# Final Project - Predicting the Customers to be Targeted in Online Retail Using a Hierarchical Clustering Model

## Step 1. Gather data, determine the method of data collection and provenance of the data

The data I will be using comes from an online dataset published on Kaggle. The dataset is a tabular form of data consist of information of online transactions with 25900 entries and 8 columns.

## Step 2. Identify an Unsupervised Learning Problem

The goal for this final project is to understand which group of users are more likely to buy wh.

To achieve this goal, a hierarchical clustering model is built to understand the data and classify the customers. K-means clustering method is also used to build a separate model and compare results.

## Step 3. Exploratory Data Analysis (EDA) - Inspect, Visualize, and Clean the Data

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 AgglomerativeClustering, 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
from sklearn.metrics import accuracy_score, confusion_matrix
import time
```

In [2]:
```
retail = pd.read_csv('OnlineRetail.csv', sep=",", encoding="ISO-8859-1", hea
retail.head()
```

Out[2]:

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

In [3]:

```
#inspect the data and get overall information
retail.describe()
```

Out[3]:

| | 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 |

In [4]:

```
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]:
```
# check if there is null values and drop them. Clean the dataset.
retail.isnull().sum()
```

Out[5]:
```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

In [6]:
```
# Droping rows having missing values

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

Out[6]:
```
(406829, 8)
```

In [7]:
```
# Changing the datatype of Customer Id as per Business understanding

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

In [8]:
```
#exploring features - customers
customer_ids = retail["CustomerID"].nunique()
customer_ids
```

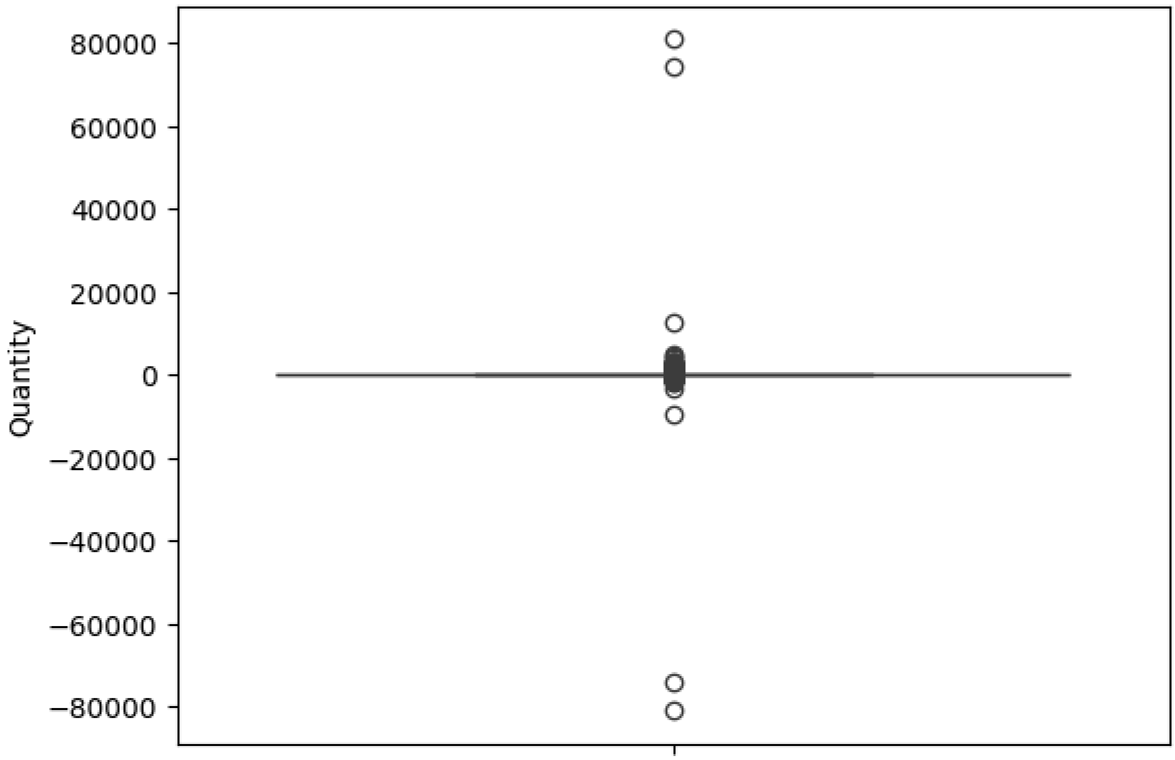
Out[8]:
```
4372
```

In [9]:
```
#exploring features - stock codes
stockcodes = retail["StockCode"].nunique()
stockcodes
```

Out[9]:
```
3684
```

In [10]:
```
#exploring features - quality
sns.boxplot(retail["Quantity"])
```

Out[10]: <Axes: ylabel='Quantity'>



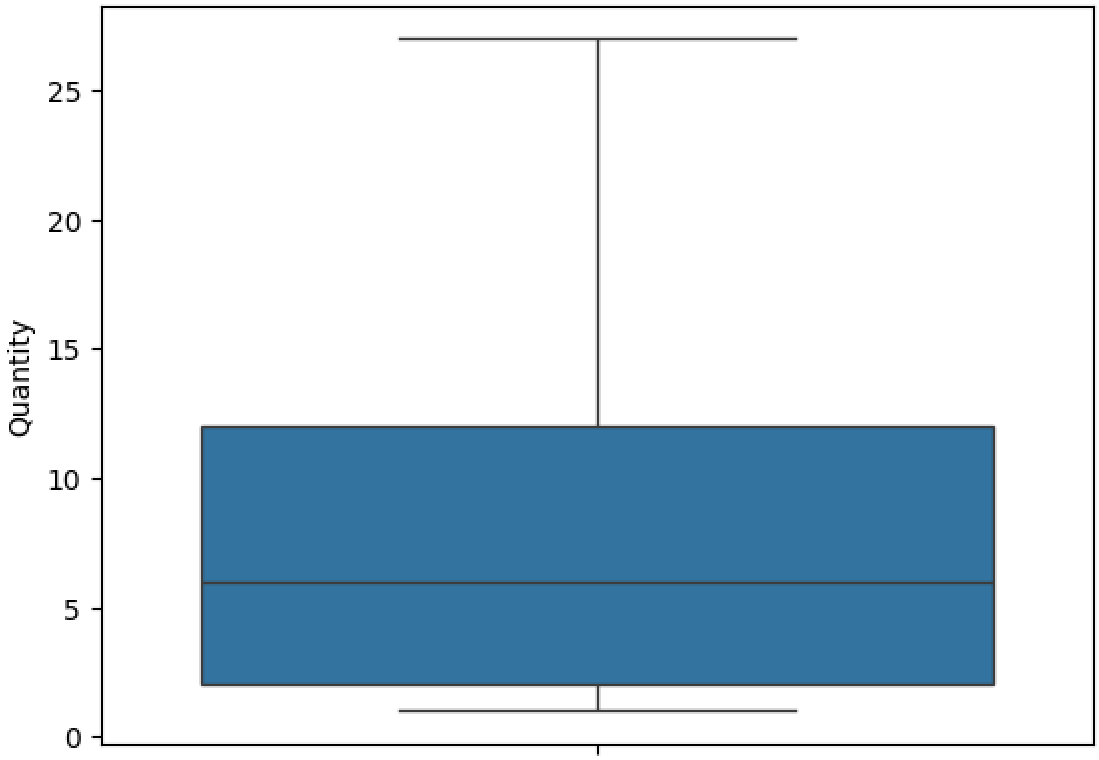

In [11]:
```
# From the quality boxplot, it looks like there are quite some outliers, anc
# doesn't make sense. This step will eliminate those values.
retail = retail[retail['Quantity'] >= 0]
retail.shape
```

Out[11]: (397924, 8)

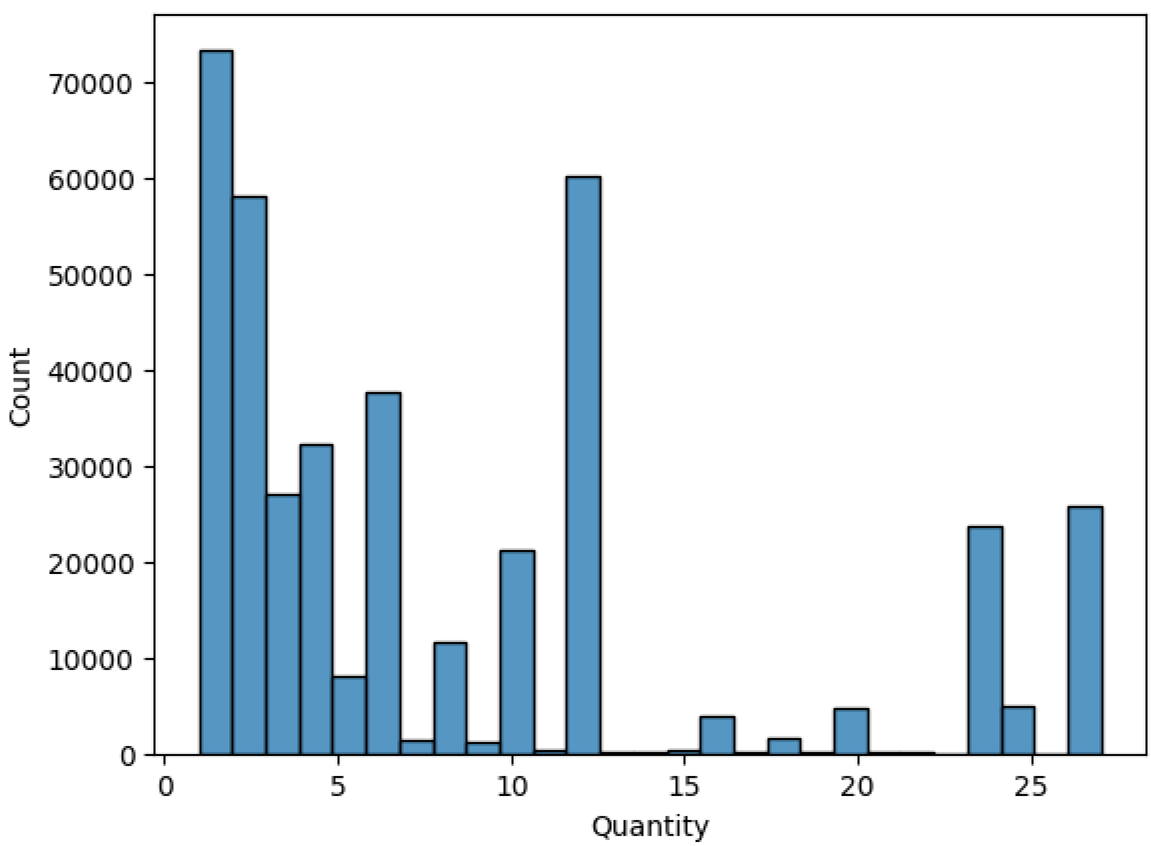
In [12]:
```
# After dropping those values, eliminate the outliers.
quantity = retail["Quantity"]

Q1 = np.percentile(quantity, 25)
Q3 = np.percentile(quantity, 75)

IQR = Q3 - Q1

upper_whisker = Q3 + 1.5 * IQR

print("Upper whisker:", upper_whisker)
```

```
Upper whisker: 27.0
```

In [13]:
```
retail.loc[retail["Quantity"] > 27, "Quantity"] = 27
sns.boxplot(retail["Quantity"])
plt.show()
```




In [14]:
```
sns.histplot(retail["Quantity"] , bins = 27)
plt.show()
```



# Step 4. Perform Analysis Using Unsupervised Learning Models of your Choice, Present Discussion, and Conclusions

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 [15]:
```
# 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[15]:

| | CustomerID | Amount |
|---|---|---|
| 0 | 12346.0 | 28.08 |
| 1 | 12347.0 | 3973.79 |
| 2 | 12348.0 | 850.07 |
| 3 | 12349.0 | 1735.05 |
| 4 | 12350.0 | 334.40 |

In [16]:
```
# 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[16]:

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

In [17]:
```
# Merging the two dfs

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

Out[17]:

| | CustomerID | Amount | Frequency |
|---|---|---|---|
| **0** | 12346.0 | 28.08 | 1 |
| **1** | 12347.0 | 3973.79 | 182 |
| **2** | 12348.0 | 850.07 | 31 |
| **3** | 12349.0 | 1735.05 | 73 |
| **4** | 12350.0 | 334.40 | 17 |

In [18]:
```
# New Attribute : Recency

# Convert to datetime to proper datatype

retail['InvoiceDate'] = pd.to_datetime(retail['InvoiceDate'],format='%d-%m-%
# Compute the maximum date to know the last transaction date

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

Out[18]:
```
Timestamp('2011-12-09 12:50:00')
```

In [19]:
```
# Compute the difference between max date and transaction date

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

Out[19]:

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

In [20]:

```
# 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[20]:

| | CustomerID | Diff |
|---|---|---|
| **0** | 12346.0 | 325 days 02:49:00 |
| **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 [21]:

```
# Extract number of days only

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

Out[21]:

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

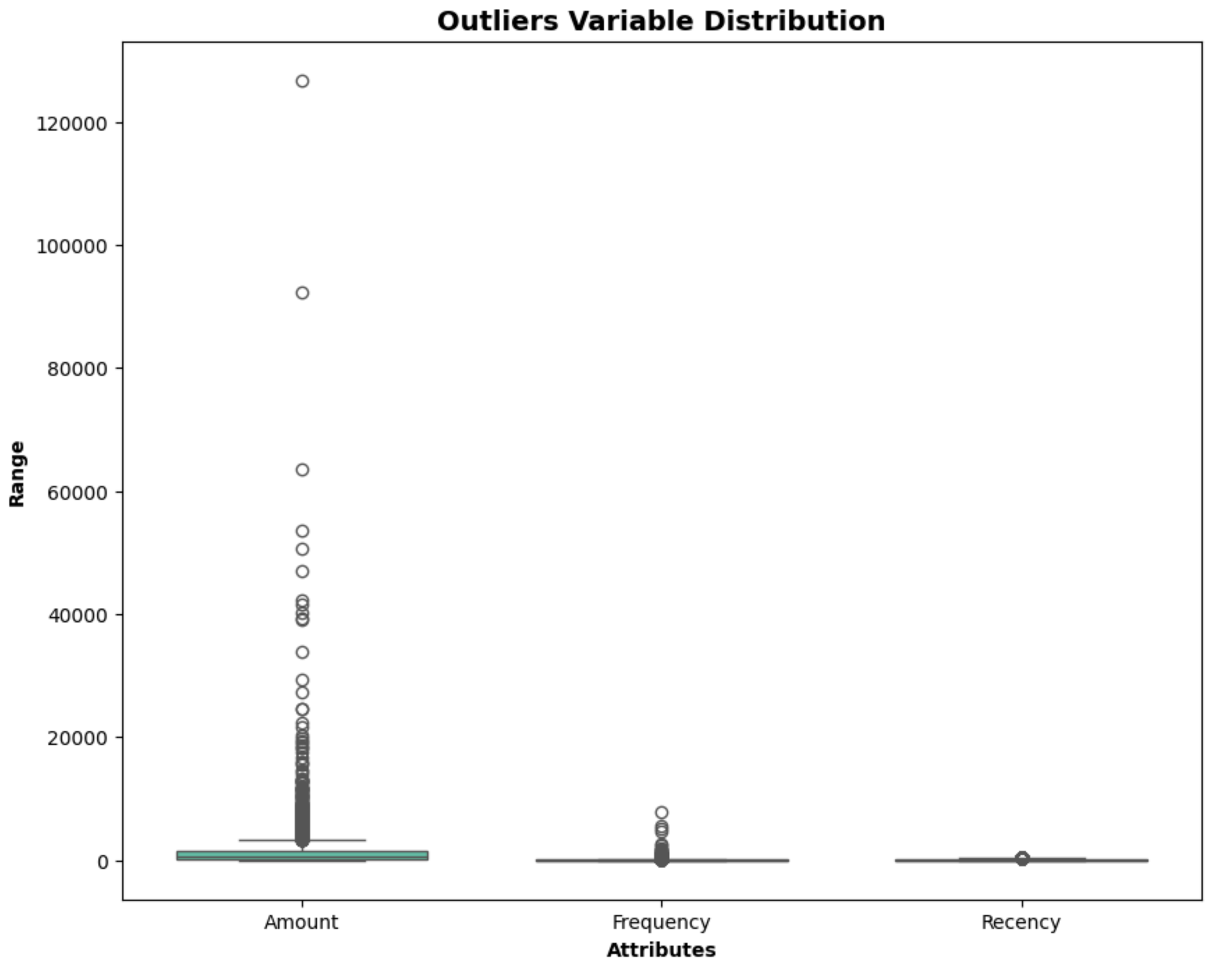
In [22]:
```
# Merge tha dataframes to get the final RFM dataframe

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

Out[22]:

| | CustomerID | Amount | Frequency | Recency |
|---|---|---|---|---|
| **0** | 12346.0 | 28.08 | 1 | 325 |
| **1** | 12347.0 | 3973.79 | 182 | 1 |
| **2** | 12348.0 | 850.07 | 31 | 74 |
| **3** | 12349.0 | 1735.05 | 73 | 18 |
| **4** | 12350.0 | 334.40 | 17 | 309 |

In [23]:
```
# 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,sat
plt.title("Outliers Variable Distribution", fontsize = 14, fontweight = 'bol
plt.ylabel("Range", fontweight = 'bold')
plt.xlabel("Attributes", fontweight = 'bold')
```

Out[23]:
```
Text(0.5, 0, 'Attributes')
```




In [24]:
```
# 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)]
```

In [25]:
```
# Rescaling the attributes using standardization scaling so that they are co

