In [4]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:

```
df=pd.read_excel("Online Retail.xlsx")
```

In [3]:

```
df.head(10)
```

Out[3]:

	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

In [5]:

```
df.isna().sum()
```

Out[5]:

```
InvoiceNo          0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID     135080
Country            0
dtype: int64
```

In [6]:

```
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 [9]:

```
# Calculating the Missing Values % contribution in DF
df_null = round(df.isnull().sum()/len(df)*100,2)
df_null
```

Out[9]:

```
InvoiceNo       0.00
StockCode       0.00
Description     0.27
Quantity        0.00
InvoiceDate     0.00
UnitPrice       0.00
CustomerID     24.93
Country         0.00
dtype: float64
```

In [10]:

```
# We can drop Description feature from our data since it is not not going to contribute in our model.
df.drop('Description',axis=1,inplace=True)
```

In [12]:

```
df.shape
```

Out[12]:

```
(541909, 7)
```

In [29]:

```
df=df.dropna()
```

In [30]:

```
df.isna().sum()
```

Out[30]:

```
InvoiceNo      0
StockCode      0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

In [32]:

```
df.describe().transpose()
```

Out[32]:

	count	mean	std	min	25%	50%	75%	max
Quantity	406829.0	12.061303	248.693370	-80995.0	2.00	5.00	12.00	80995.0
UnitPrice	406829.0	3.460471	69.315162	0.0	1.25	1.95	3.75	38970.0
CustomerID	406829.0	15287.690570	1713.600303	12346.0	13953.00	15152.00	16791.00	18287.0

In [37]:

```
df['CustomerID']=df['CustomerID'].astype(str)
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_2164\2503539684.py:1: 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#returni
ng-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  df['CustomerID']=df['CustomerID'].astype(str)
```

In [38]:

```
# To get derciptive analysis of object and string variables
df.describe(include=['O'])
```

Out[38]:

	InvoiceNo	StockCode	CustomerID	Country
count	406829	406829	406829	406829
unique	22190	3684	4372	37
top	576339	85123A	17841.0	United Kingdom
freq	542	2077	7983	361878

1. InvoiceNo: Total entries in preprocessed data are 4,06,829 but transactions are 22,190. Most number of entries (count of unique products) are in Invoice No. '576339' and is 542 nos.
2. StockCode: There are total 3684 unique products in our data and product with stock code '85123A' appears most frequently (2077 times) in our data.
3. CustomerID: There are 4372 unique customers in our final preprocessed data. Customer with ID '17841' appears most frequently in data (7983 times)
4. Country: Company has customers across 37 countries. Most entries are from United Kingdom in our dataset (361878)

1. Performing Cohort Analysis

a. Creating month cohorts and analyzing active customers for each cohort.

In [39]:

```
# Convert to InvoiceDate to Year-Month format

df['month_year']=df['InvoiceDate'].dt.to_period('M')
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_2164\1745575587.py:1: 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#returni
ng-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  df['month_year']=df['InvoiceDate'].dt.to_period('M')
```

In [42]:

```
df['month_year']
```

Out[42]:

```
0         2010-12
1         2010-12
2         2010-12
3         2010-12
4         2010-12
           ...
541904    2011-12
541905    2011-12
541906    2011-12
541907    2011-12
541908    2011-12
Name: month_year, Length: 406829, dtype: period[M]
```

In [48]:

```
# Create month cohort

month_cohort= df.groupby('month_year')['CustomerID'].nunique()
```

In [49]:

```
month_cohort
```

Out[49]:

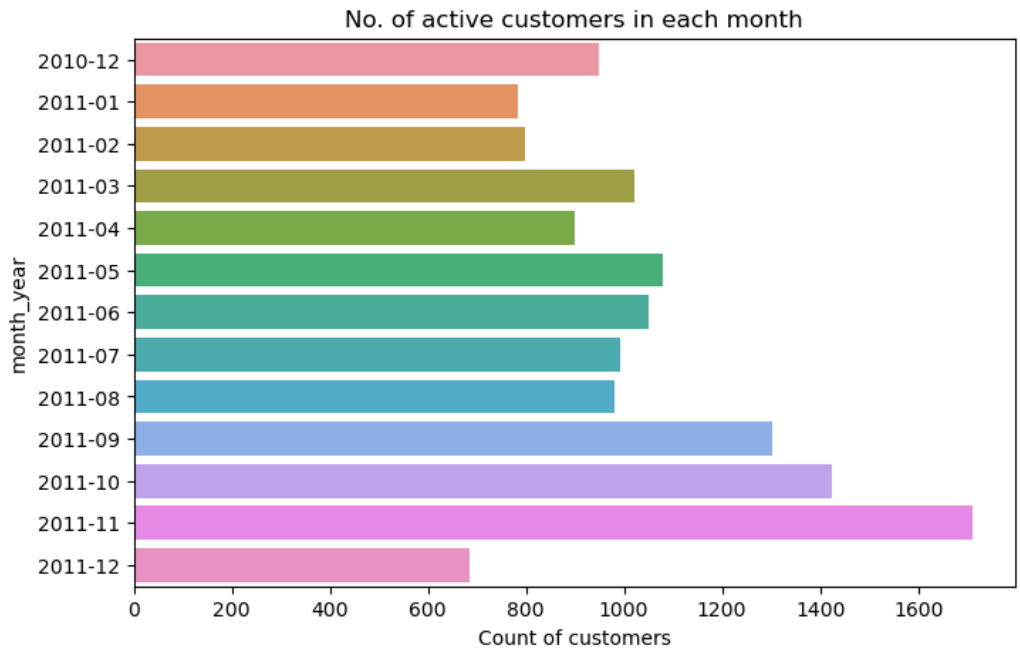
```
month_year
2010-12     948
2011-01     783
2011-02     798
2011-03    1020
2011-04     899
2011-05    1079
2011-06    1051
2011-07     993
2011-08     980
2011-09    1302
2011-10    1425
2011-11    1711
2011-12     686
Freq: M, Name: CustomerID, dtype: int64
```

In [55]:

```
plt.figure(figsize=(8,5))
sns.barplot(x=month_cohort.values,y=month_cohort.index)
plt.xlabel("Count of customers")
plt.title("No. of active customers in each month")
```

Out[55]:

```
Text(0.5, 1.0, 'No. of active customers in each month')
```



b. Analyzing the retention rate of customers.

In [57]:

```
month_cohort-month_cohort.shift(1)
```

Out[57]:

```
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 [58]:

```
retention_rate = round(month_cohort.pct_change(periods=1)*100,2)
retention_rate
```

Out[58]:

```
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 [61]:

```
plt.figure(figsize=(8,5))
sns.barplot(x=retention_rate.values,y=retention_rate.index)
plt.xlabel('Retention Rate(%)')
plt.title('Month-wise Retention Rate')
```

Out[61]:

```
Text(0.5, 1.0, 'Month-wise Retention Rate')
```

3. Building a RFM model – Recency Frequency and Monetary based on their behaviour.

Recency is about when was the last order of a customer. It means the number of days since a customer made the last purchase. If it's a case for a website or an app, this could be interpreted as the last visit day or the last login time. Frequency is about the number of purchase in a given period. It could be 3 months, 6 months or 1 year. So we can understand this value as for how often or how many times a customer used the product of a company. The bigger the value is, the more engaged the customers are. Could we say them as our VIP? Not necessary. Cause we also have to think about how much they actually paid for each purchase, which means monetary value. Monetary is the total amount of money a customer spent in that given period. Therefore big spenders will be differentiated with other customers such as MVP or VIP.

Note: Rate "Recency" for customer who have been active more recently better than the less recent customer, because each company wants its customers to be recent Rate "Frequency" and "Monetary Value" higher label because we want Customer to spend more money and visit more often.

Monetory Analysis

In [63]:

```
df['Amount']=df['Quantity']*df['UnitPrice']
df.head()
```

Out[63]:

	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 [67]:

```
df_monetary = df.groupby('CustomerID').sum()['Amount'].reset_index()
df_monetary
```

Out[67]:

	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	2094.88
4371	18287.0	1837.28

4372 rows × 2 columns

Frequency Analysis

In [69]:

```
df_frequency = df.groupby('CustomerID').nunique()['InvoiceNo'].reset_index()
df_frequency
```

Out[69]:

	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

Recency Analysis

In [71]:

```
from datetime import timedelta
```

In [72]:

```
ref_day = max(df['InvoiceDate']) + timedelta(days=1)
df['days_to_last_order'] = (ref_day - df['InvoiceDate']).dt.days
df.head()
```

Out[72]:

	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 [78]:

```
df_recency=df.groupby('CustomerID').min()['days_to_last_order'].reset_index()
df_recency
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_2164\2039511598.py:1: FutureWarning: Dropping invalid columns in DataFra
meGroupBy.min is deprecated. In a future version, a TypeError will be raised. Before calling .min, select only colu
mns which should be valid for the function.
  df_recency=df.groupby('CustomerID').min()['days_to_last_order'].reset_index()
```

Out[78]:

	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 Metrics

In [80]:

```
df_rf = pd.merge(df_recency,df_frequency,on='CustomerID',how='inner')
df_rf
```

Out[80]:

	CustomerID	days_to_last_order	InvoiceNo
0	12346.0	326	2
1	12347.0	2	7
2	12348.0	75	4
3	12349.0	19	1
4	12350.0	310	1
...	...	...	...
4367	18280.0	278	1
4368	18281.0	181	1
4369	18282.0	8	3
4370	18283.0	4	16
4371	18287.0	43	3

4372 rows × 3 columns

In [121]:

```
df_rfm = pd.merge(df_rf,df_monetary,on='CustomerID',how='inner')
df_rfm
```

Out[121]:

	CustomerID	days_to_last_order	InvoiceNo	Amount
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
...	...	...	...	...
4367	18280.0	278	1	180.60
4368	18281.0	181	1	80.82
4369	18282.0	8	3	176.60
4370	18283.0	4	16	2094.88
4371	18287.0	43	3	1837.28

4372 rows × 4 columns

In [122]:

```
df_rfm.columns=['CustomerID','Recency','Frequency','Monetary']
df_rfm.head()
```

Out[122]:

	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

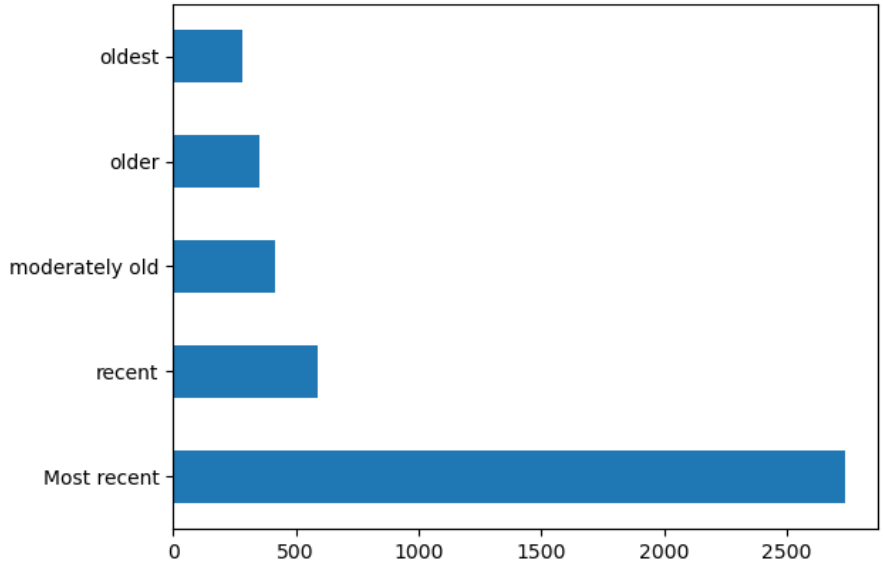
Building RFM Segments

In [123]:

```
df_rfm['Recency_labels'] = pd.cut(df_rfm['Recency'],bins=5, labels = ['Most recent','recent','moderately old','older','oldest'])
df_rfm['Recency_labels'].value_counts().plot(kind='barh')
df_rfm['Recency_labels'].value_counts()
```

Out[123]:

```
Most recent       2734
recent             588
moderately old     416
older              353
oldest             281
Name: Recency_labels, dtype: int64
```

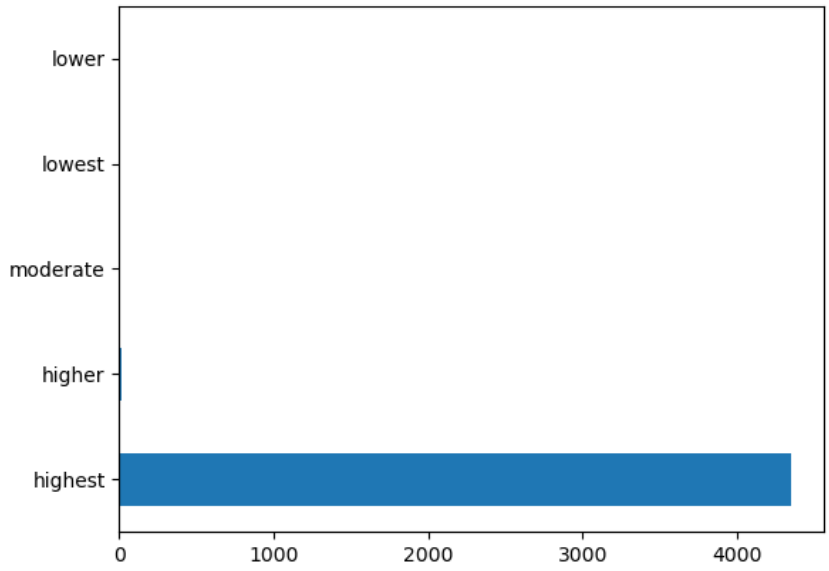


In [124]:

```
df_rfm['Frequency_labels'] = pd.cut(df_rfm['Frequency'],bins=5, labels = ['highest','higher','moderate','lower','lowest'])
df_rfm['Frequency_labels'].value_counts().plot(kind='barh')
df_rfm['Frequency_labels'].value_counts()
```

Out[124]:

```
highest     4348
higher        18
moderate       3
lowest         2
lower          1
Name: Frequency_labels, dtype: int64
```

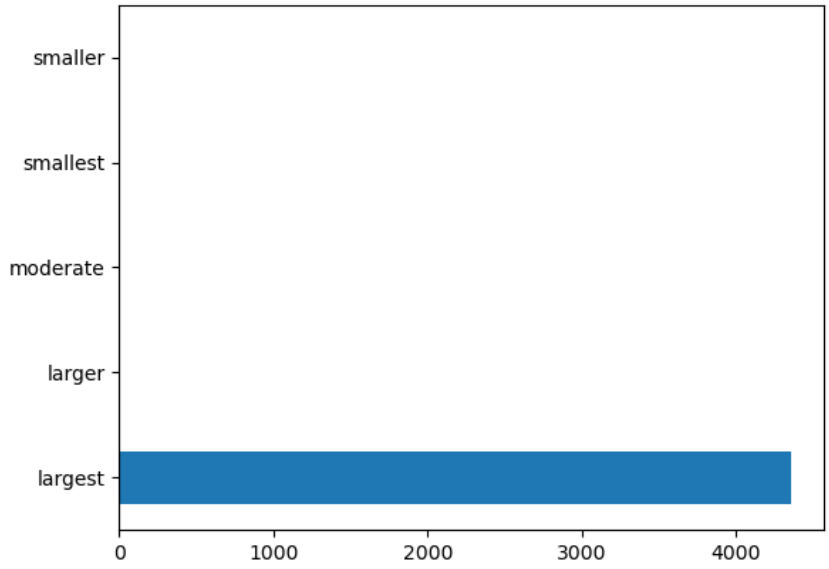


In [125]:

```
df_rfm['Monetary_labels'] = pd.cut(df_rfm['Monetary'],bins=5, labels = ['largest','larger','moderate','smaller','smallest'])
df_rfm['Monetary_labels'].value_counts().plot(kind='barh')
df_rfm['Monetary_labels'].value_counts()
```

Out[125]:

```
largest     4358
larger         8
moderate       3
smallest       2
smaller        1
Name: Monetary_labels, dtype: int64
```



In [126]:

```
df_rfm['Monetary_labels']
```

Out[126]:

```
0       largest
1       largest
2       largest
3       largest
4       largest
         ...
4367    largest
4368    largest
4369    largest
4370    largest
4371    largest
Name: Monetary_labels, Length: 4372, dtype: category
Categories (5, object): ['largest' < 'larger' < 'moderate' < 'smaller' < 'smallest']
```

In [127]:

```
df_rfm.head()
```

Out[127]:

	CustomerID	Recency	Frequency	Monetary	Recency_labels	Frequency_labels	Monetary_labels
0	12346.0	326	2	0.00	oldest	highest	largest
1	12347.0	2	7	4310.00	Most recent	highest	largest
2	12348.0	75	4	1797.24	Most recent	highest	largest
3	12349.0	19	1	1757.55	Most recent	highest	largest
4	12350.0	310	1	334.40	oldest	highest	largest

In [150]:

```
df_rfm['Recency_labels']=df_rfm['Recency_labels'].astype(str)
df_rfm['Frequency_labels']=df_rfm['Frequency_labels'].astype(str)
df_rfm['Monetary_labels']=df_rfm['Monetary_labels'].astype(str)
```

In [154]:

```
df_rfm.dtypes
```

Out[154]:

```
CustomerID           object
Recency               int64
Frequency             int64
Monetary            float64
Recency_labels       object
Frequency_labels     object
Monetary_labels      object
dtype: object
```

In [153]:

```
df_rfm['CustomerID']=df_rfm['CustomerID'].astype(str)
```

In [158]:

```
df_rfm['rfm_segment'] = df_rfm[['Recency_labels','Frequency_labels','Monetary_labels']].agg('-'.join, axis=1)
df_rfm.head()
```

Out[158]:

	CustomerID	Recency	Frequency	Monetary	Recency_labels	Frequency_labels	Monetary_labels	rfm_segment
0	12346.0	326	2	0.00	oldest	highest	largest	oldest-highest-largest
1	12347.0	2	7	4310.00	Most recent	highest	largest	Most recent-highest-largest
2	12348.0	75	4	1797.24	Most recent	highest	largest	Most recent-highest-largest
3	12349.0	19	1	1757.55	Most recent	highest	largest	Most recent-highest-largest
4	12350.0	310	1	334.40	oldest	highest	largest	oldest-highest-largest

RFM Score

In [159]:

```
Recency_dict = {'Most recent':5,'recent':4,'moderately old':3,'older':2,'oldest':1}
Frequency_dict = {'highest':5,'higher':4,'moderate':3,'lower':2,'lowest':1}
Monetary_dict = {'largest':5,'larger':4,'moderate':3,'smaller':2,'smallest':1}
```

In [160]:

```
df_rfm['rfm_score'] = df_rfm['Recency_labels'].map(Recency_dict) + df_rfm['Frequency_labels'].map(Frequency_dict)
                                                 + df_rfm['Monetary_labels'].map(Monetary_dict)
```

In [164]:

```
df_rfm.head(10)
```

Out[164]:

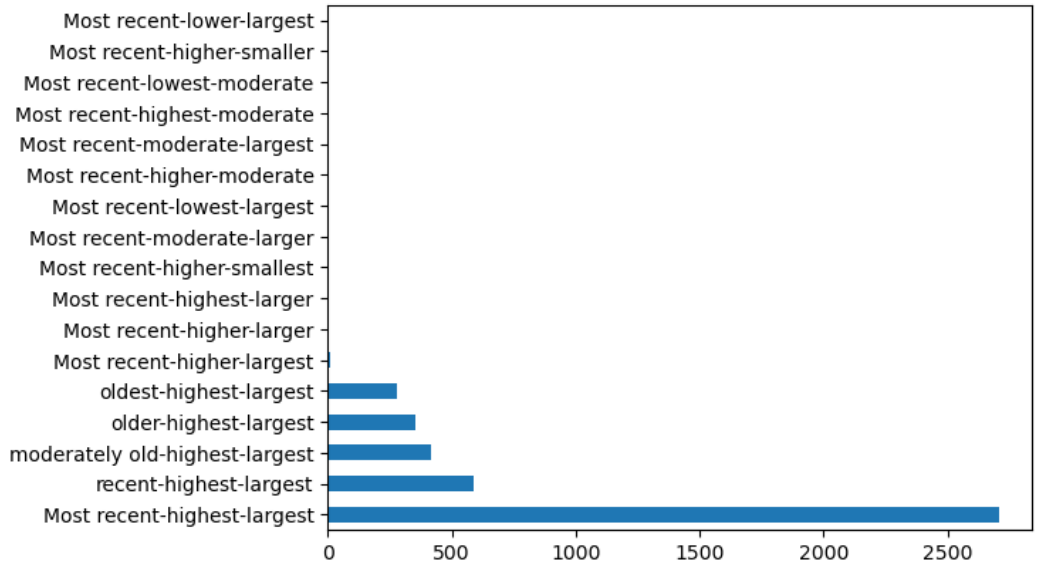
	CustomerID	Recency	Frequency	Monetary	Recency_labels	Frequency_labels	Monetary_labels	rfm_segment	rfm_score
0	12346.0	326	2	0.00	oldest	highest	largest	oldest-highest-largest	11
1	12347.0	2	7	4310.00	Most recent	highest	largest	Most recent-highest-largest	15
2	12348.0	75	4	1797.24	Most recent	highest	largest	Most recent-highest-largest	15
3	12349.0	19	1	1757.55	Most recent	highest	largest	Most recent-highest-largest	15
4	12350.0	310	1	334.40	oldest	highest	largest	oldest-highest-largest	11
5	12352.0	36	11	1545.41	Most recent	highest	largest	Most recent-highest-largest	15
6	12353.0	204	1	89.00	moderately old	highest	largest	moderately old-highest-largest	13
7	12354.0	232	1	1079.40	older	highest	largest	older-highest-largest	12
8	12355.0	214	1	459.40	moderately old	highest	largest	moderately old-highest-largest	13
9	12356.0	23	3	2811.43	Most recent	highest	largest	Most recent-highest-largest	15

In [165]:

```
df_rfm['rfm_segment'].value_counts().plot(kind='barh')
```

Out[165]:

<AxesSubplot:>

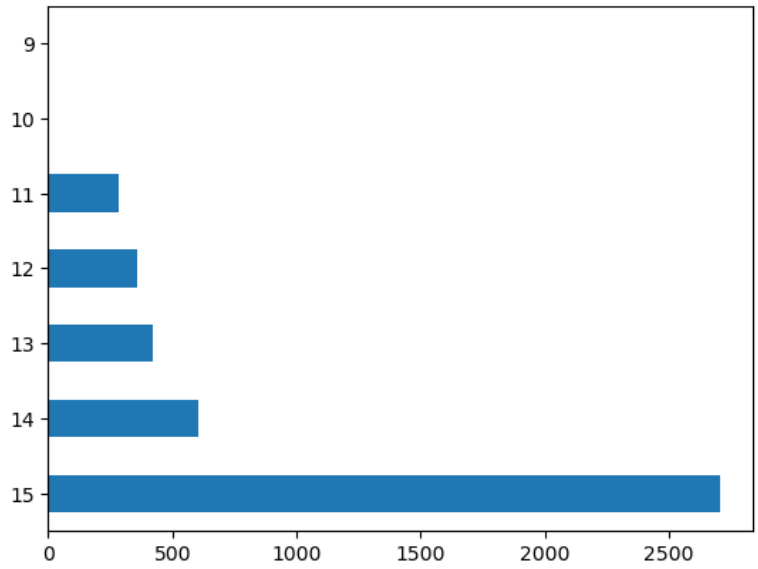


In [167]:

```
df_rfm['rfm_score'].value_counts().plot(kind='barh')
```

Out[167]:

<AxesSubplot:>



Data Modeling:

1. 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 [170]:

```
print(df_rfm.shape)
df_rfm.head(10)
```

(4372, 9)

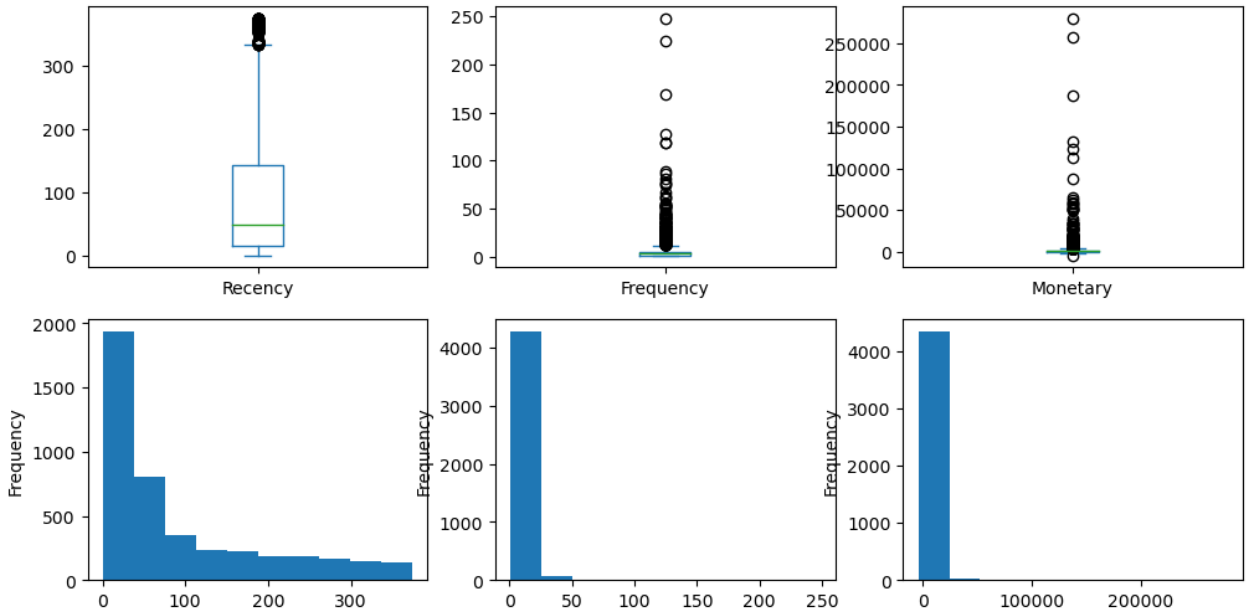
Out[170]:

	CustomerID	Recency	Frequency	Monetary	Recency_labels	Frequency_labels	Monetary_labels	rfm_segment	rfm_score
0	12346.0	326	2	0.00	oldest	highest	largest	oldest-highest-largest	11
1	12347.0	2	7	4310.00	Most recent	highest	largest	Most recent-highest-largest	15
2	12348.0	75	4	1797.24	Most recent	highest	largest	Most recent-highest-largest	15
3	12349.0	19	1	1757.55	Most recent	highest	largest	Most recent-highest-largest	15
4	12350.0	310	1	334.40	oldest	highest	largest	oldest-highest-largest	11
5	12352.0	36	11	1545.41	Most recent	highest	largest	Most recent-highest-largest	15
6	12353.0	204	1	89.00	moderately old	highest	largest	moderately old-highest-largest	13
7	12354.0	232	1	1079.40	older	highest	largest	older-highest-largest	12
8	12355.0	214	1	459.40	moderately old	highest	largest	moderately old-highest-largest	13
9	12356.0	23	3	2811.43	Most recent	highest	largest	Most recent-highest-largest	15

In [176]:

```
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')
```



In [177]:

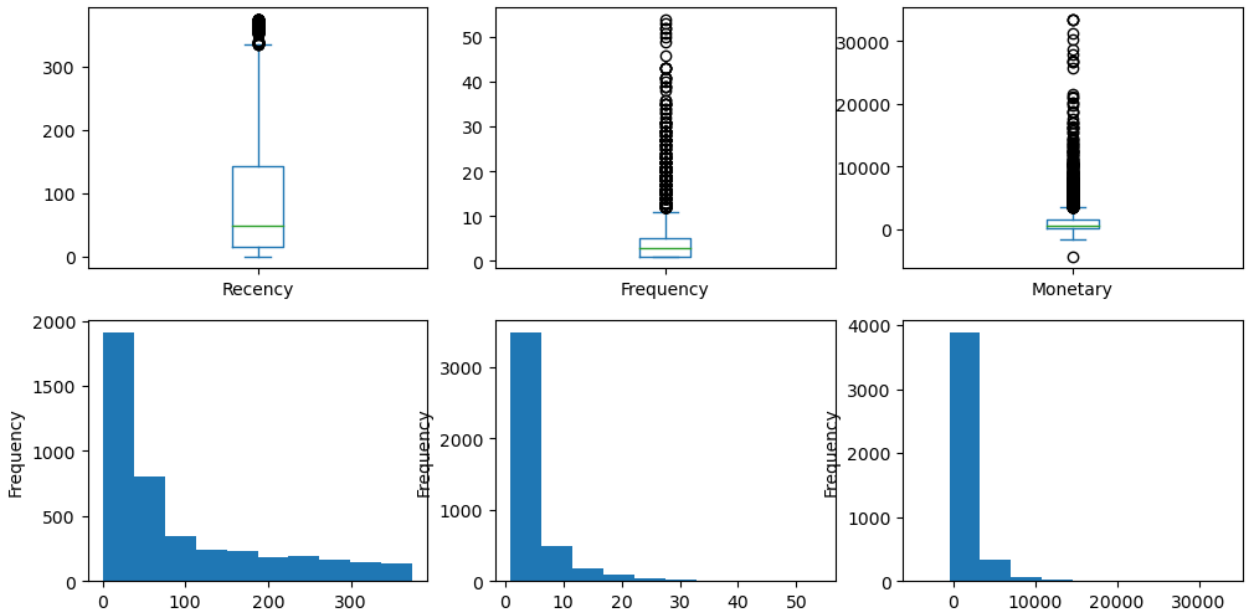
```
## Outliers: Frequency and Monetary features in above data seem to have lot of outliers. Lets drop them.
df_rfm = df_rfm[(df_rfm['Frequency']<60) & (df_rfm['Monetary']<40000)]
df_rfm.shape
```

Out[177]:

(4346, 9)

In [180]:

```
## 26 Customers removed as outlier from out data.
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')
```



Log Transformation: Now since all three features have right skewed data therefore we will use log transformation of these features in our model.

In [181]:

```
df_rfm_log_trans = pd.DataFrame()
df_rfm_log_trans['Recency'] = np.log(df_rfm['Recency'])
df_rfm_log_trans['Frequency'] = np.log(df_rfm['Frequency'])
df_rfm_log_trans['Monetary'] = np.log(df_rfm['Monetary']-df_rfm['Monetary'].min()+1)
```

Standard Scalar Transformation: It is extremely important to rescale the features so that they have a comparable scale.

In [183]:

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

In [185]:

```
scaler = StandardScaler()

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[185]:

	Recency	Frequency	Monetary
0	1.402988	-0.388507	-0.772738
1	-2.100874	0.967301	1.481096
2	0.392218	0.361655	0.361257
3	-0.552268	-1.138669	0.340058
4	1.368370	-1.138669	-0.529472

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

In [186]:

```
# k-means with some arbitrary k
kmeans = KMeans(n_clusters=3, max_iter=50)
kmeans.fit(df_rfm_scaled)
```

Out[186]:

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

In [189]:

```
kmeans.labels_
```

Out[189]:

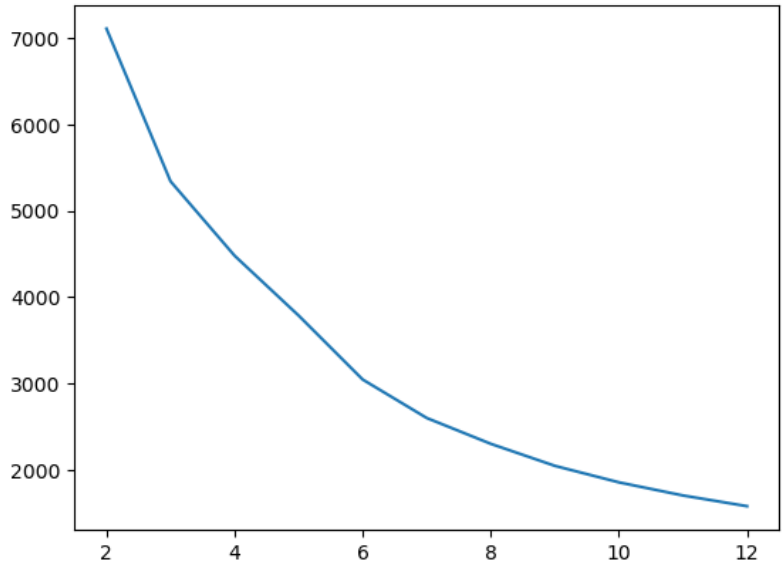
```
array([1, 2, 0, ..., 0, 2, 0])
```

In [190]:

```
# 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 [193]:

```
# 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[193]:

	clusters	intertia
0	2	7109.543832
1	3	5340.525069
2	4	4478.737998
3	5	3785.077930
4	6	3043.935539
5	7	2598.387195
6	8	2299.088161
7	9	2044.061042
8	10	1852.880260
9	11	1700.709898
10	12	1576.715203

In [195]:

```
# 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 0.4413791847967848
For n_clusters=3, the silhouette score is 0.3802382416135483
For n_clusters=4, the silhouette score is 0.36233719822438876
For n_clusters=5, the silhouette score is 0.34224817008395947
For n_clusters=6, the silhouette score is 0.34433991423703114
For n_clusters=7, the silhouette score is 0.3429138670452843
For n_clusters=8, the silhouette score is 0.33606617648024995
For n_clusters=9, the silhouette score is 0.34638748046307805
For n_clusters=10, the silhouette score is 0.3559410074082501
```

We can select optimum number of clusters as 3 in our final model

In [196]:

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

Out[196]:

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

c. Analyze these clusters and comment on the results.

In [198]:

```
# assign the label
df_rfm['Cluster_Id'] = kmeans.labels_
df_rfm.head(10)
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_2164\34536310.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 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_rfm['Cluster_Id'] = kmeans.labels_
```

Out[198]:

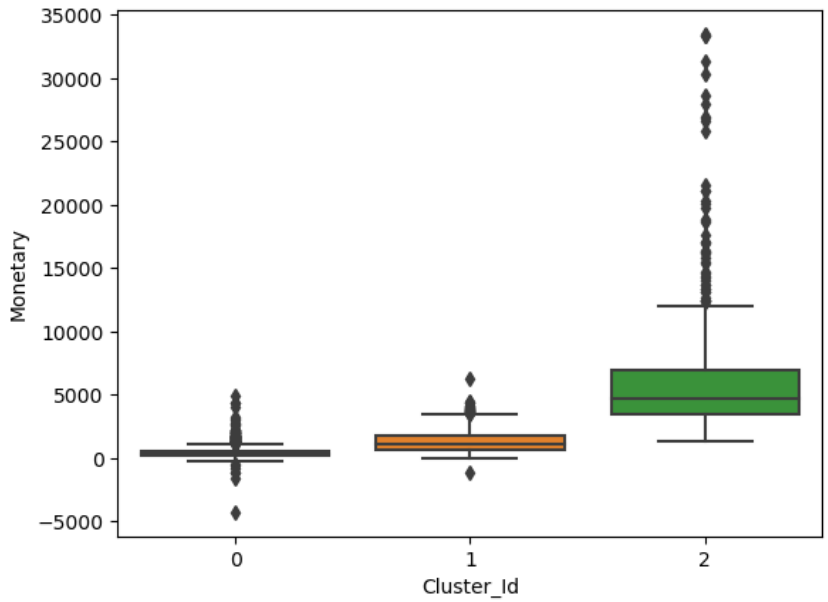
	CustomerID	Recency	Frequency	Monetary	Recency_labels	Frequency_labels	Monetary_labels	rfm_segment	rfm_score	Cluster_Id
0	12346.0	326	2	0.00	oldest	highest	largest	oldest-highest-largest	11	0
1	12347.0	2	7	4310.00	Most recent	highest	largest	Most recent-highest-largest	15	2
2	12348.0	75	4	1797.24	Most recent	highest	largest	Most recent-highest-largest	15	1
3	12349.0	19	1	1757.55	Most recent	highest	largest	Most recent-highest-largest	15	0
4	12350.0	310	1	334.40	oldest	highest	largest	oldest-highest-largest	11	0
5	12352.0	36	11	1545.41	Most recent	highest	largest	Most recent-highest-largest	15	1
6	12353.0	204	1	89.00	moderately old	highest	largest	moderately old-highest-largest	13	0
7	12354.0	232	1	1079.40	older	highest	largest	older-highest-largest	12	0
8	12355.0	214	1	459.40	moderately old	highest	largest	moderately old-highest-largest	13	0
9	12356.0	23	3	2811.43	Most recent	highest	largest	Most recent-highest-largest	15	1

In [199]:

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

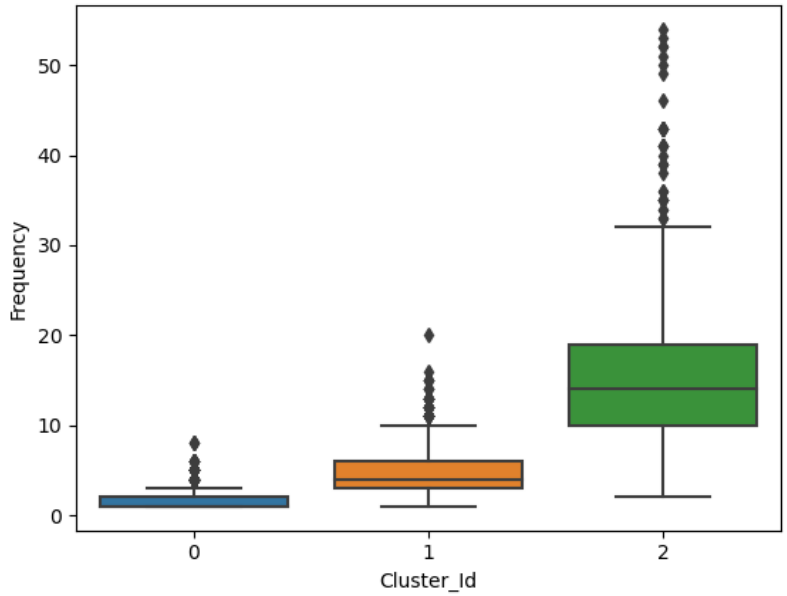
Out[199]:

```
<AxesSubplot:xlabel='Cluster_Id', ylabel='Monetary'>
```



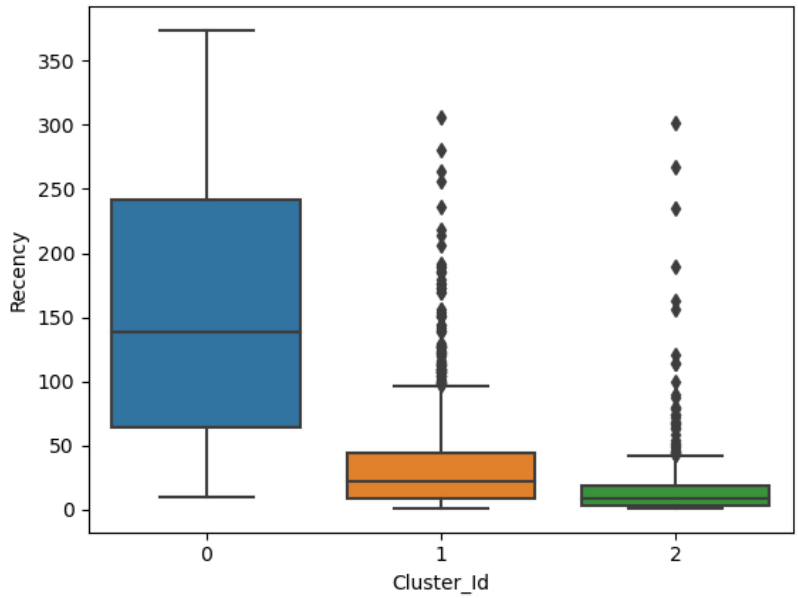
In [200]:

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



In [201]:

```
# 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.

In [202]:

```
from pandas import ExcelWriter
```

In [210]:

```
writer = pd.ExcelWriter('Project_3.xlsx')
```

In [211]:

```
# Writing dataframe to excel file for creating visualization in tableau
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 [212]:

```
product_desc = pd.read_excel("Online Retail.xlsx")
product_desc = product_desc[['StockCode', 'Description']]
product_desc = product_desc.drop_duplicates()
product_desc.to_csv('product_desc.csv', index=False)
```

In [ ]:

```

```