Creating a K-means Clustering to Group Customers of Retail Store based on their purchase history.

Reading and Undestanding 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
retail=pd.read_csv(f'C:\\Users\\biswa\\Downloads\\OnlineRetailer customer c
retail.head(4)
```

Out[2]:

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

In [3]:
```
# Checking the Necessary Information
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 [4]:
```
retail.describe()
```

Out[4]:

	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 Cleansing

In [5]:
```
# Calculating the Missing Values
df_null = round(100*(retail.isnull().sum())/len(retail), 2)
df_null
```

Out[5]:
```
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 [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)
```

Data Preparation

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

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

In [8]:
```
# 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[8]:

	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 [9]:
```
# 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[9]:

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

In [10]:
```
# Merging the two dfs
rfm = pd.merge(rfm_m, rfm_f, on='CustomerID', how='inner')
rfm.head()
```

Out[10]:

	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 [11]:
```
# Convert to datetime to proper datatype
retail['InvoiceDate'] = pd.to_datetime(retail['InvoiceDate'],format='%d-%m
```

In [12]:
```
# Compute the maximum date to know the last transaction date
max_date = max(retail['InvoiceDate'])
max_date
```

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

In [13]:
```
# Compute the difference between max date and transaction date
retail['Diff'] = max_date - retail['InvoiceDate']
retail.head(4)
```

Out[13]:

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

In [14]:
```
# 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[14]:

	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 [15]:
```
# Extract number of days only
rfm_p['Diff'] = rfm_p['Diff'].dt.days
rfm_p.head()
```

Out[15]:

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

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

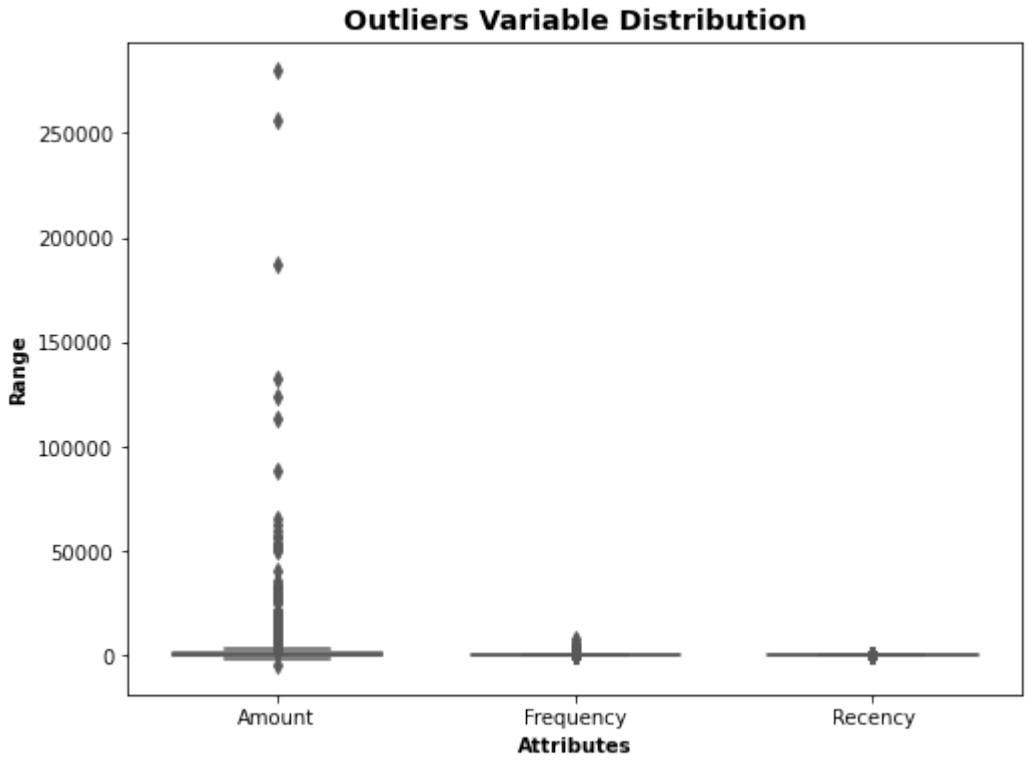
	CustomerID	Amount	Frequency	Recency
0	12346.0	0.00	2	325
1	12347.0	4310.00	182	1
2	12348.0	1797.24	31	74
3	12349.0	1757.55	73	18
4	12350.0	334.40	17	309

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

Statistical
Domain specific

In [17]:
```
# Outlier Analysis of Amount, Frequency and Recency
attributes = ['Amount','Frequency','Recency']
plt.rcParams['figure.figsize'] = [8,6]
sns.boxplot(data = rfm[attributes], orient="v", palette="Set2" ,whis=1.5,sa
plt.title("Outliers Variable Distribution", fontsize = 14, fontweight = 'bo
plt.ylabel("Range", fontweight = 'bold')
plt.xlabel("Attributes", fontweight = 'bold')
```

Out[17]: Text(0.5, 0, 'Attributes')



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

Standardisation Rescaling the Attributes

In [19]:
```
# 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[19]:
```
(4293, 3)
```