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

# Instantiate
scaler = StandardScaler()

# fit_transform
```

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

Out[25]: (4277, 3)

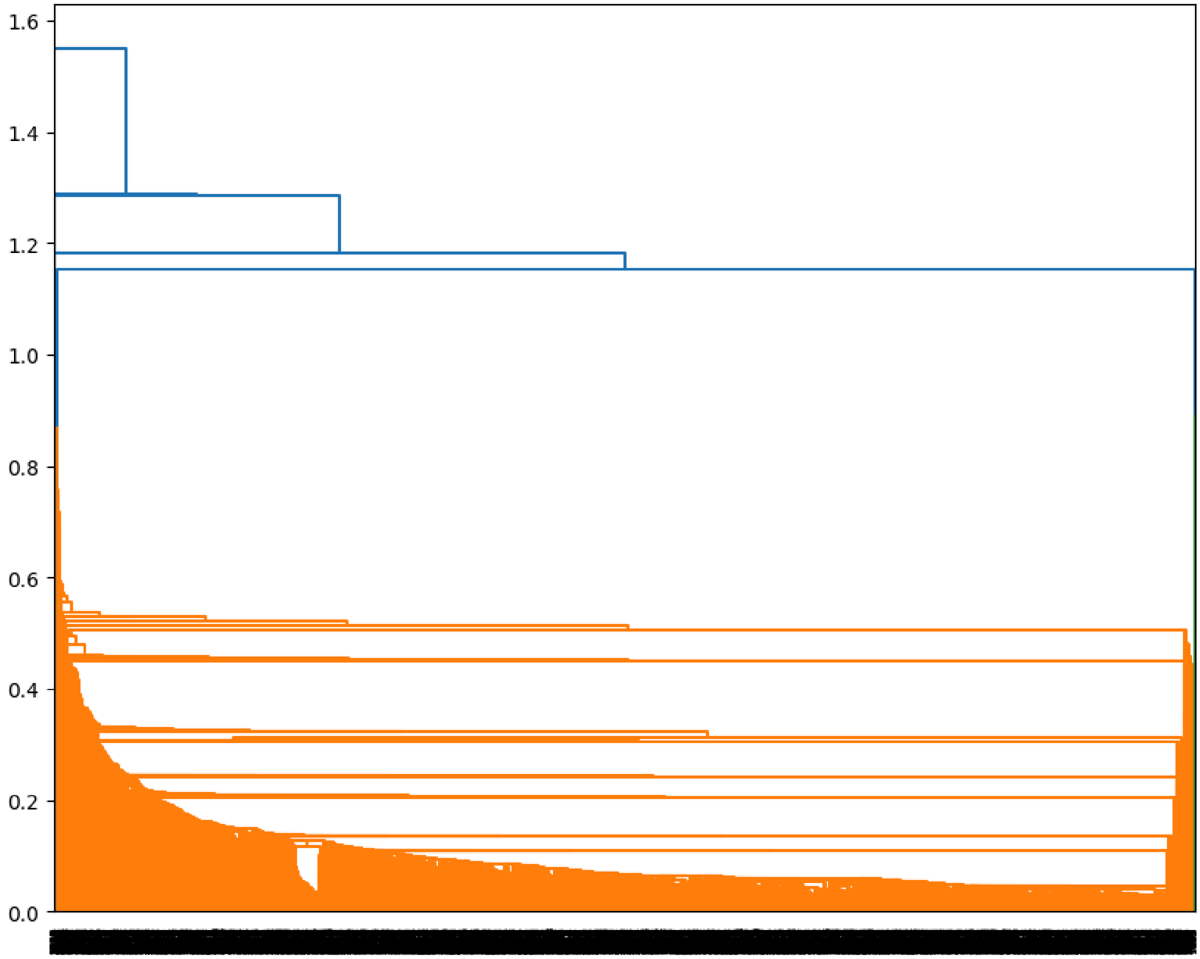
In [26]:
```
rfm_df_scaled = pd.DataFrame(rfm_df_scaled)
rfm_df_scaled.columns = ['Amount', 'Frequency', 'Recency']
rfm_df_scaled.head()
```

Out[26]:

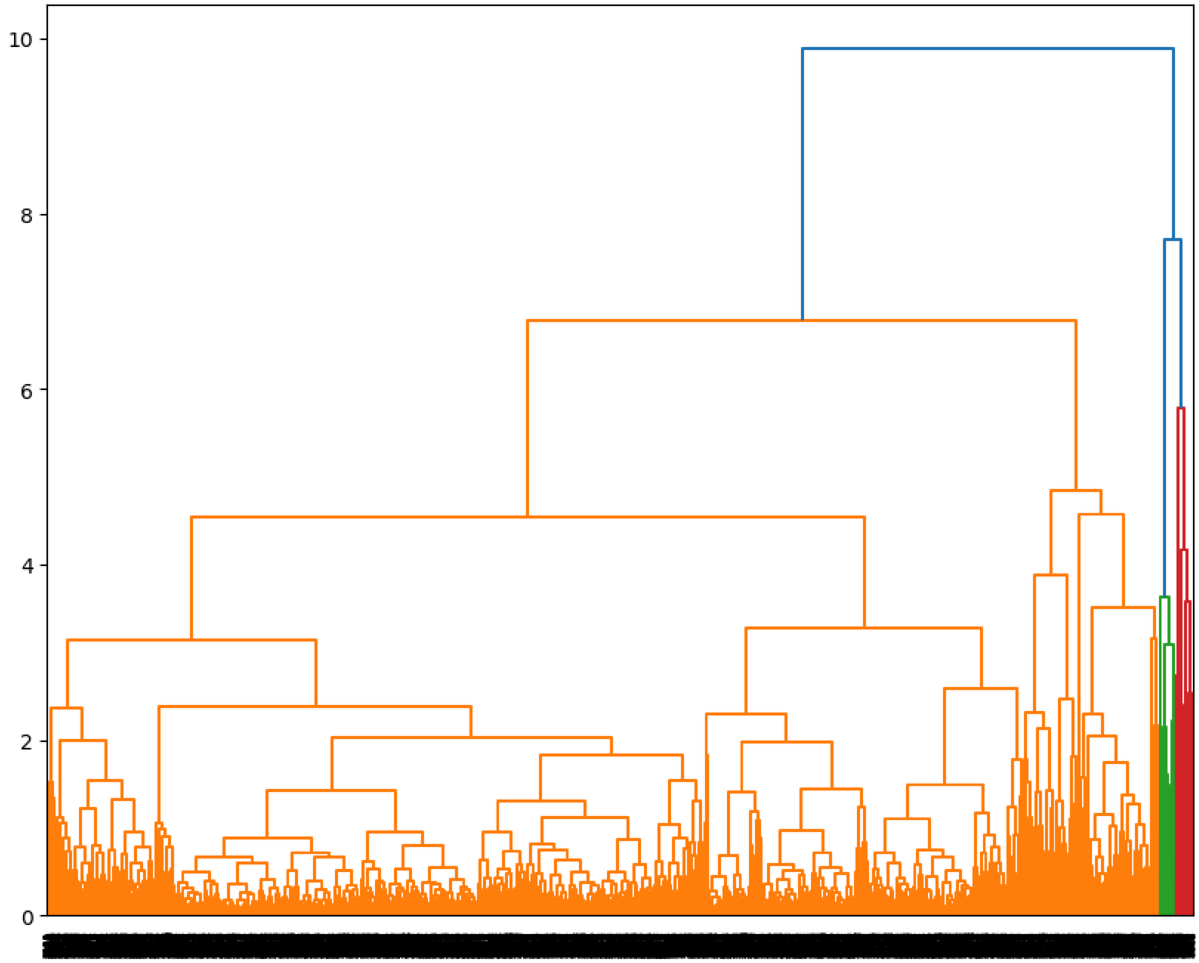
| | Amount | Frequency | Recency |
|---|---|---|---|
| **0** | -0.728238 | -0.771867 | 2.318406 |
| **1** | 1.760418 | 1.080408 | -0.914920 |
| **2** | -0.209789 | -0.464860 | -0.186424 |
| **3** | 0.348390 | -0.035051 | -0.745270 |
| **4** | -0.535035 | -0.608130 | 2.158736 |

## Building the Model with Hierarchical Clustering

In [27]:
```
# Single linkage:

