Overview

[Online retail is a transnational data set (https://archive.ics.uci.edu/ml/datasets/online+retail)](https://archive.ics.uci.edu/ml/datasets/online+retail) which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail. The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers.

Business Goal

We aim to segement the Customers based on RFM so that the company can target its customers efficiently.

The steps are broadly divided into:

1. Step 1: Reading and Understanding the Data
2. Step 2: Data Cleansing
3. Step 3: Data Preparation
4. Step 4: Model Building
5. Step 5: Final Analysis

This kernel is based on the assignment by IIITB collaborated with upgrad

If this Kernel helped you in any way, some UPVOTES would be very much appreciated

Step 1 : Reading and Understanding Data

In [1]:
```
# import required libraries for dataframe and visualization

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt

# import required libraries for clustering
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cut_tree
```

In [2]:
```
# Reading the data on which analysis needs to be done

retail = pd.read_csv('../input/online-retail-customer-clustering/Onlin
eRetail.csv', sep=",", encoding="ISO-8859-1", header=0)
retail.head()
```

Out[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01-12-2010 08:26	2.55	17850.0	U Kin
1	536365	71053	WHITE METAL LANTERN	6	01-12-2010 08:26	3.39	17850.0	U Kin
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01-12-2010 08:26	2.75	17850.0	U Kin
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01-12-2010 08:26	3.39	17850.0	U Kin
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01-12-2010 08:26	3.39	17850.0	U Kin

In [3]:
```
# shape of df

retail.shape
```

Out[3]: (541909, 8)

In [4]:
```
# df info

retail.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
InvoiceNo      541909 non-null object
StockCode      541909 non-null object
Description    540455 non-null object
Quantity       541909 non-null int64
InvoiceDate    541909 non-null object
UnitPrice      541909 non-null float64
CustomerID     406829 non-null float64
Country        541909 non-null object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

In [5]:
```
# df description

retail.describe()
```

Out[5]:

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

Step 2 : Data Cleansing

In [6]:
```
# Calculating the Missing Values % contribution in DF

df_null = round(100*(retail.isnull().sum())/len(retail), 2)
df_null
```

Out[6]:
```
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 [7]:
```
# Droping rows having missing values

retail = retail.dropna()
retail.shape
```

Out[7]:
```
(406829, 8)
```

In [8]:
```
# Changing the datatype of Customer Id as per Business understanding

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

Step 3 : Data Preparation

We are going to analysis the Customers based on below 3 factors:

- R (Recency): Number of days since last purchase
- F (Frequency): Number of tracsactions
- M (Monetary): Total amount of transactions (revenue contributed)

In [9]:
```
# New Attribute : Monetary

retail['Amount'] = retail['Quantity']*retail['UnitPrice']
rfm_m = retail.groupby('CustomerID')['Amount'].sum()
rfm_m = rfm_m.reset_index()
rfm_m.head()
```

Out[9]:

	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

In [10]:
```
# New Attribute : Frequency

rfm_f = retail.groupby('CustomerID')['InvoiceNo'].count()
rfm_f = rfm_f.reset_index()
rfm_f.columns = ['CustomerID', 'Frequency']
rfm_f.head()
```

Out[10]:

	CustomerID	Frequency
0	12346.0	2
1	12347.0	182
2	12348.0	31
3	12349.0	73
4	12350.0	17

In [11]:
```
# Merging the two dfs

rfm = pd.merge(rfm_m, rfm_f, on='CustomerID', how='inner')
rfm.head()
```

Out[11]:

	CustomerID	Amount	Frequency
0	12346.0	0.00	2
1	12347.0	4310.00	182
2	12348.0	1797.24	31
3	12349.0	1757.55	73
4	12350.0	334.40	17

In [12]:
```
# New Attribute : Recency

# Convert to datetime to proper datatype

retail['InvoiceDate'] = pd.to_datetime(retail['InvoiceDate'],format='%
d-%m-%Y %H:%M')
```

In [13]:
```
# Compute the maximum date to know the last transaction date

max_date = max(retail['InvoiceDate'])
max_date
```

Out[13]:
```
Timestamp('2011-12-09 12:50:00')
```

In [14]:
```
# Compute the difference between max date and transaction date

retail['Diff'] = max_date - retail['InvoiceDate']
retail.head()
```

Out[14]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	Kin
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	Kin
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	Kin
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	Kin
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	Kin

In [15]:

```
# Compute last transaction date to get the recency of customers

rfm_p = retail.groupby('CustomerID')['Diff'].min()
rfm_p = rfm_p.reset_index()
rfm_p.head()
```

Out[15]:

	CustomerID	Diff
0	12346.0	325 days 02:33:00
1	12347.0	1 days 20:58:00
2	12348.0	74 days 23:37:00
3	12349.0	18 days 02:59:00
4	12350.0	309 days 20:49:00

In [16]:

```
# Extract number of days only

rfm_p['Diff'] = rfm_p['Diff'].dt.days
rfm_p.head()
```

Out[16]:

	CustomerID	Diff
0	12346.0	325
1	12347.0	1
2	12348.0	74
3	12349.0	18
4	12350.0	309

In [17]:

