Customer Segmentation using Clustering Algorithms

Problem Statement

An online retail store is trying to understand the various customer purchase patterns for their firm and also understand Segment of the customers based on their purchasing behavior.

Aim

The objective of the project is to find useful insights about the customer purchasing history that can add advantage for the online retailer. And also need to Segment the customers based on their purchasing behavior.

Getting Started

Import Libraries

In [1]:

```
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
```

Import Data Set

In [2]:
```
# Reading the data on which analysis needs to be done
retail = pd.read_csv('OnlineRetail.csv', encoding='unicode_escape')
retail.head()
```

Out[2]:

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

EDA

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):
 #   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  object 
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object 
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

In [5]:
```
# statistical 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

Data Preprocessing

In [6]:

```
#Checking Null Values

retail.isnull().sum()
```

Out[6]:

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

In [7]:
```
#Dropping Null Values Since A huge chunk of data is missing
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)
```

Feature Consruction

We are going to analyze 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'])
```

In [13]:
```
retail['InvoiceDate']
```

Out[13]:
```
0         2010-12-01 08:26:00
1         2010-12-01 08:26:00
2         2010-12-01 08:26:00
3         2010-12-01 08:26:00
4         2010-12-01 08:26:00
                  ...
541904    2011-12-09 12:50:00
541905    2011-12-09 12:50:00
541906    2011-12-09 12:50:00
541907    2011-12-09 12:50:00
541908    2011-12-09 12:50:00
Name: InvoiceDate, Length: 406829, dtype: datetime64[ns]
```

In [14]:
```python
# Compute the maximum date to know the last transaction date

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

Out[14]:
```
Timestamp('2011-12-09 12:50:00')
```

In [15]:
```python
# Compute the difference between max date and transaction date

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

Out[15]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Amount	Diff
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	15.30	373 days 04:24:00
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	373 days 04:24:00
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	22.00	373 days 04:24:00
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	373 days 04:24:00
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	373 days 04:24:00

In [16]:

```python
# 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[16]:

	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 [17]:

```python
# Extract number of days only

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

Out[17]:

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

In [18]:
```python
# 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[18]:

	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

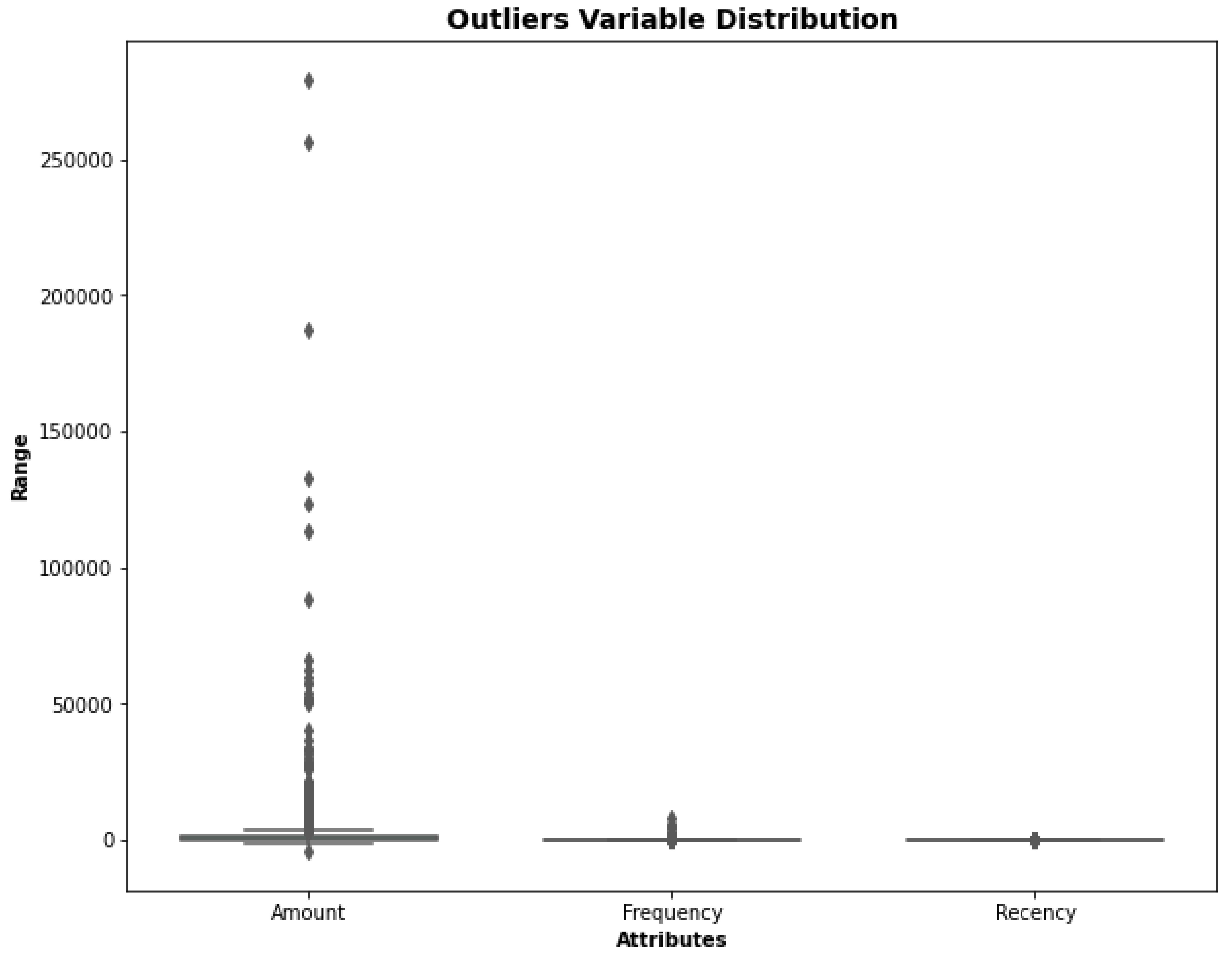
There are 2 types of outliers and we will treat outliers as it can skew our dataset

- Statistical
- Domain specific

In [19]:
```python
# 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[19]:
```
Text(0.5, 0, 'Attributes')
```



In [20]:
```
# 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*IQR)]

# 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:
- Min-Max scaling
- Standardisation (mean-0, sigma-1)
- Here, we will use Standardisation Scaling.

In [21]:

```
# Rescaling the attributes

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

# Instantiate
scaler = StandardScaler()
```

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

Out[21]: (4293, 3)

In [22]:

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

Out[22]:

	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

Choosing Model for Algorithm

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 [23]:
```
# k-means with some arbitrary k

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

Out[23]:
```
KMeans(max_iter=50, n_clusters=4)
```

In [24]:
```
kmeans.labels_
```

Out[24]:
```
array([0, 1, 2, ..., 0, 2, 2])
```

How to Find the Optimal Number of Clusters

1. Elbow Curve

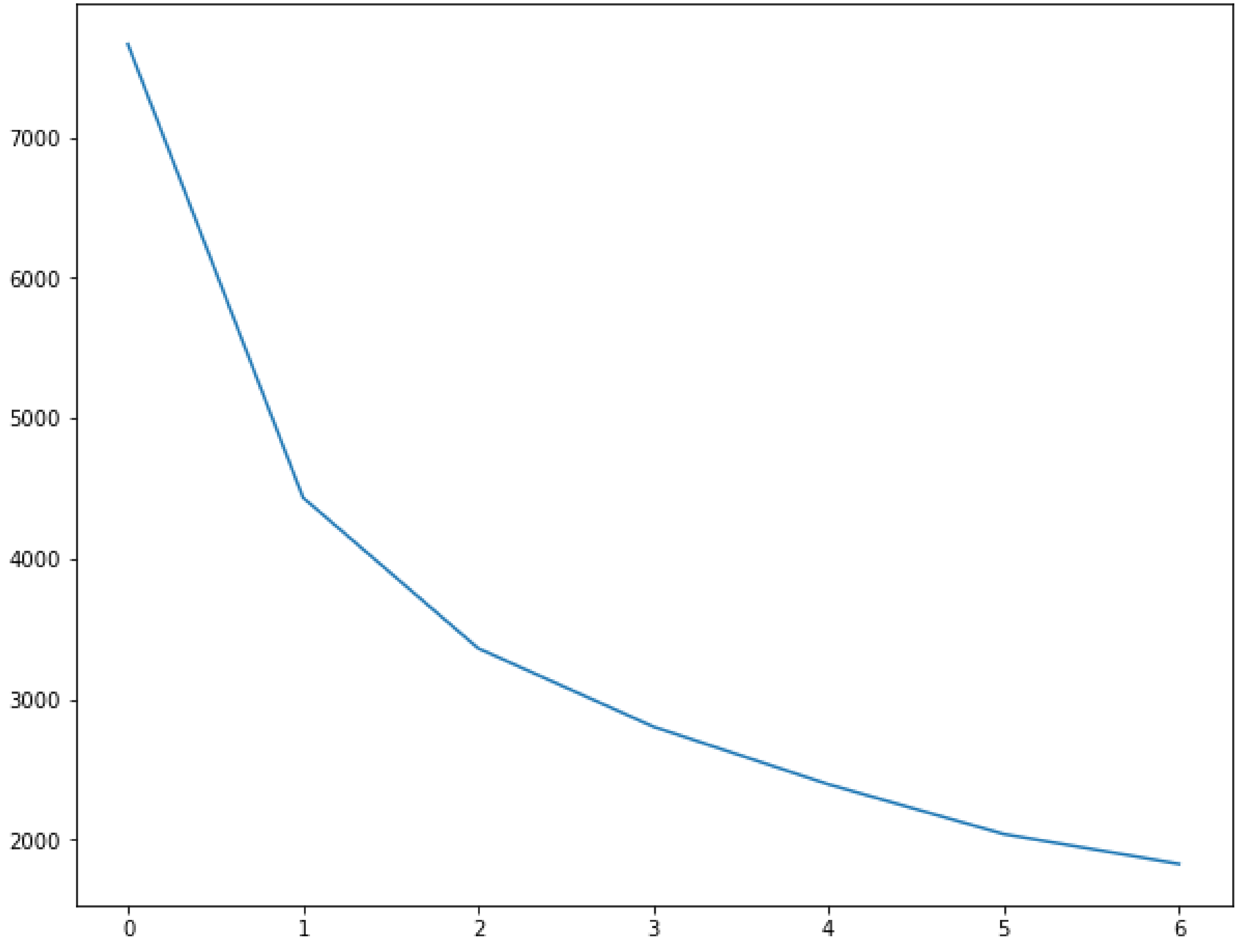
A fundamental step for any unsupervised algorithm is to determine the optimal number of clusters into which the data may be clustered. The Elbow Method is one of the most popular methods to determine this optimal value of k.

In [25]:
```
# 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);
```



2. 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 [26]:

```
# 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 0.48161393329059127
For n_clusters=5, the silhouette score is 0.46399070900769845
For n_clusters=6, the silhouette score is 0.4176766796246588
For n_clusters=7, the silhouette score is 0.41763065866927357
For n_clusters=8, the silhouette score is 0.4088271515039422
```

Model Building & Evaluation

In [27]:

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

Out[27]:

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

In [28]:

```python
kmeans.labels_
```

Out[28]:

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

In [29]:

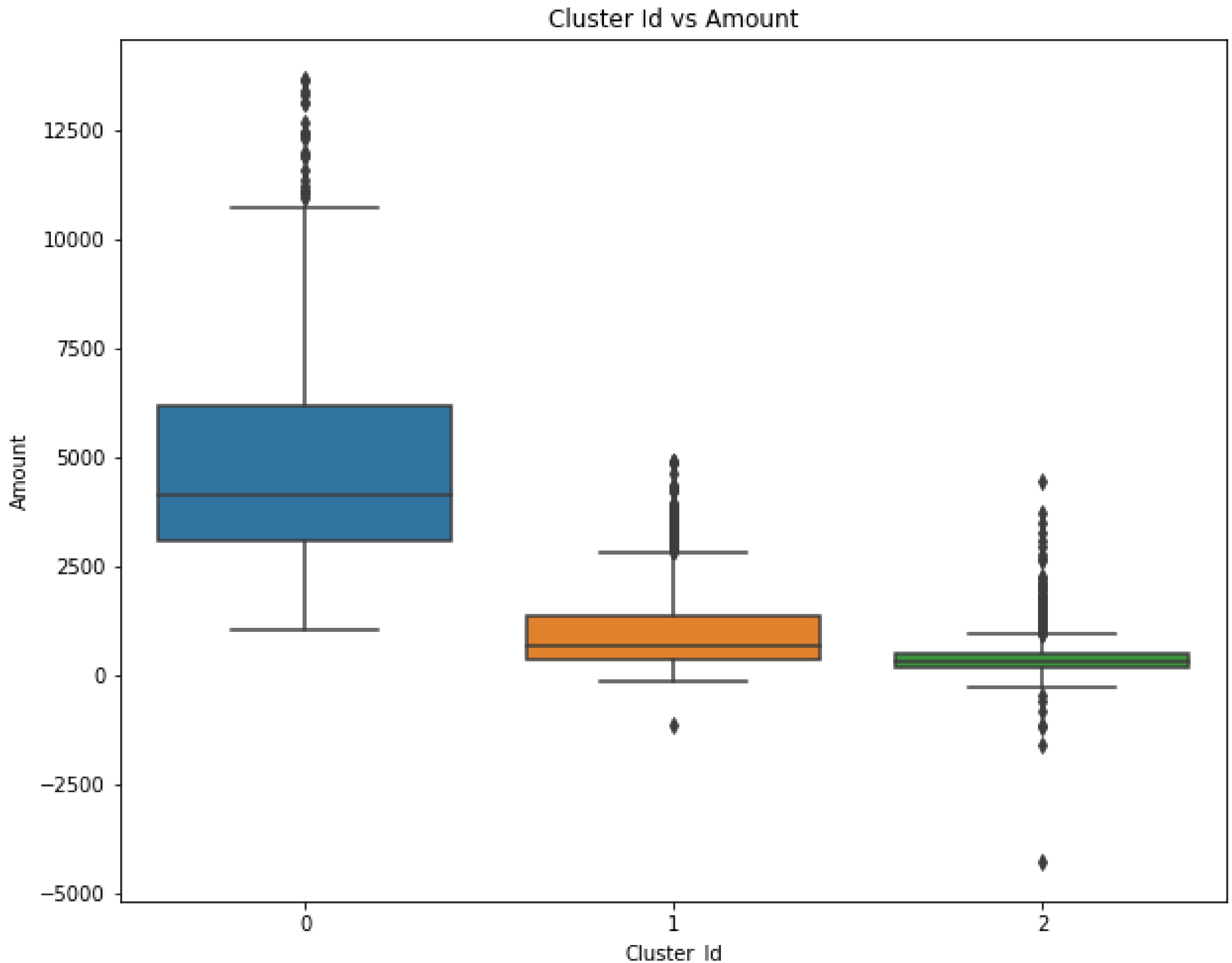
```python
# assign the label
rfm['Cluster_Id'] = kmeans.labels_
rfm.head()
```

Out[29]:

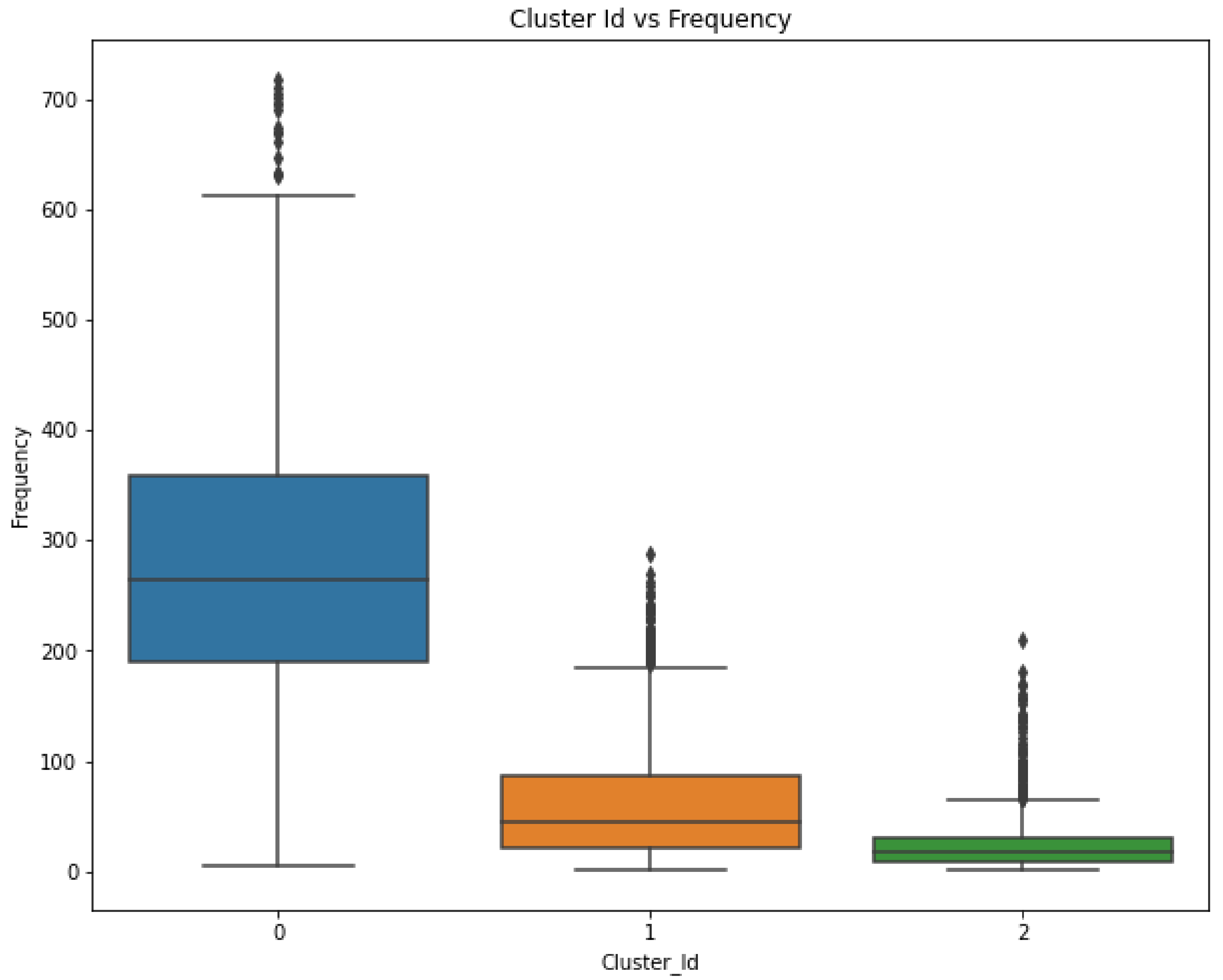
	CustomerID	Amount	Frequency	Recency	Cluster_Id
0	12346.0	0.00	2	325	2
1	12347.0	4310.00	182	1	0
2	12348.0	1797.24	31	74	1
3	12349.0	1757.55	73	18	1
4	12350.0	334.40	17	309	2

In [30]:

```
# Box plot to visualize Cluster Id vs Frequency
plt.title("Cluster Id vs Amount")
sns.boxplot(x='Cluster_Id', y='Amount', data=rfm);
```

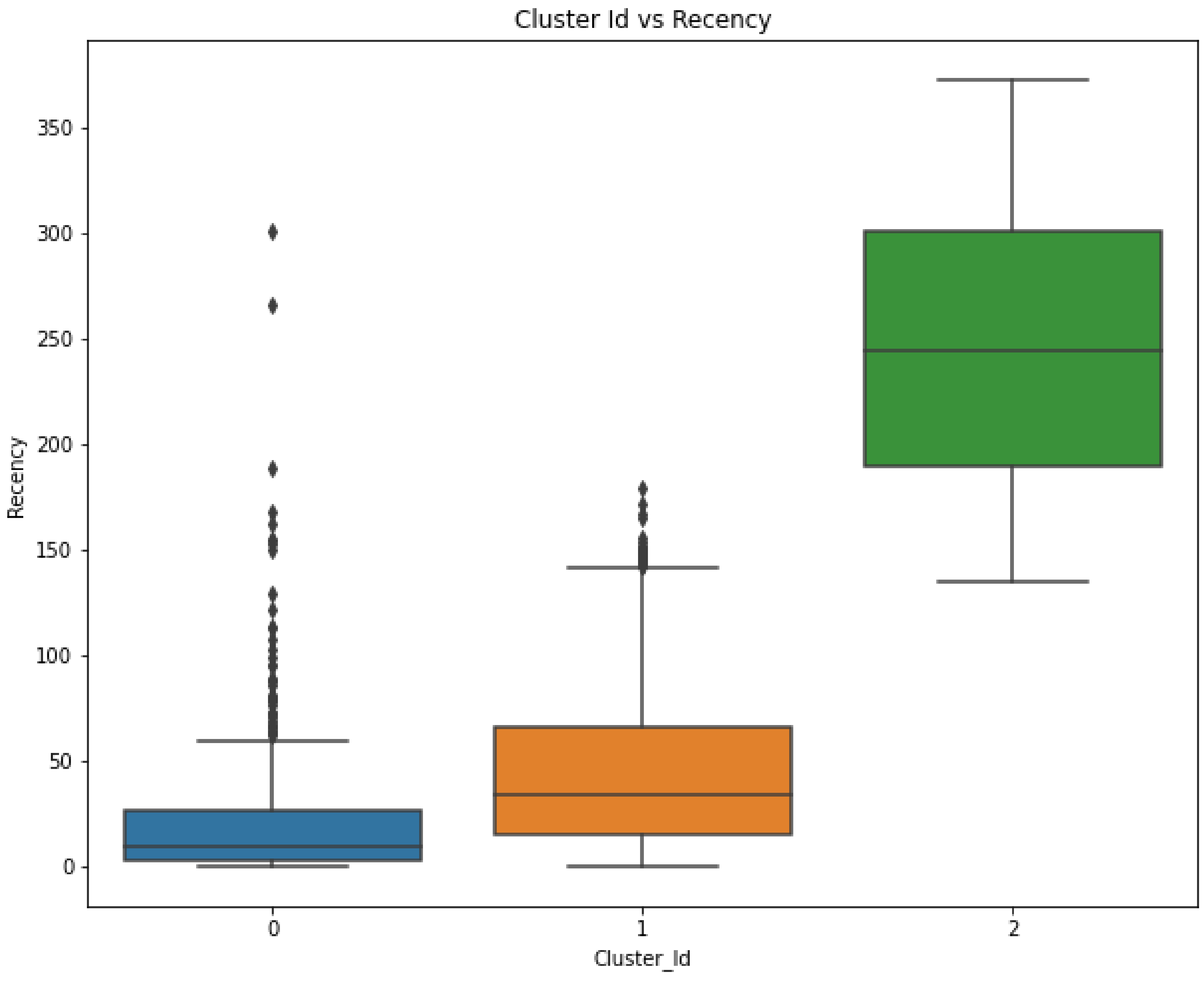


In [31]:
```
# Box plot to visualize Cluster Id vs Frequency
plt.title("Cluster Id vs Frequency")
sns.boxplot(x='Cluster_Id', y='Frequency', data=rfm);
```



In [32]:
```
# Box plot to visualize Cluster Id vs Recency
```

```python
plt.title("Cluster Id vs Recency")
sns.boxplot(x='Cluster_Id', y='Recency', data=rfm);
```



Inferences from the Project

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.