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




In [28]:
```
# Complete linkage

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




In [29]:
```
# 3 clusters
cluster_labels = cut_tree(mergings, n_clusters=3).reshape(-1, )
cluster_labels
```

Out[29]: array([0, 0, 0, ..., 0, 0, 0])

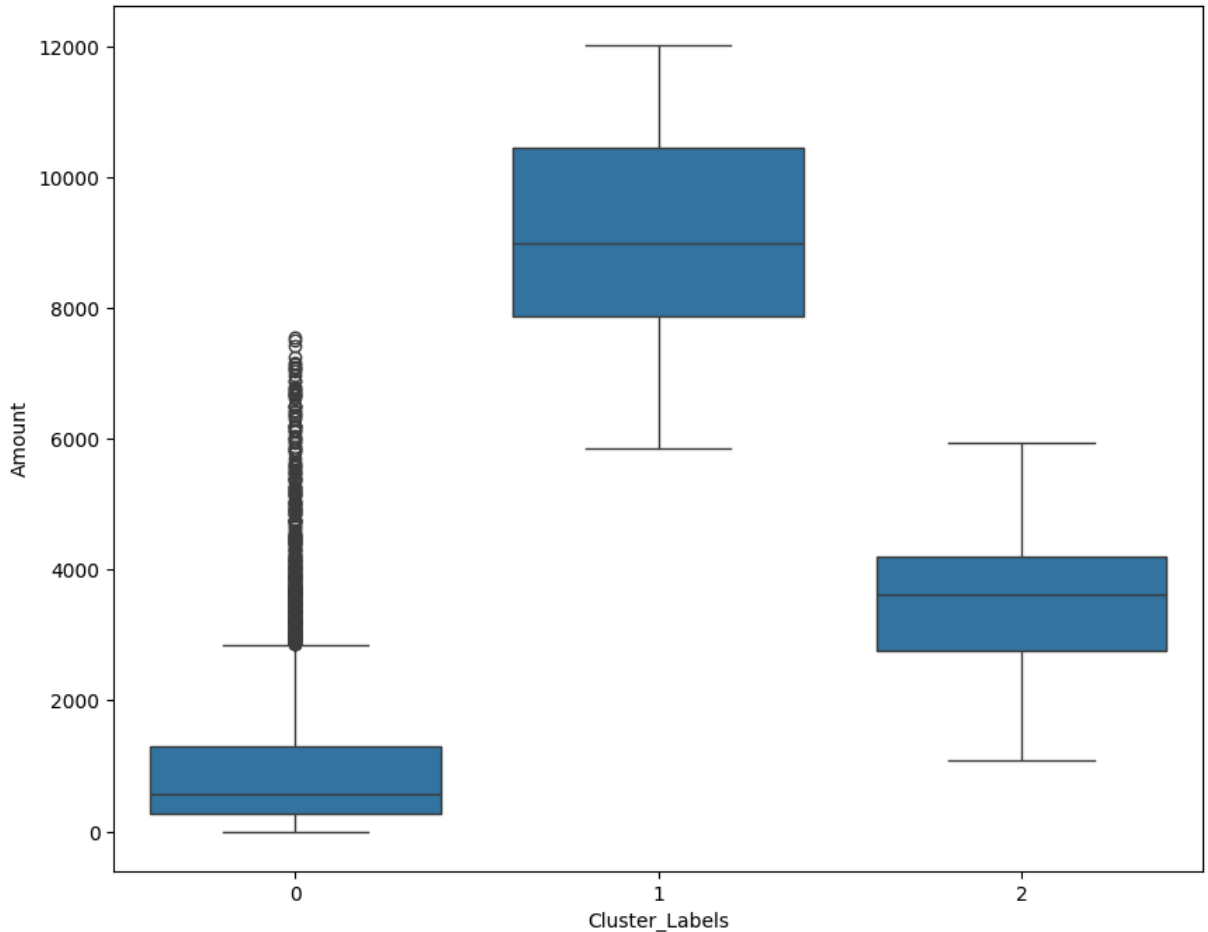
In [30]:
```
# Assign cluster labels

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

Out[30]:

| | CustomerID | Amount | Frequency | Recency | Cluster_Labels |
|---|---|---|---|---|---|
| **0** | 12346.0 | 28.08 | 1 | 325 | 0 |
| **1** | 12347.0 | 3973.79 | 182 | 1 | 0 |
| **2** | 12348.0 | 850.07 | 31 | 74 | 0 |
| **3** | 12349.0 | 1735.05 | 73 | 18 | 0 |
| **4** | 12350.0 | 334.40 | 17 | 309 | 0 |

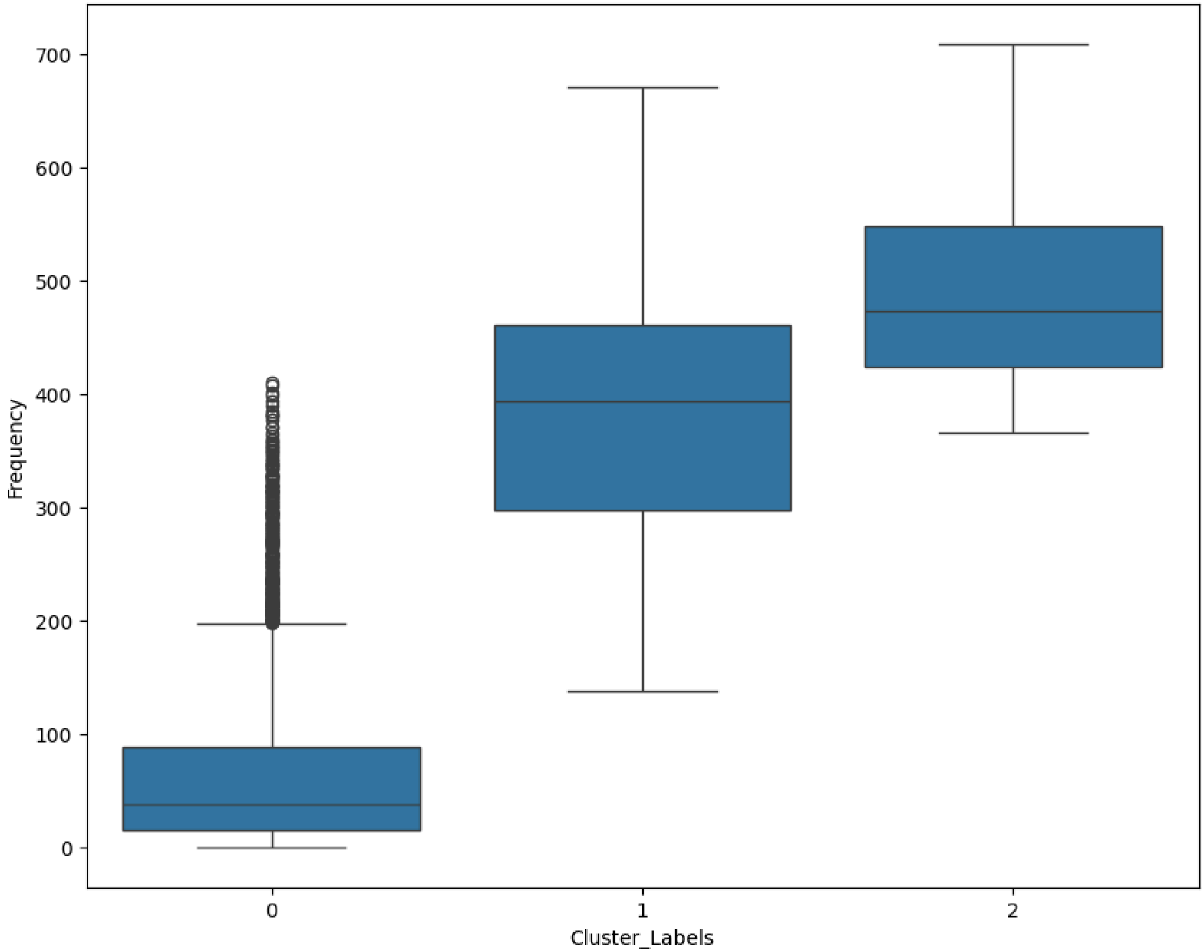
In [31]:
```
# Plot Cluster Id vs Amount

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

Out[31]: <Axes: xlabel='Cluster_Labels', ylabel='Amount'>



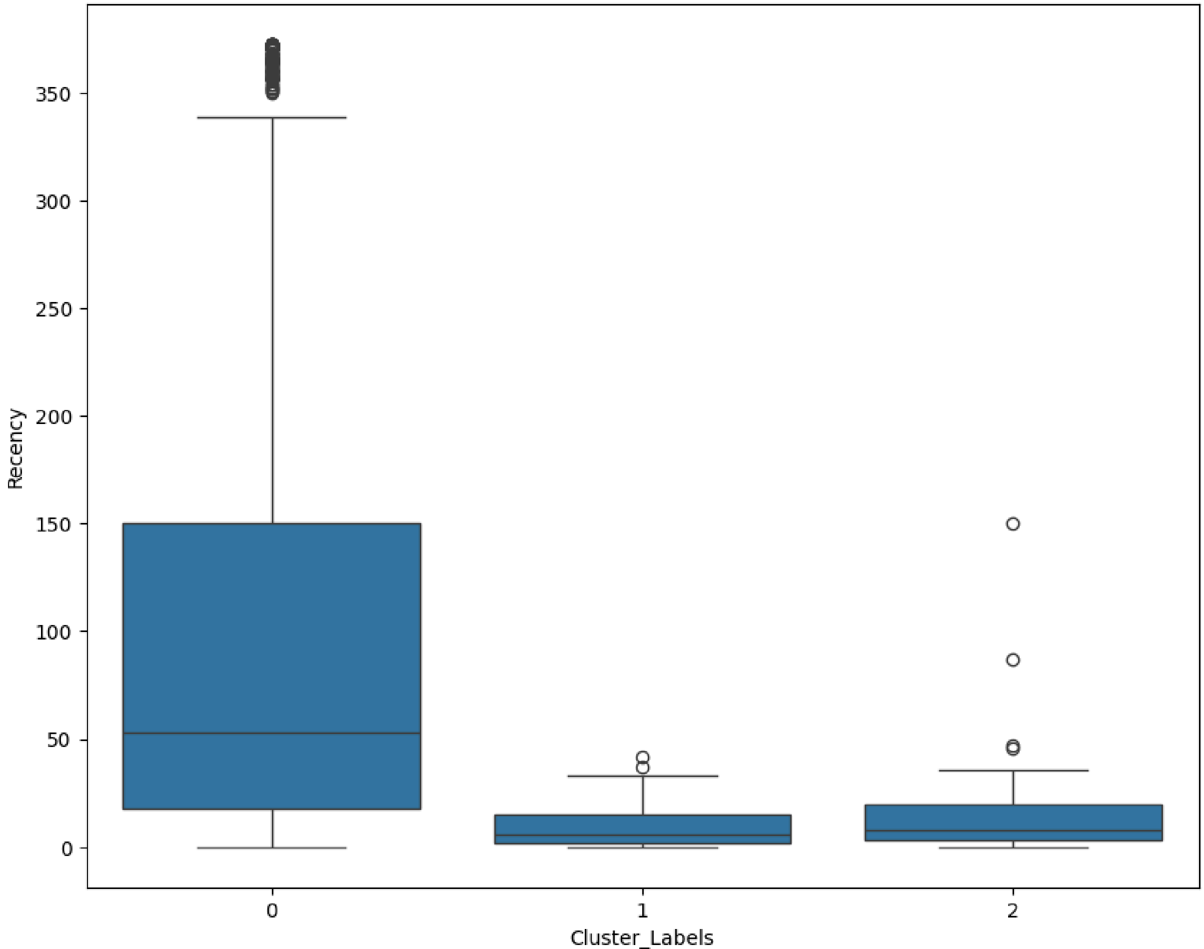

In [32]:
```python
# Plot Cluster Id vs Frequency

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

Out[32]: <Axes: xlabel='Cluster_Labels', ylabel='Frequency'>



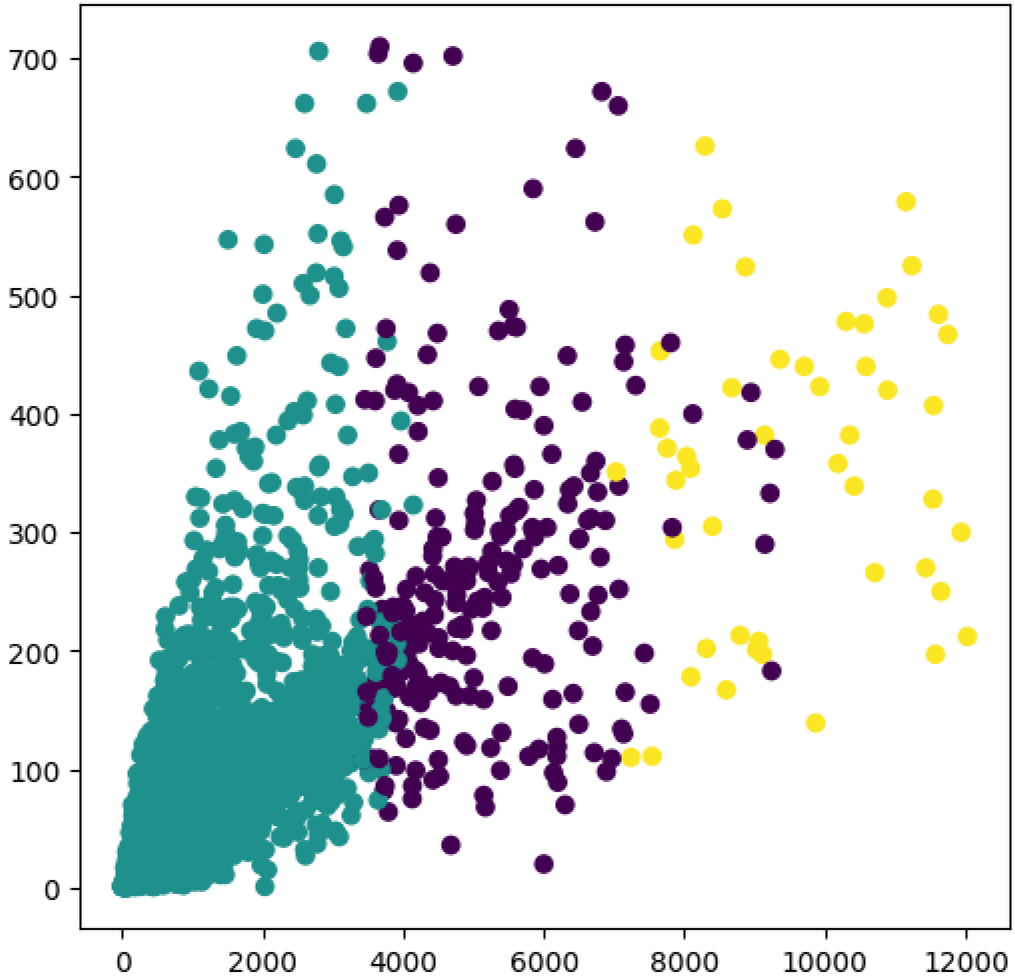

In [33]:
```
# Plot Cluster Id vs Recency

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

Out[33]: <Axes: xlabel='Cluster_Labels', ylabel='Recency'>



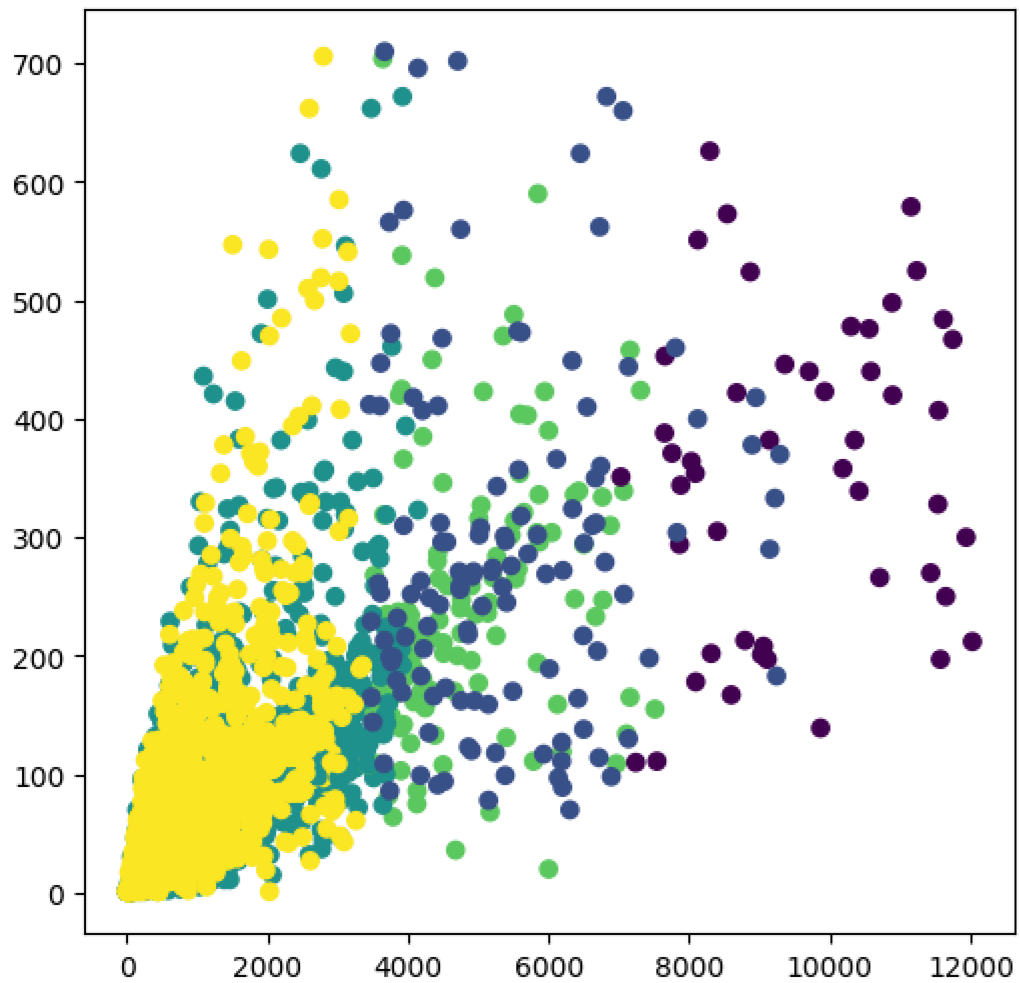

In [35]:
```
#build a model
model_3 = AgglomerativeClustering(n_clusters = 3, linkage='complete').fit(rf
#labels = model.fit_predict(data)
print(model_3.labels_)

# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(rfm['Amount'], rfm['Frequency'],
            c = model_3.fit_predict(rfm), label = model_3.labels_)
plt.show()
```

```
[1 0 1 ... 1 1 1]
```



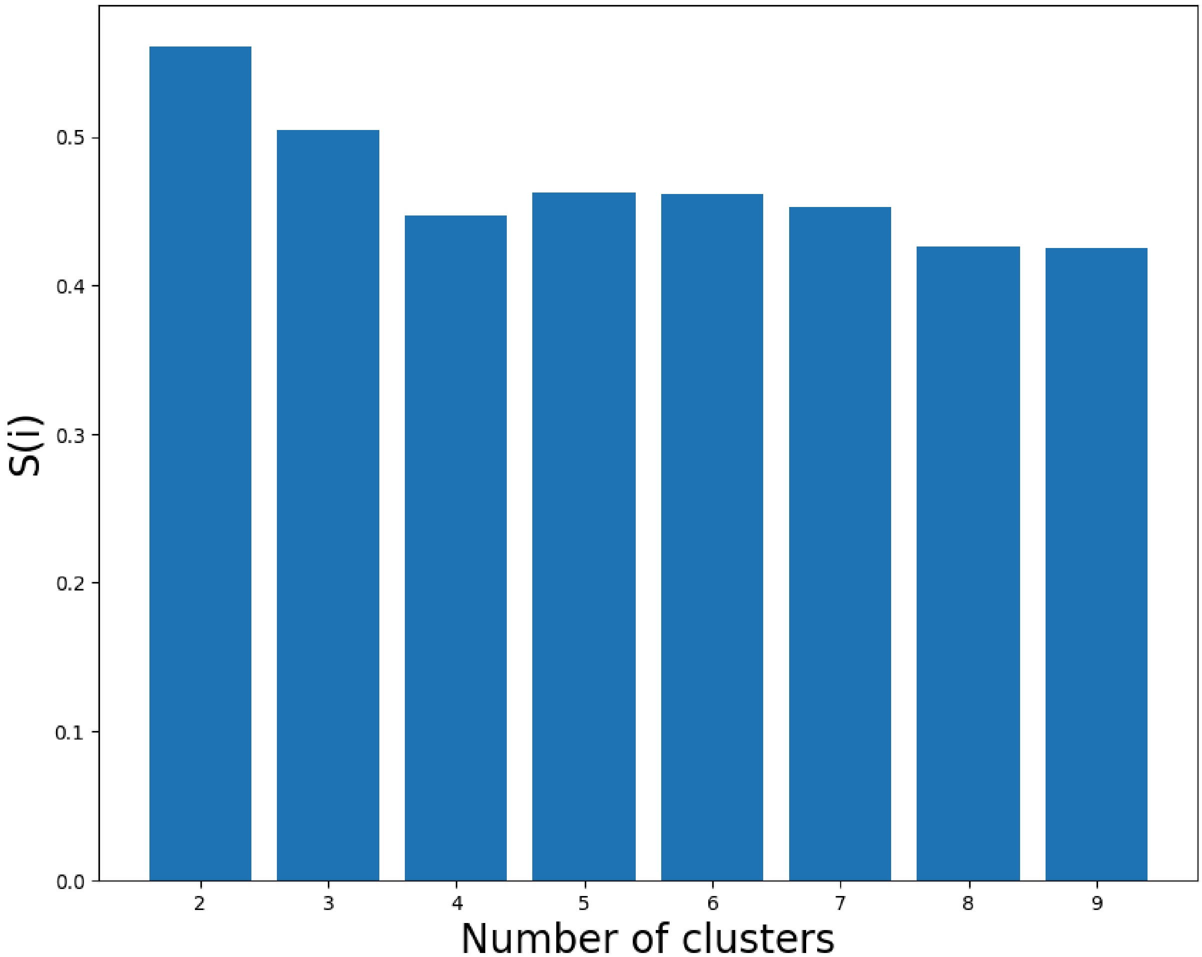

In [36]:
```
#build a model
model_5 = AgglomerativeClustering(n_clusters = 5, linkage='complete').fit(rf
#labels = model.fit_predict(data)
print(model_5.labels_)

# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(rfm['Amount'], rfm['Frequency'],
           c = model_5.fit_predict(rfm), label = model_5.labels_)
plt.show()
```

```
[2 3 2 ... 4 4 4]
```



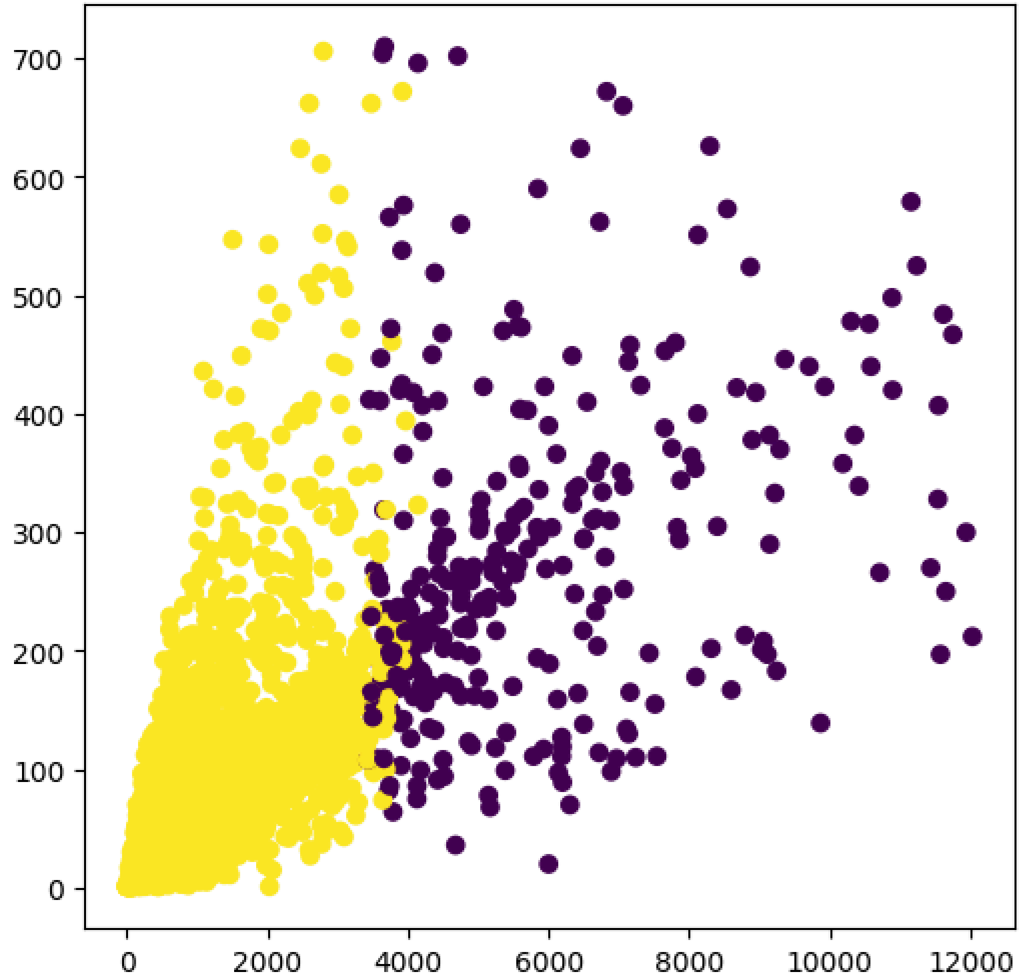

In [37]:
```
k = [2, 3, 4, 5, 6, 7, 8, 9]
model_2 = AgglomerativeClustering(n_clusters = 2, linkage='complete').fit(rf
model_4 = AgglomerativeClustering(n_clusters = 4, linkage='complete').fit(rf
model_6 = AgglomerativeClustering(n_clusters = 6, linkage='complete').fit(rf
model_7 = AgglomerativeClustering(n_clusters = 7, linkage='complete').fit(rf
model_8 = AgglomerativeClustering(n_clusters = 8, linkage='complete').fit(rf
model_9 = AgglomerativeClustering(n_clusters = 9, linkage='complete').fit(rf
# Appending the silhouette scores of the different models to the list
silhouette_scores = []
silhouette_scores.append(
        silhouette_score(rfm, model_2.fit_predict(rfm)))
silhouette_scores.append(
        silhouette_score(rfm, model_3.fit_predict(rfm)))
silhouette_scores.append(
        silhouette_score(rfm, model_4.fit_predict(rfm)))
silhouette_scores.append(
        silhouette_score(rfm, model_5.fit_predict(rfm)))
silhouette_scores.append(
        silhouette_score(rfm, model_6.fit_predict(rfm)))
silhouette_scores.append(
        silhouette_score(rfm, model_7.fit_predict(rfm)))
silhouette_scores.append(
        silhouette_score(rfm, model_8.fit_predict(rfm)))
silhouette_scores.append(
        silhouette_score(rfm, model_9.fit_predict(rfm)))
```

```
# Plotting a bar graph to compare the results
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 20)
plt.ylabel('S(i)', fontsize = 20)
plt.show()
```

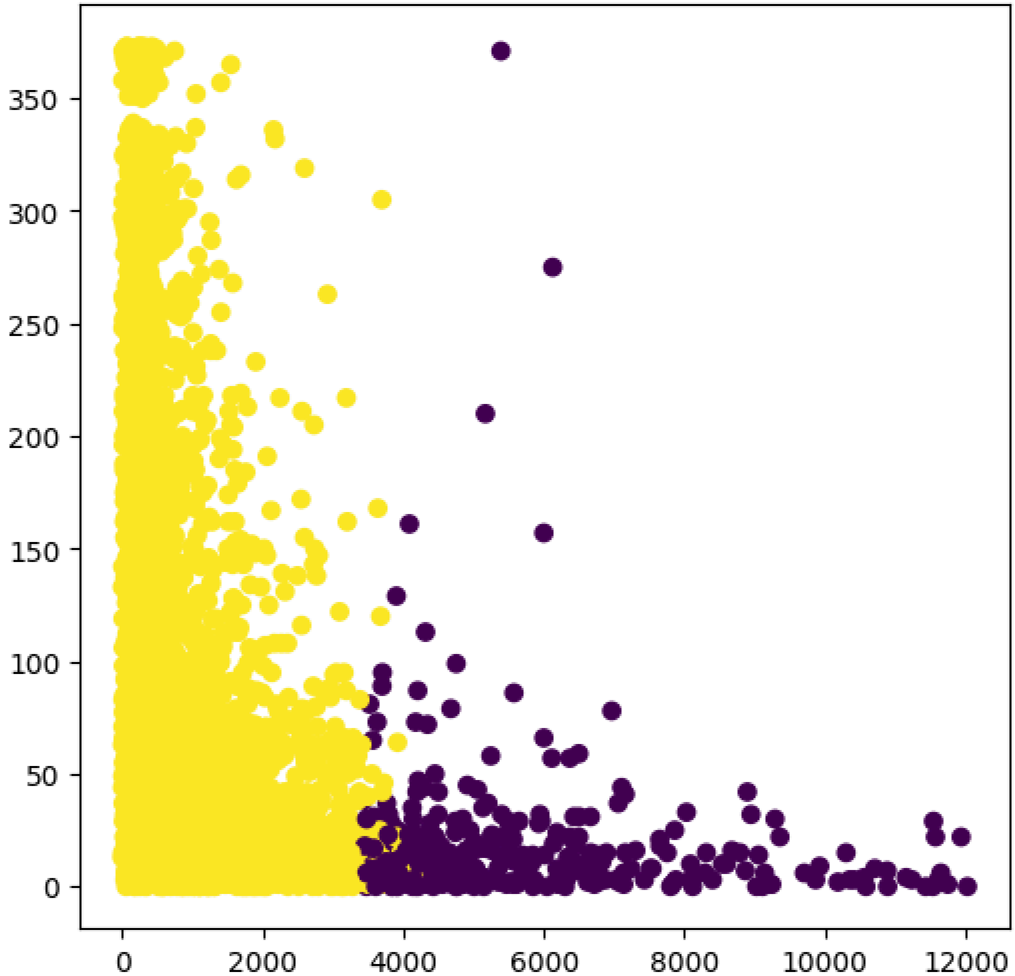



In [38]:
```
# This shows that the optimal number of clusters is 2.

#build a model
model = AgglomerativeClustering(n_clusters = 2, linkage='complete').fit(rfm)
#labels = model.fit_predict(data)
print(model.labels_)

# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(rfm['Amount'], rfm['Frequency'],
            c = model.fit_predict(rfm), label = model.labels_)
plt.show()
```

```
[1 0 1 ... 1 1 1]
```




In [39]:
```
plt.figure(figsize =(6, 6))
plt.scatter(rfm['Amount'], rfm['Recency'],
            c = model.fit_predict(rfm), label = model.labels_)
plt.show()
```




## K Means Clustering

In [40]:
```
# Now we will build a model with K-Means Clustering and compare results.
# k-means with some arbitrary k

kmeans = KMeans(n_clusters=3, max_iter=50)
kmeans.fit(rfm_df_scaled)
```

Out[40]:
```
KMeans
KMeans(max_iter=50, n_clusters=3)
```

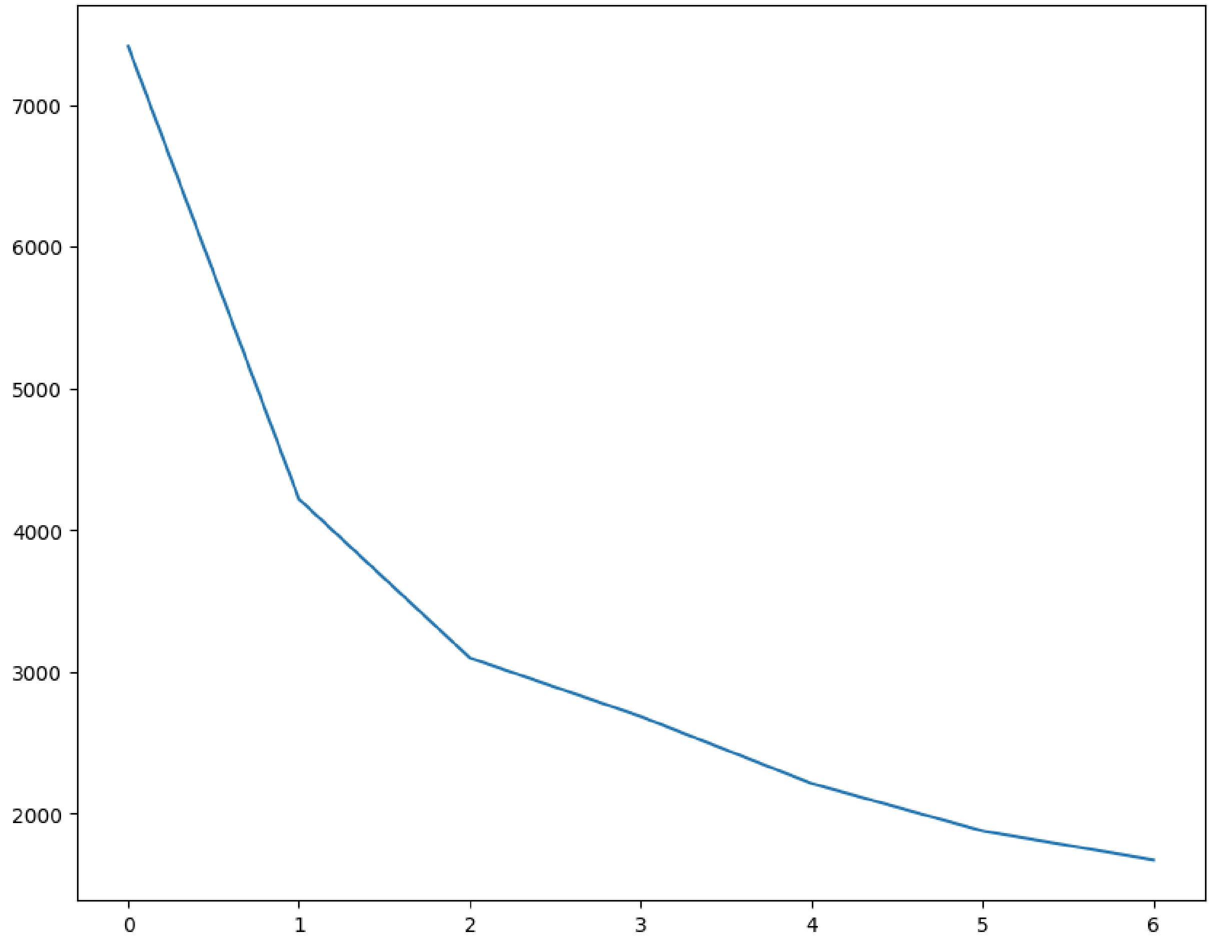
In [41]:
```
kmeans.labels_
```

Out[41]:
```
array([2, 1, 0, ..., 2, 0, 0], dtype=int32)
```

In [42]:
```
# find the optimal number of clusters using 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[42]: [<matplotlib.lines.Line2D at 0x7fdf5f524e20>]




## Discussion and Conclusion

Both the hierarchical clustering model built using the AgglomerativeClustering module and the K-means clustering using the K-means module show that separating the data into two clusters would give the best modeling results.

It shows that we can group the users into two groups, and predict each group's purchasing likelihood of amount, frequency, and recency using this model.

Note that this dataset also has another column with more detailed description of what item was purchased. This will be more valuable data to process and predict. However, due to the time limit and understanding of natural language processing and grouping, this will remain a future item to explore.