```
# Merge tha dataframes to get the final RFM dataframe

rfm = pd.merge(rfm, rfm_p, on='CustomerID', how='inner')
rfm.columns = ['CustomerID', 'Amount', 'Frequency', 'Recency']
rfm.head()
```

Out[17]:

	CustomerID	Amount	Frequency	Recency
0	12346.0	0.00	2	325
1	12347.0	4310.00	182	1
2	12348.0	1797.24	31	74
3	12349.0	1757.55	73	18
4	12350.0	334.40	17	309

There are 2 types of outliers and we will treat outliers as it can skew our dataset

- Statistical
- Domain specific

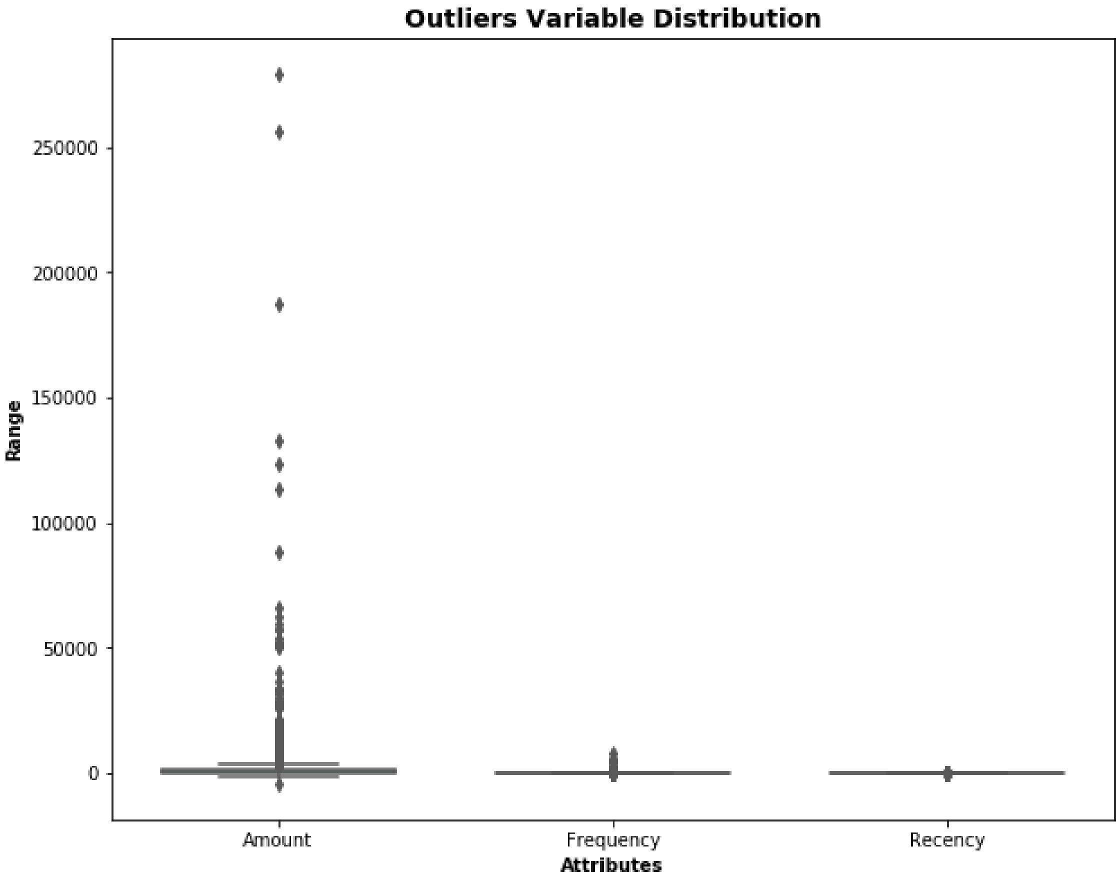
In [18]:

```
# Outlier Analysis of Amount Frequency and Recency

attributes = ['Amount','Frequency','Recency']
plt.rcParams['figure.figsize'] = [10,8]
sns.boxplot(data = rfm[attributes], orient="v", palette="Set2" ,whis=
1.5,saturation=1, width=0.7)
plt.title("Outliers Variable Distribution", fontsize = 14, fontweight
= 'bold')
plt.ylabel("Range", fontweight = 'bold')
plt.xlabel("Attributes", fontweight = 'bold')
```

Out[18]:

```
Text(0.5, 0, 'Attributes')
```



In [19]:
```
# Removing (statistical) outliers for Amount
Q1 = rfm.Amount.quantile(0.05)
Q3 = rfm.Amount.quantile(0.95)
IQR = Q3 - Q1
rfm = rfm[(rfm.Amount >= Q1 - 1.5*IQR) & (rfm.Amount <= Q3 + 1.5*IQR)]

# Removing (statistical) outliers for Recency
Q1 = rfm.Recency.quantile(0.05)
Q3 = rfm.Recency.quantile(0.95)
IQR = Q3 - Q1
rfm = rfm[(rfm.Recency >= Q1 - 1.5*IQR) & (rfm.Recency <= Q3 + 1.5*IQ
R)]

# Removing (statistical) outliers for Frequency
Q1 = rfm.Frequency.quantile(0.05)
Q3 = rfm.Frequency.quantile(0.95)
IQR = Q3 - Q1
rfm = rfm[(rfm.Frequency >= Q1 - 1.5*IQR) & (rfm.Frequency <= Q3 + 1.5
*IQR)]
```

Rescaling the Attributes

It is extremely important to rescale the variables so that they have a comparable scale.| There are two common ways of rescaling:

1. Min-Max scaling
2. Standardisation (mean-0, sigma-1)

Here, we will use Standardisation Scaling.

In [20]:
```
# Rescaling the attributes

rfm_df = rfm[['Amount', 'Frequency', 'Recency']]

# Instantiate
scaler = StandardScaler()

# fit_transform
rfm_df_scaled = scaler.fit_transform(rfm_df)
rfm_df_scaled.shape
```

Out[20]:
```
(4293, 3)
```

In [21]:
```
rfm_df_scaled = pd.DataFrame(rfm_df_scaled)
rfm_df_scaled.columns = ['Amount', 'Frequency', 'Recency']
rfm_df_scaled.head()
```

Out[21]:

	Amount	Frequency	Recency
0	-0.723738	-0.752888	2.301611
1	1.731617	1.042467	-0.906466
2	0.300128	-0.463636	-0.183658
3	0.277517	-0.044720	-0.738141
4	-0.533235	-0.603275	2.143188

Step 4 : Building the Model

K-Means Clustering

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.

The algorithm works as follows:

- First we initialize k points, called means, randomly.
- We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
- We repeat the process for a given number of iterations and at the end, we have our clusters.

In [22]:
```
# k-means with some arbitrary k

kmeans = KMeans(n_clusters=4, max_iter=50)
kmeans.fit(rfm_df_scaled)
```

Out[22]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='aut
o',
       random_state=None, tol=0.0001, verbose=0)
```

In [23]:
```
kmeans.labels_
```

Out[23]:
```
array([0, 1, 3, ..., 0, 3, 3], dtype=int32)
```

Finding the Optimal Number of Clusters

Elbow Curve to get the right number of Clusters

A fundamental step for any unsupervised algorithm is to determine the optimal number of clusters into which the data may be clustered. The Elbow Method is one of the most popular methods to determine this optimal value of k.

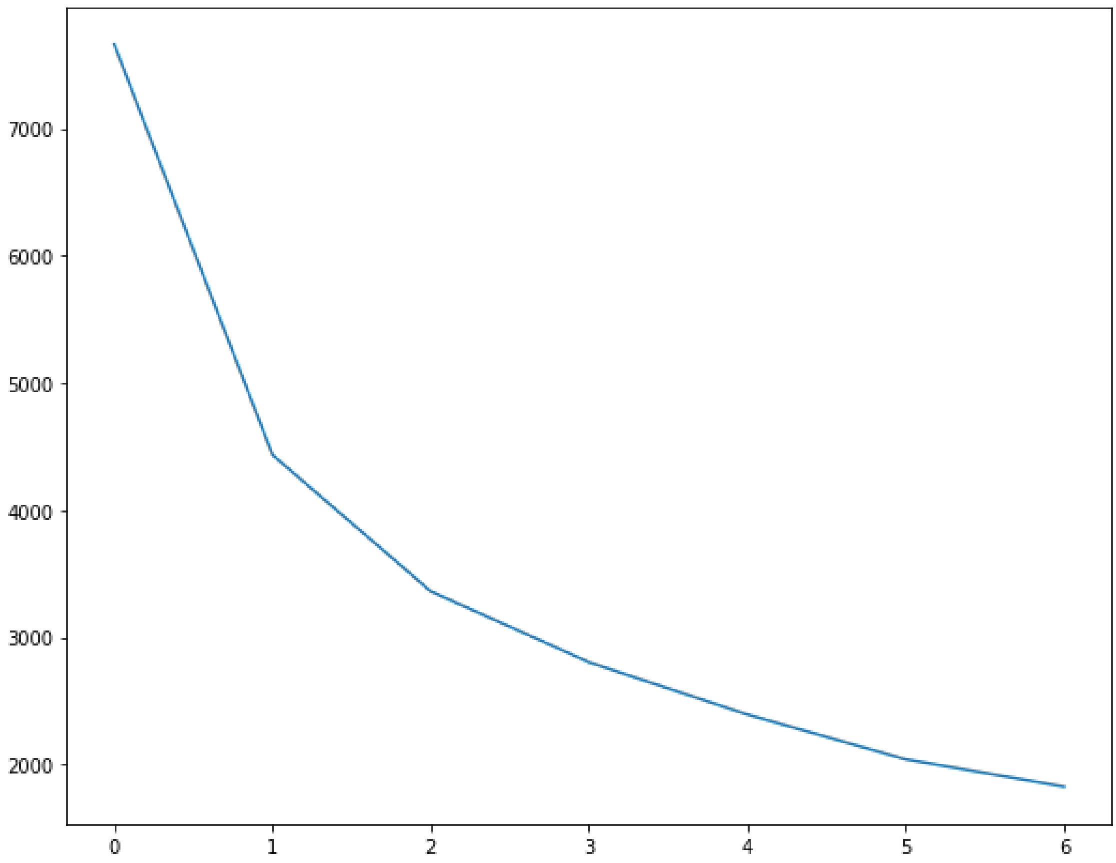
In [24]:
```
# Elbow-curve/SSD

ssd = []
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(rfm_df_scaled)

    ssd.append(kmeans.inertia_)

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

Out[24]:
```
[<matplotlib.lines.Line2D at 0x7bc1ee207518>]
```



Silhouette Analysis

$$\text{silhouette score} = \frac{p - q}{max(p, q)}$$

p is the mean distance to the points in the nearest cluster that the data point is not a part of

q is the mean intra-cluster distance to all the points in its own cluster.

- The value of the silhouette score range lies between -1 to 1.
- A score closer to 1 indicates that the data point is very similar to other data points in the cluster,
- A score closer to -1 indicates that the data point is not similar to the data points in its cluster.

In [25]:
```
# Silhouette analysis
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]

for num_clusters in range_n_clusters:

    # intialise kmeans
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(rfm_df_scaled)

    cluster_labels = kmeans.labels_

    # silhouette score
    silhouette_avg = silhouette_score(rfm_df_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 0.5415858652525395
For n_clusters=3, the silhouette score is 0.5084896296141937
For n_clusters=4, the silhouette score is 0.4816551560193964
For n_clusters=5, the silhouette score is 0.4662700564189704
For n_clusters=6, the silhouette score is 0.41753051875511704
For n_clusters=7, the silhouette score is 0.417831912137652
For n_clusters=8, the silhouette score is 0.4077658194052448
```

In [26]:
```
# Final model with k=3
kmeans = KMeans(n_clusters=3, max_iter=50)
kmeans.fit(rfm_df_scaled)
```

Out[26]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='aut
o',
       random_state=None, tol=0.0001, verbose=0)
```

In [27]:
```
kmeans.labels_
```

Out[27]:
```
array([0, 2, 1, ..., 0, 1, 1], dtype=int32)
```

In [28]:
```
# assign the label
rfm['Cluster_Id'] = kmeans.labels_
rfm.head()
```

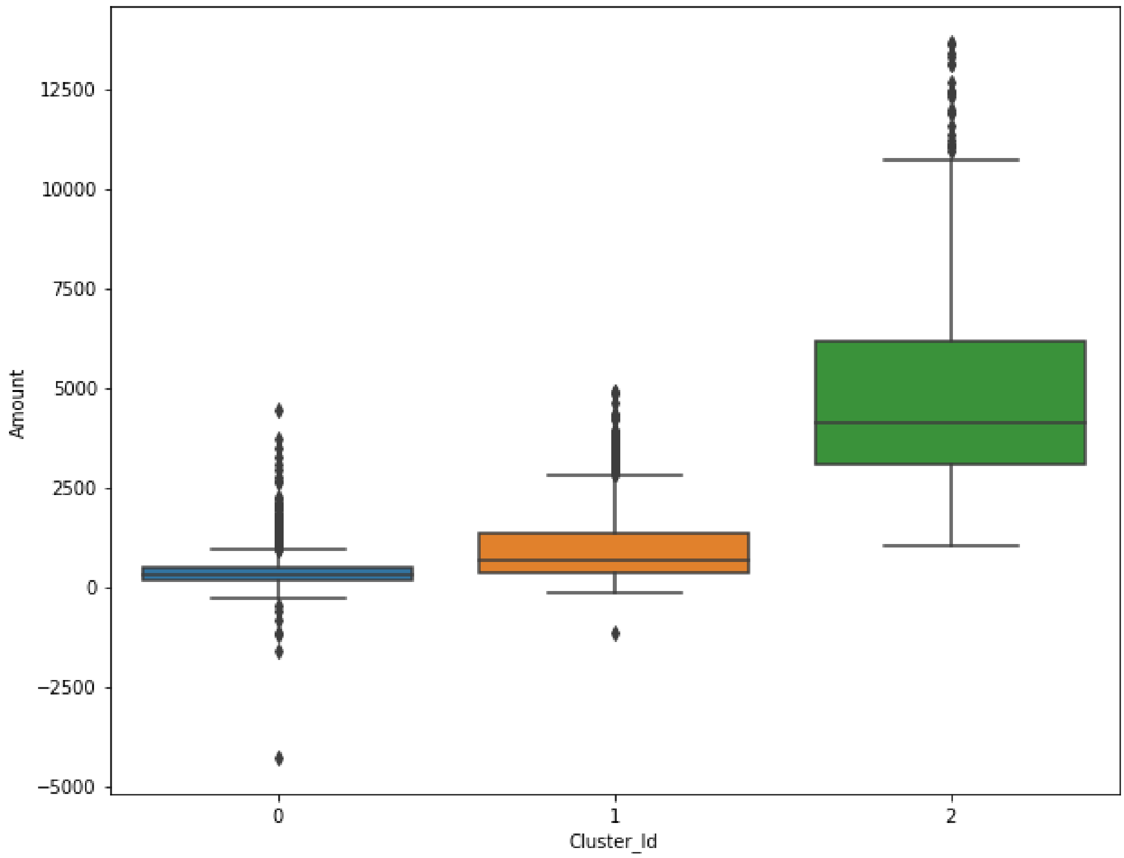
Out[28]:

	CustomerID	Amount	Frequency	Recency	Cluster_Id
0	12346.0	0.00	2	325	0
1	12347.0	4310.00	182	1	2
2	12348.0	1797.24	31	74	1
3	12349.0	1757.55	73	18	1
4	12350.0	334.40	17	309	0

In [29]:
```
# Box plot to visualize Cluster Id vs Frequency

sns.boxplot(x='Cluster_Id', y='Amount', data=rfm)
```

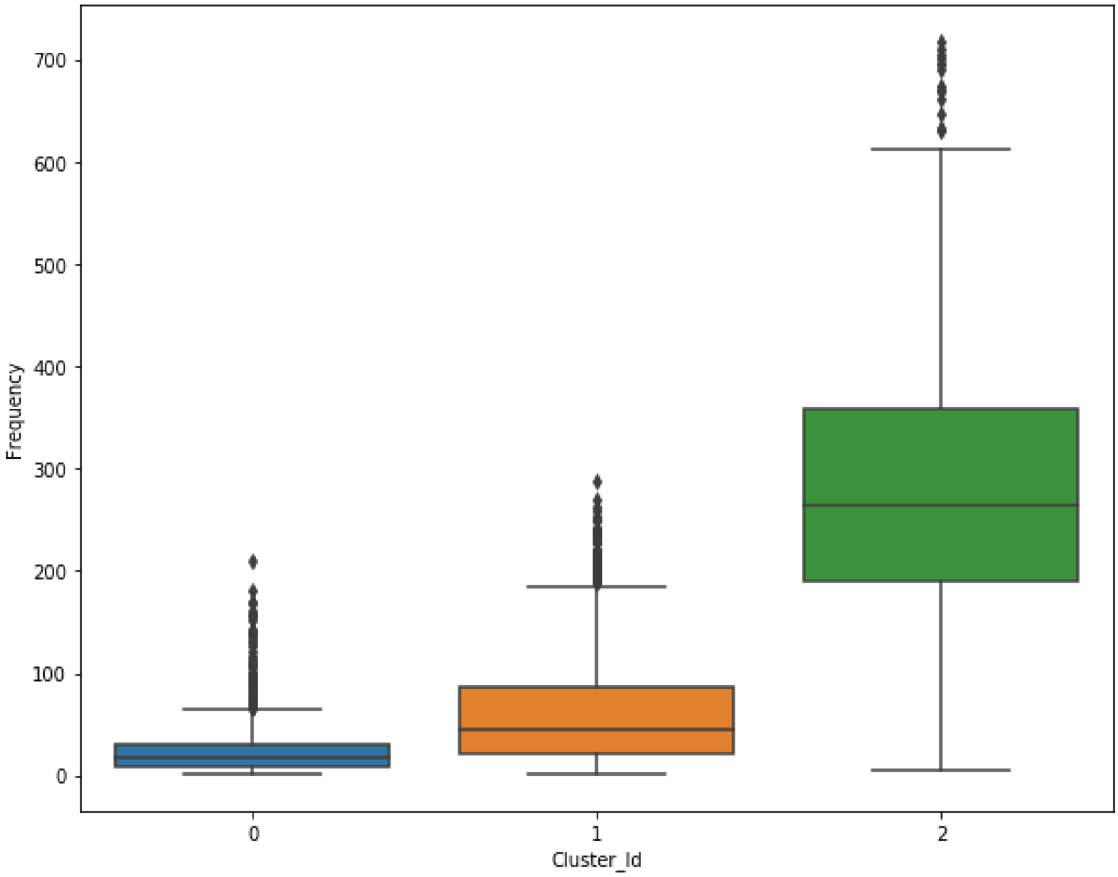
Out[29]: `<matplotlib.axes._subplots.AxesSubplot at 0x7bc1ee1cb518>`



In [30]:
```
# Box plot to visualize Cluster Id vs Frequency

sns.boxplot(x='Cluster_Id', y='Frequency', data=rfm)
```

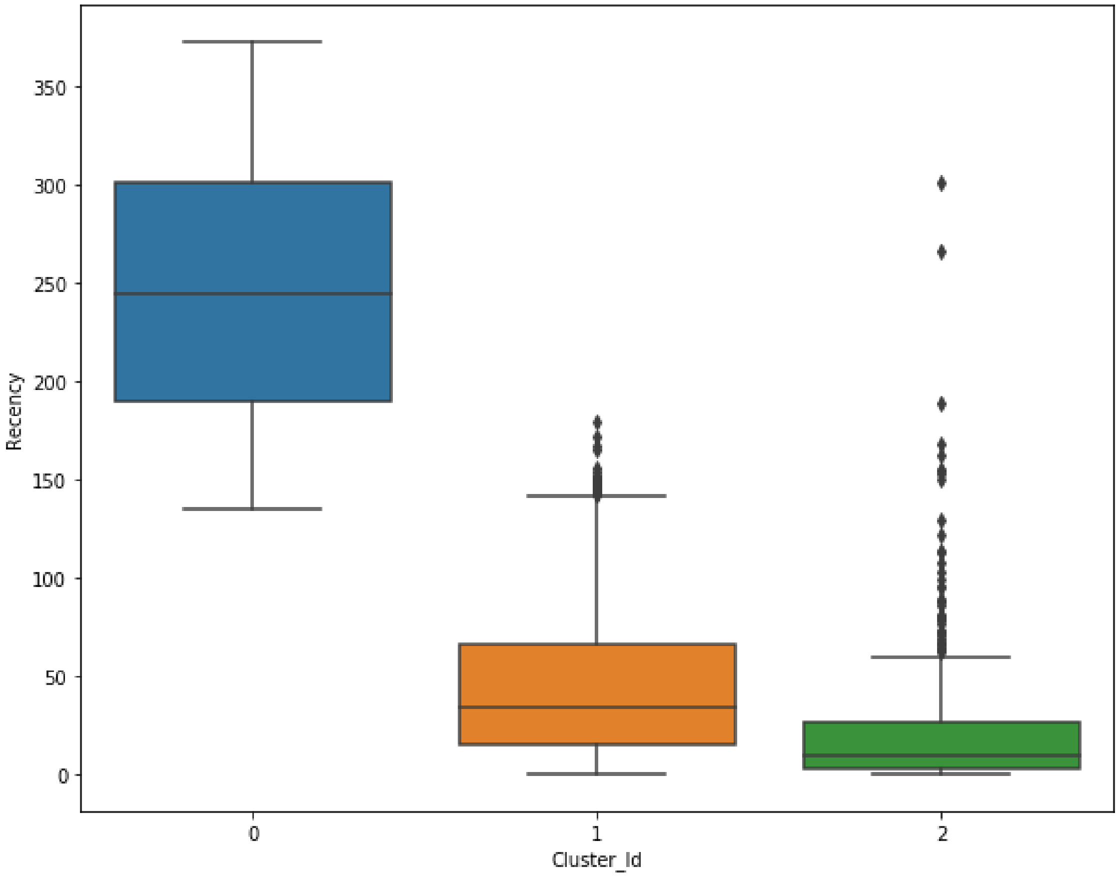
Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7bc1ee1538d0>



In [31]:
```
# Box plot to visualize Cluster Id vs Recency

sns.boxplot(x='Cluster_Id', y='Recency', data=rfm)
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7bc1ee1cb860>



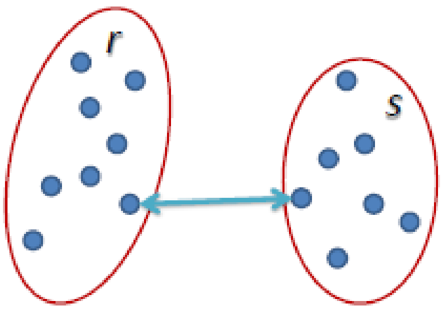
Hierarchical Clustering

Hierarchical clustering involves creating clusters that have a predetermined ordering from top to bottom. For example, all files and folders on the hard disk are organized in a hierarchy. There are two types of hierarchical clustering,

- Divisive
- Agglomerative.

Single Linkage:

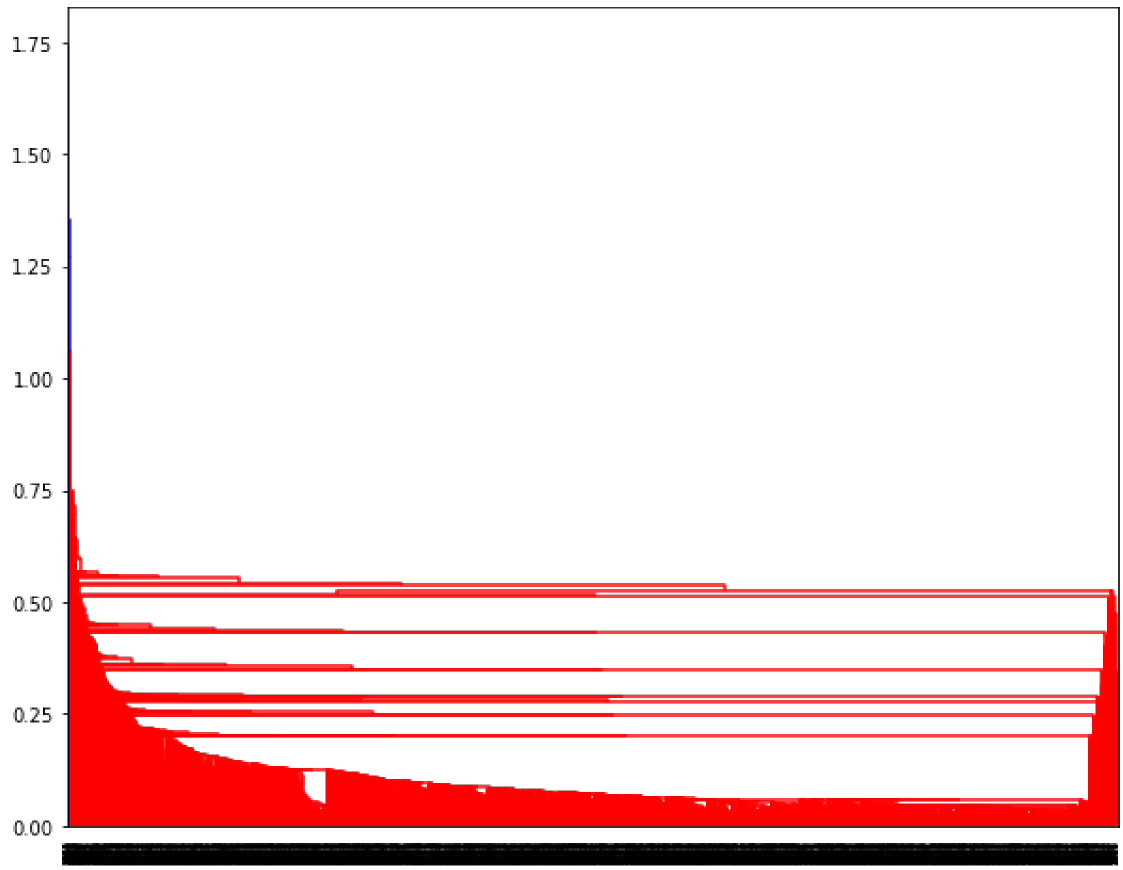
In single linkage hierarchical clustering, the distance between two clusters is defined as the shortest distance between two points in each cluster. For example, the distance between clusters “r” and “s” to the left is equal to the length of the arrow between their two closest points.



$$L(r,s) = \min(D(x_{ri}, x_{sj}))$$

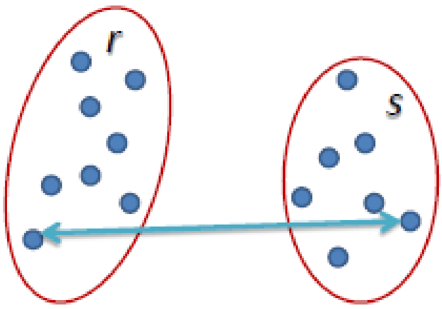
In [32]:
```
# Single linkage:

mergings = linkage(rfm_df_scaled, method="single", metric='euclidean')
dendrogram(mergings)
plt.show()
```



Complete Linkage

In complete linkage hierarchical clustering, the distance between two clusters is defined as the longest distance between two points in each cluster. For example, the distance between clusters “r” and “s” to the left is equal to the length of the arrow between their two furthest points.

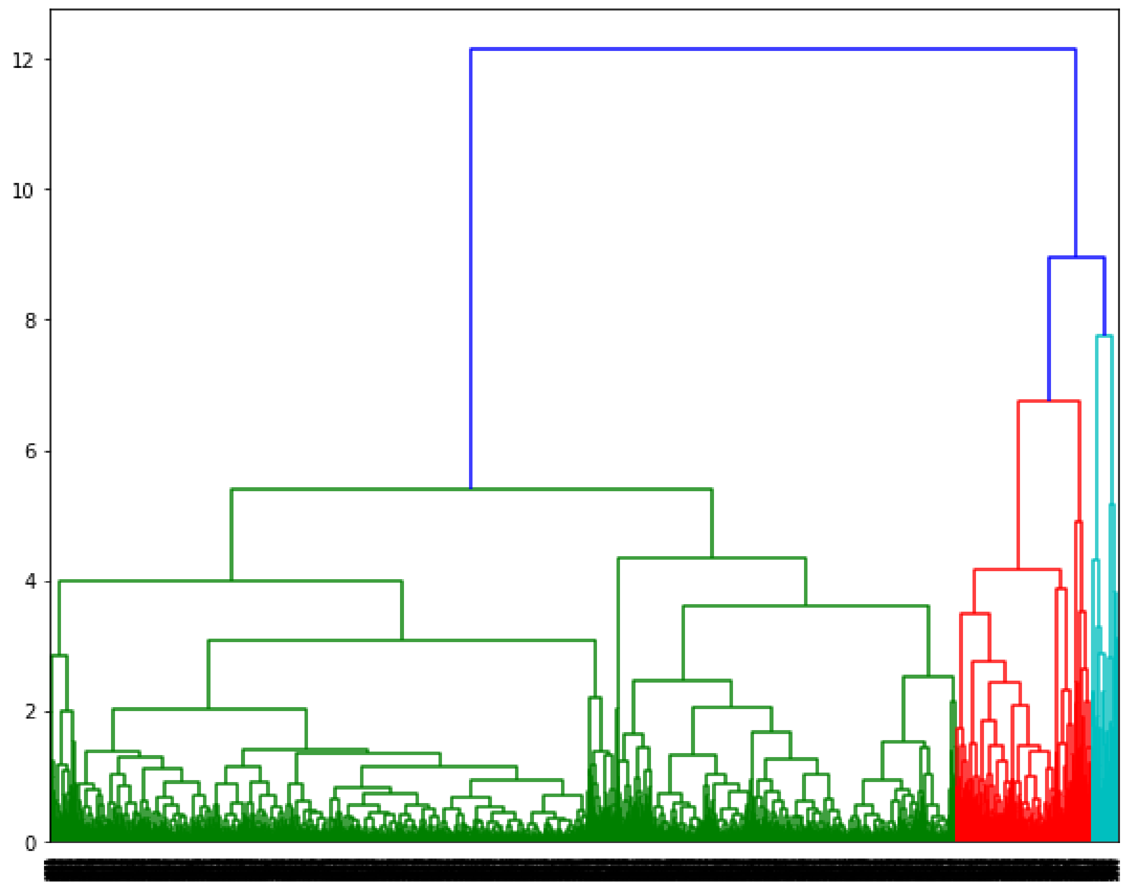


$$L(r,s) = \max(D(x_{ri}, x_{sj}))$$

In [33]:

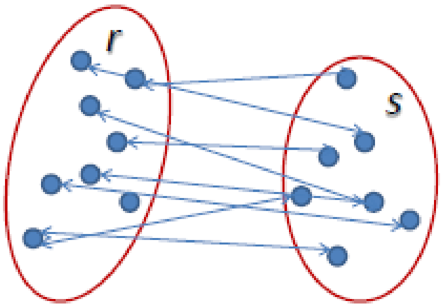
```
# Complete linkage

mergings = linkage(rfm_df_scaled, method="complete", metric='euclidea
n')
dendrogram(mergings)
plt.show()
```



Average Linkage:

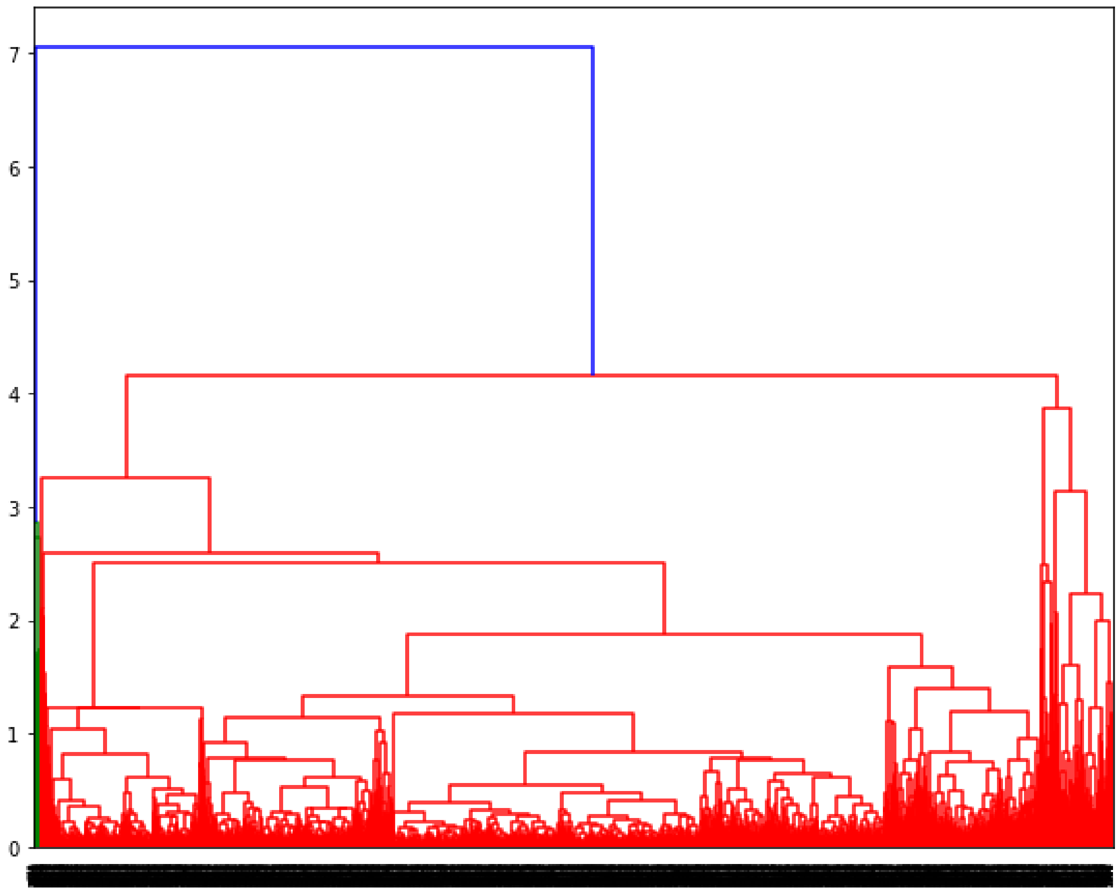
In average linkage hierarchical clustering, the distance between two clusters is defined as the average distance between each point in one cluster to every point in the other cluster. For example, the distance between clusters “r” and “s” to the left is equal to the average length each arrow between connecting the points of one cluster to the other.



$$L(r,s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} D(x_{ri}, x_{sj})$$

In [34]:
```
# Average Linkage

mergings = linkage(rfm_df_scaled, method="average", metric='euclidea
n')
dendrogram(mergings)
plt.show()
```



Cutting the Dendrogram based on K

In [35]:
```
# 3 clusters
cluster_labels = cut_tree(mergings, n_clusters=3).reshape(-1, )
cluster_labels
```

Out[35]: `array([0, 0, 0, ..., 0, 0, 0])`

In [36]:
```
# Assign cluster labels

rfm['Cluster_Labels'] = cluster_labels
rfm.head()
```

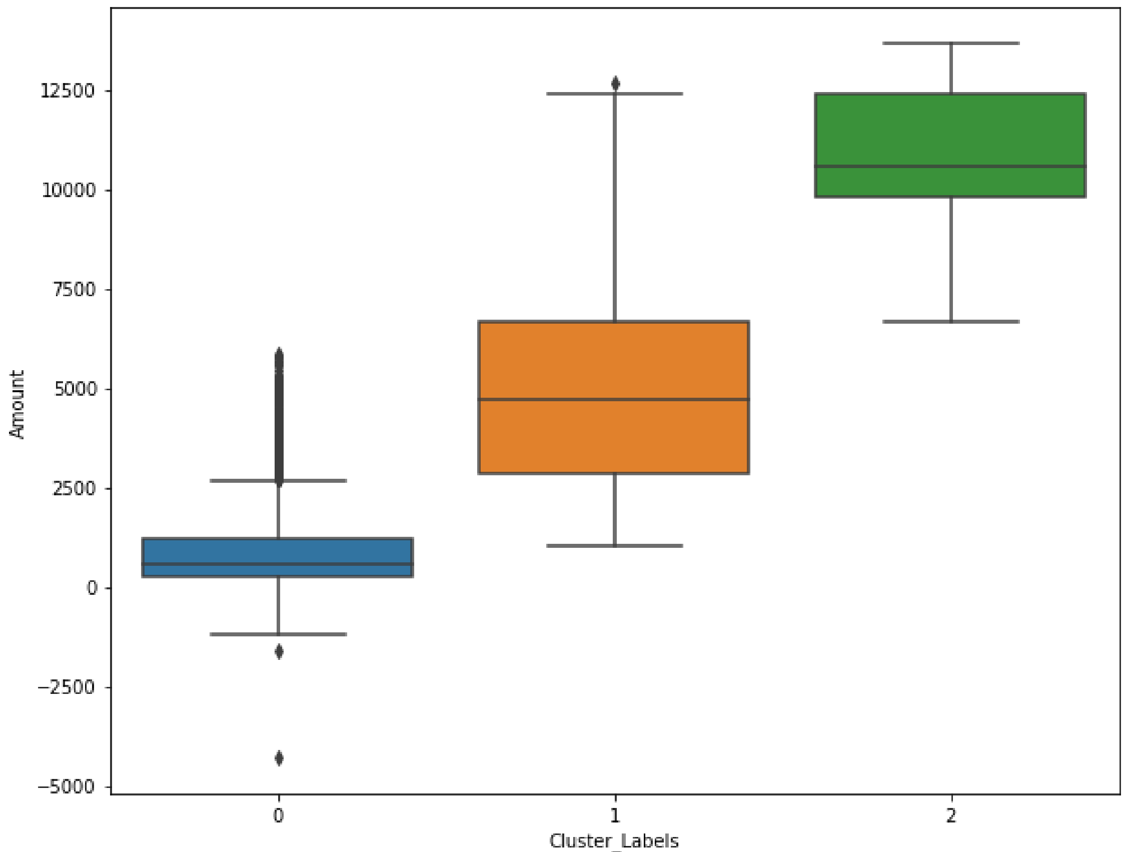
Out[36]:

	CustomerID	Amount	Frequency	Recency	Cluster_Id	Cluster_Labels
0	12346.0	0.00	2	325	0	0
1	12347.0	4310.00	182	1	2	0
2	12348.0	1797.24	31	74	1	0
3	12349.0	1757.55	73	18	1	0
4	12350.0	334.40	17	309	0	0

In [37]:
```
# Plot Cluster Id vs Amount

sns.boxplot(x='Cluster_Labels', y='Amount', data=rfm)
```

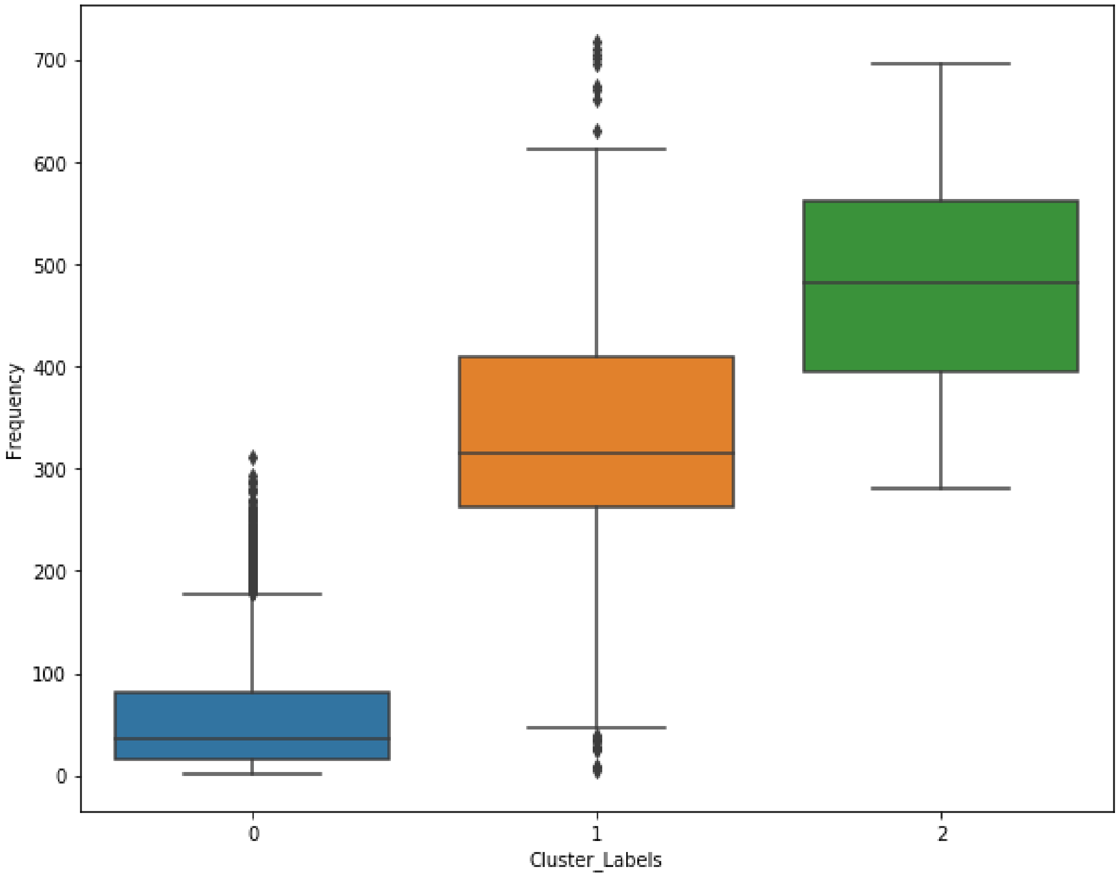
Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x7bc1d69ec668>



In [38]:
```
# Plot Cluster Id vs Frequency

sns.boxplot(x='Cluster_Labels', y='Frequency', data=rfm)
```

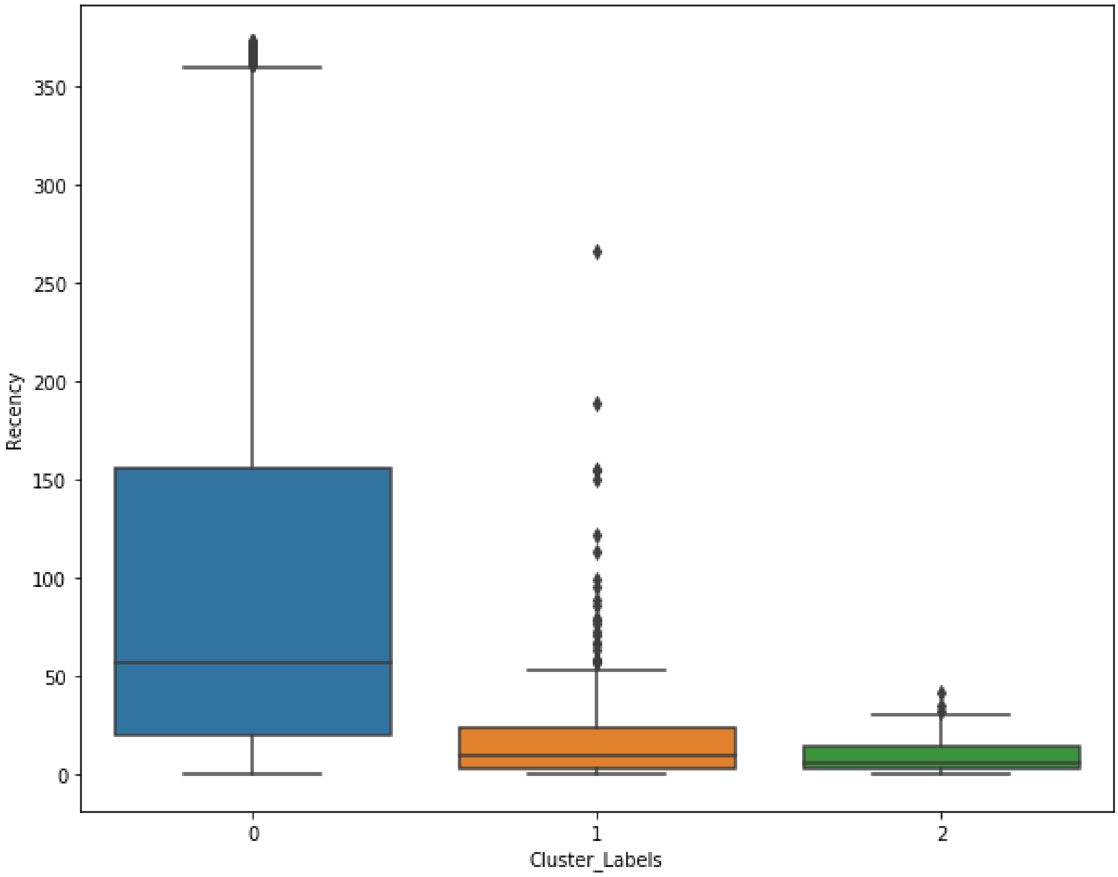
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x7bc1ec32ba90>



In [39]:
```
# Plot Cluster Id vs Recency

sns.boxplot(x='Cluster_Labels', y='Recency', data=rfm)
```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7bc1edf5c518>



Step 5 : Final Analysis

Inference:

K-Means Clustering with 3 Cluster Ids

- Customers with Cluster Id 1 are the customers with high amount of transactions as compared to other customers.
- Customers with Cluster Id 1 are frequent buyers.
- Customers with Cluster Id 2 are not recent buyers and hence least of importance from business point of view.

Hierarchical Clustering with 3 Cluster Labels

- Customers with Cluster_Labels 2 are the customers with high amount of transactions as compared to other customers.
- Customers with Cluster_Labels 2 are frequent buyers.
- Customers with Cluster_Labels 0 are not recent buyers and hence least of importance from business point of view.

If this Kernel helped you in any way, some UPVOTES would be very much appreciated