In [20]:
```
rfm_df_scaled = pd.DataFrame(rfm_df_scaled)
rfm_df_scaled.columns = ['Amount', 'Frequency', 'Recency']
rfm_df_scaled.head()
```

Out[20]:

	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

K-Means Clustering

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. The algorithm works as follows:

```
(i) First we initialize k points, called means, randomly.
(ii) We categorize each item to its closest mean and we update the
mean's coordinates, which are the averages of the items categorize
d in that mean so far.
(iii) We repeat the process for a given number of iterations and a
t the end, we have our clusters.
```

In [21]:
```
# k-means with some arbitrary k
kmeans = KMeans(n_clusters=4, max_iter=50)
kmeans.fit(rfm_df_scaled)
```

Out[21]:
```
KMeans(max_iter=50, n_clusters=4)
```

In [22]:
```
kmeans.labels_
```

Out[22]:
```
array([2, 0, 1, ..., 2, 1, 1])
```

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

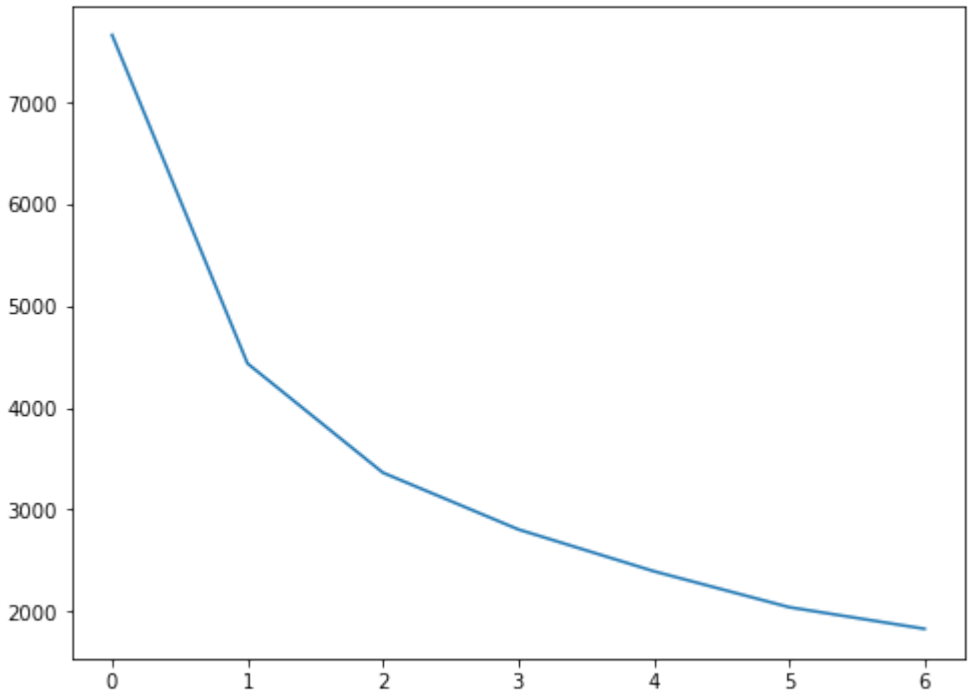
value of k.

In [23]:
```
# 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[23]:
```
[<matplotlib.lines.Line2D at 0x1e4ac21c2b0>]
```



In [24]:
```
# Final model with k=3
kmeans = KMeans(n_clusters=3, max_iter=50)
kmeans.fit(rfm_df_scaled)
```

Out[24]:
```
KMeans(max_iter=50, n_clusters=3)
```

In [25]:
```
kmeans.labels_
```

Out[25]:
```
array([0, 1, 2, ..., 0, 2, 2])
```

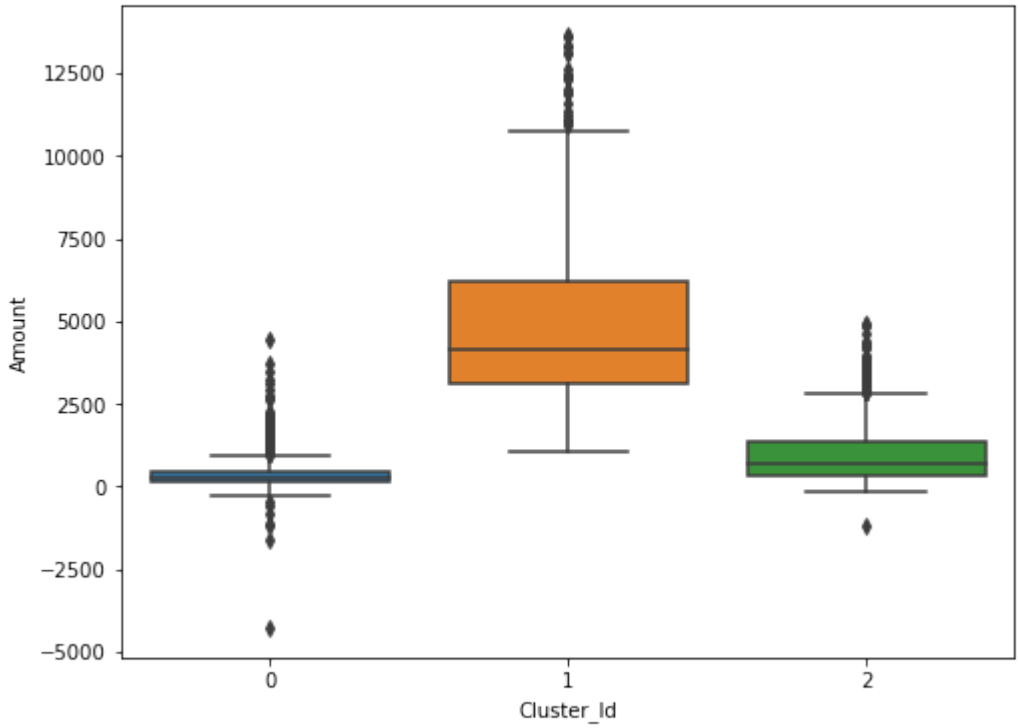
In [26]:
```
# assign the label
rfm['Cluster_Id'] = kmeans.labels_
rfm.head()
```

Out[26]:

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

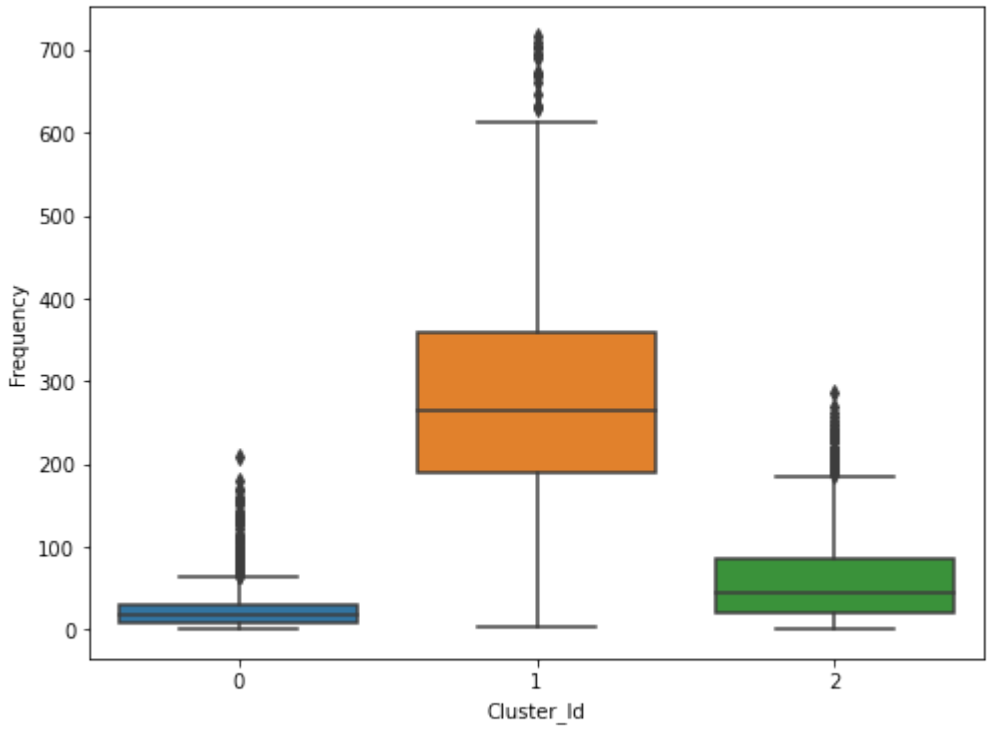
In [27]:
```
# Box plot to visualize Cluster Id vs Frequency
sns.boxplot(x='Cluster_Id', y='Amount', data=rfm)
```

Out[27]: <AxesSubplot:xlabel='Cluster_Id', ylabel='Amount'>



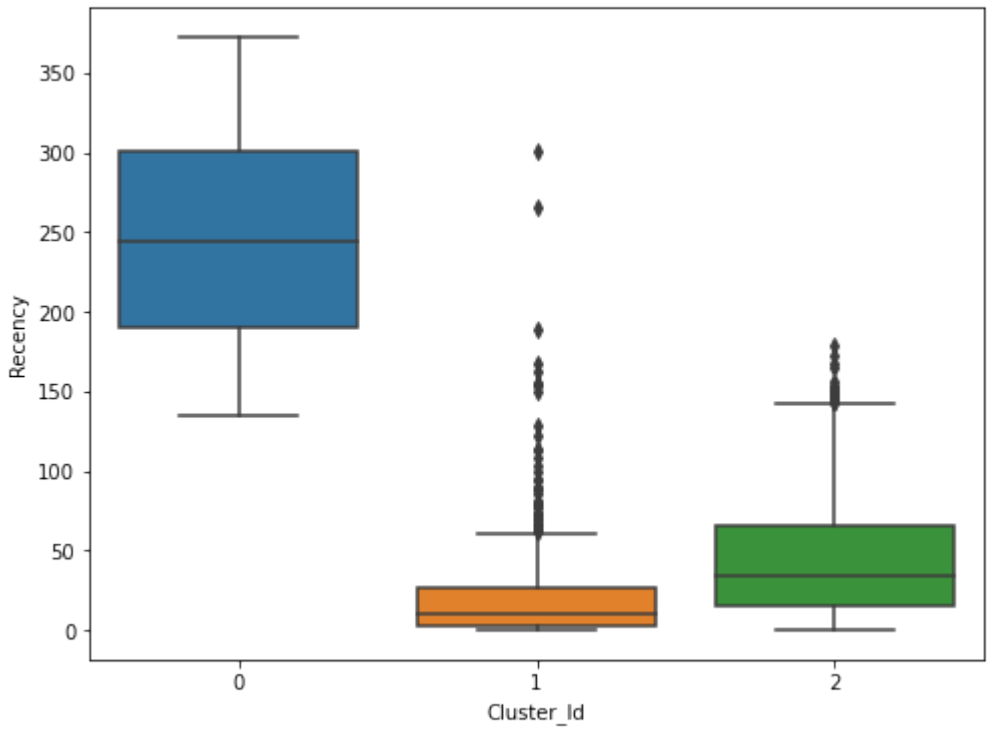
In [28]:
```
# Box plot to visualize Cluster Id vs Frequency
sns.boxplot(x='Cluster_Id', y='Frequency', data=rfm)
```

Out[28]: <AxesSubplot:xlabel='Cluster_Id', ylabel='Frequency'>



In [29]:
```
# Box plot to visualize Cluster Id vs Recency
sns.boxplot(x='Cluster_Id', y='Recency', data=rfm)
```

Out[29]: <AxesSubplot:xlabel='Cluster_Id', ylabel='Recency'>



Conclusions

K-Means Clustering with 3 Cluster Ids

```
(i) Customers with Cluster Id 1 are the customers with high amount
of transactions as compared to other customers.
(ii) Customers with Cluster Id 1 are most frequent buyers.
(iii) Customers with Cluster Id 0 are not recent buyers and hence
least of importance from business point of view.
```

So, from above three points, it's visible that from business point of view Customers with Cluster_Id 1 must have the highest priorities followed by Cluster_Id 2 and Cluster_Id 0.

In [ ]: