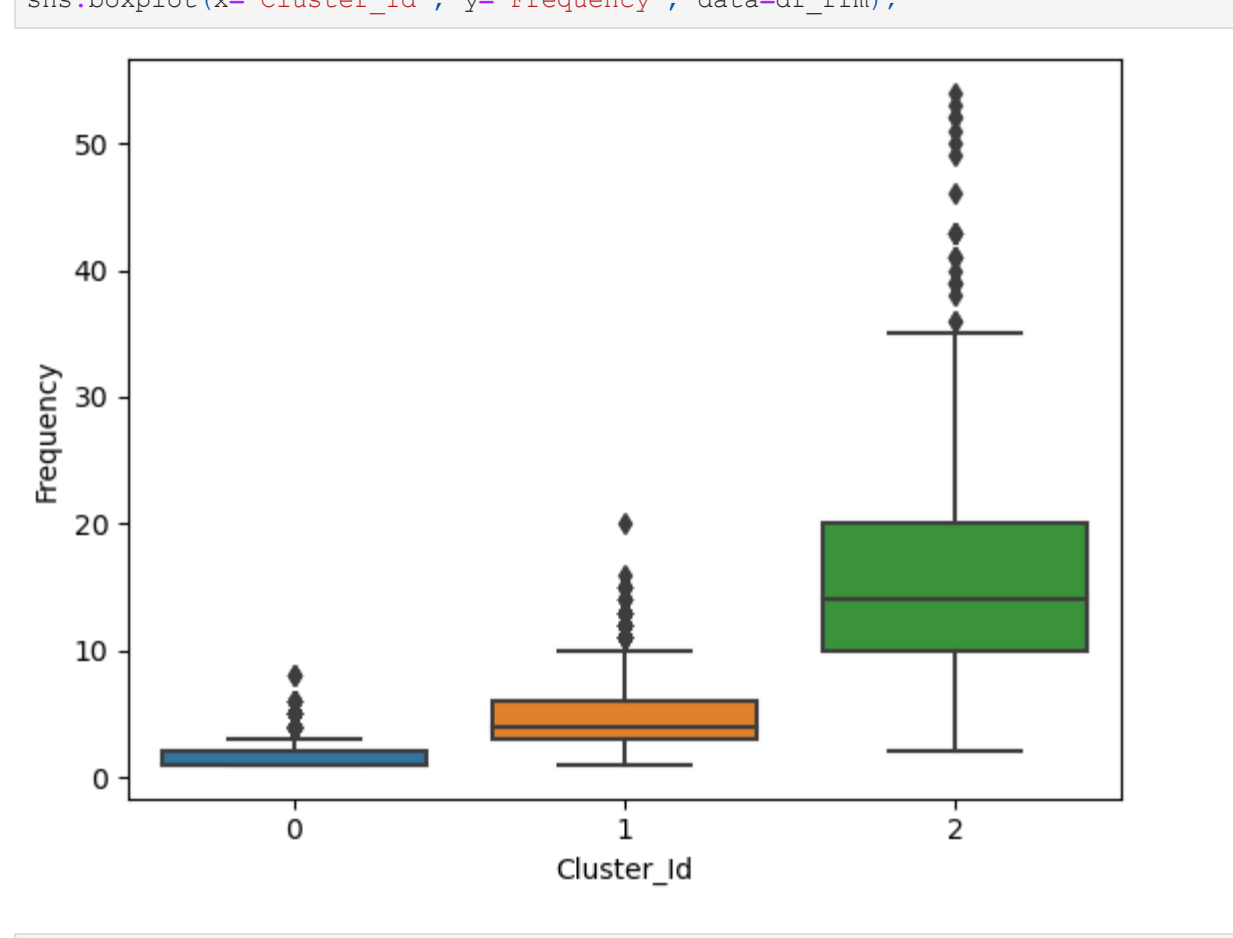
In [96]:

```
# Box plot to visualize Cluster Id vs Monetary
sns.boxplot(x='Cluster_Id', y='Monetary', data=df_rfm);
```



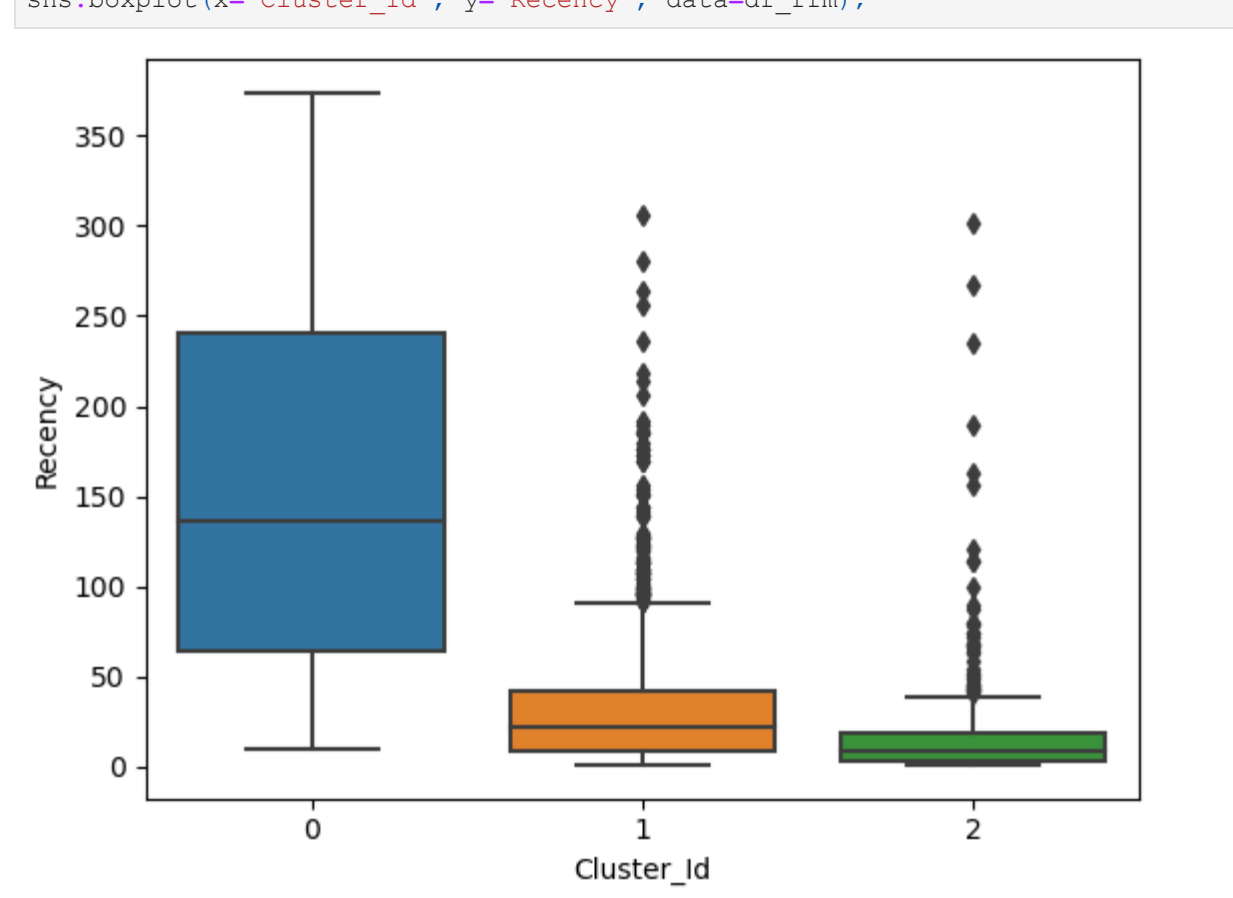
In [97]:

```
# Box plot to visualize Cluster Id vs Frequency
sns.boxplot(x='Cluster_Id', y='Frequency', data=df_rfm);
```



In [98]:

```
# Box plot to visualize Cluster Id vs Recency
sns.boxplot(x='Cluster_Id', y='Recency', data=df_rfm);
```



Inference:

As we can observe from above boxplots that our model has nicely created 3 segements of customer with the interpretation as below:

1. Customers with Cluster Id 0 are less frequent buyers with low monetary expenditure and also they have not purchased anything in recent time and hence least important for business.
2. Customers with Cluster Id 1 are the customers having Recency, Frequency and Monetary score in the medium range.
3. Customers with Cluster Id 2 are the most frequent buyers, spending high amount and recently placing orders so they are the most important customers from business point of view.

Data Reporting:

Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

1. Country-wise analysis to demonstrate average spend. Use a bar chart to show the monthly figures
2. Bar graph of top 15 products which are mostly ordered by the users to show the number of products sold
3. Bar graph to show the count of orders vs. hours throughout the day
4. Plot the distribution of RFM values using histogram and frequency charts
5. Plot error (cost) vs. number of clusters selected
6. Visualize to compare the RFM values of the clusters using heatmap

In [105…

```
# Writing dataframe to excel file for creating visualization in tableau
writer = pd.ExcelWriter('C:\\Users\\rishabh\\Desktop\\Data Science\\Project\\Capstone Project\\Online Retail\\P
retail_df.to_excel(writer, sheet_name='master_data', index=False)
df_rfm.to_excel(writer, sheet_name='rfm_data', index=False)
df_inertia.to_excel(writer, sheet_name='inertia', index=False)
writer.save()
```

In [ ]:

In [ ]: