In [1]:
```
# Overview
# Online retail is a transnational data set which contains all the transactions occurring between 01/1
2/2010 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 efficientl
y.

# The steps are broadly divided into:
# Step 1: Reading and Understanding the Data
# Step 2: Data Cleansing
# Step 3: Data Preparation
# Step 4: Model Building
# Step 5: Final Analysis
```

In [2]:
```
# Step 1 : Reading and Understanding Data
# 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 [3]:
```
# Reading the data on which analysis needs to be done

retail = pd.read_csv('OnlineRetail.csv', sep=",", encoding="ISO-8859-1", header=0)
retail.head()
```

Out[3]:

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

In [4]:
```
# shape of df

retail.shape
```

Out[4]: (541909, 8)

In [5]:
```
# 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 [6]:
```
# df description

retail.describe()
```

Out[6]:

	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 [7]:
```
# Step 2 : Data Cleansing
# Calculating the Missing Values % contribution in DF

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

Out[7]:
```
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 [8]:
```
# Droping rows having missing values

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

Out[8]: (406829, 8)

In [9]:
```
# Changing the datatype of Customer Id as per Business understanding

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

In [10]:
```
# 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)

# 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[10]:

	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 [11]:
```
# 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[11]:

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

In [12]:
```
# Merging the two dfs

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

Out[12]:

	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 [13]:
```
# New Attribute : Recency

# Convert to datetime to proper datatype

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

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

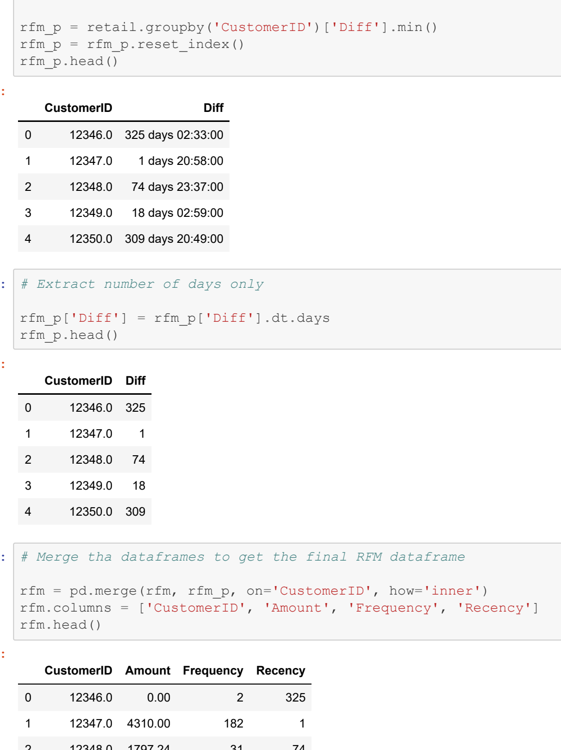
In [18]:
```
# Merge tha dataframes to get the final RFM dataframe

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

In [19]:
```
# There are 2 types of outliers and we will treat outliers as it can skew our dataset
# Statistical
# Domain specific

# 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)]
```

In [21]:
```
# 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.


# 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[21]: (4293, 3)

In [22]:
```
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

In [23]:
```
# 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.

# k-means with some arbitrary k

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

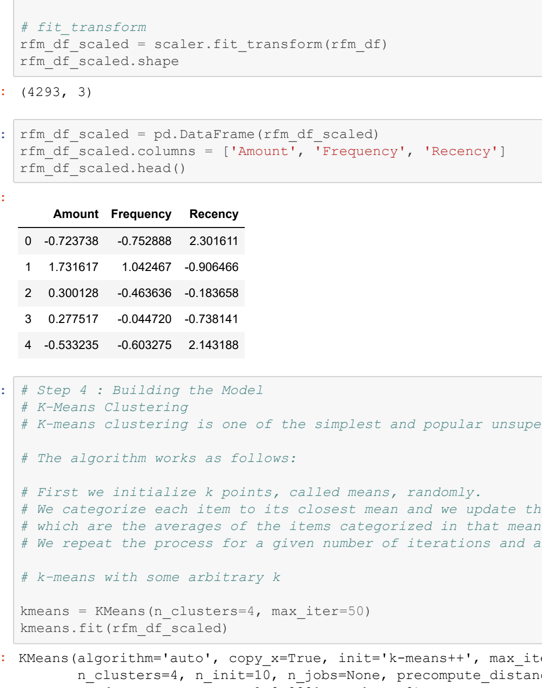
Out[23]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

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

Out[24]: array([0, 3, 2, ..., 0, 2, 2])

In [25]:
```
# 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 opti
mal value of k.

# 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[25]: [<matplotlib.lines.Line2D at 0x2a201747548>]



In [26]:
```
# Silhouette Analysis
# silhouette score=(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 cluste
r,
# A score closer to -1 indicates that the data point is not similar to the data points in its cluster.

# 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 0.5411246404292333
For n_clusters=3, the silhouette score is 0.5084896296141937
For n_clusters=4, the silhouette score is 0.4781841150952288
For n_clusters=5, the silhouette score is 0.4662700564189704
For n_clusters=6, the silhouette score is 0.4171229822428261
For n_clusters=7, the silhouette score is 0.4175904105308637
For n_clusters=8, the silhouette score is 0.40958630945020674
```

In [27]:
```
# Final model with k=3
kmeans = KMeans(n_clusters=3, max_iter=50)
kmeans.fit(rfm_df_scaled)
```

Out[27]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [28]:
```
 kmeans.labels_
```

Out[28]: array([0, 1, 2, ..., 0, 2, 2])

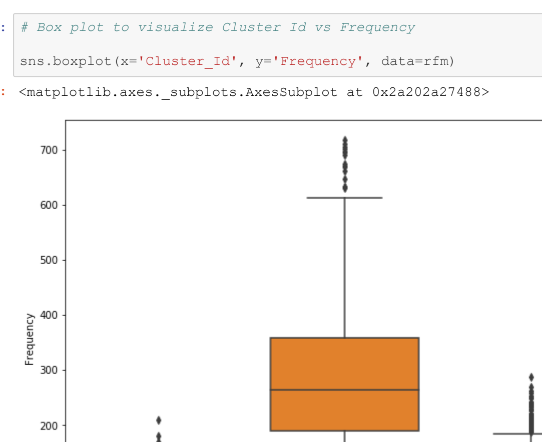
In [29]:
```
# assign the label
rfm['Cluster_Id'] = kmeans.labels_
rfm.head()
```

Out[29]:

	CustomerID	Amount	Frequency	Recency	Cluster_Id
0	12346.0	0.00	2	325	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

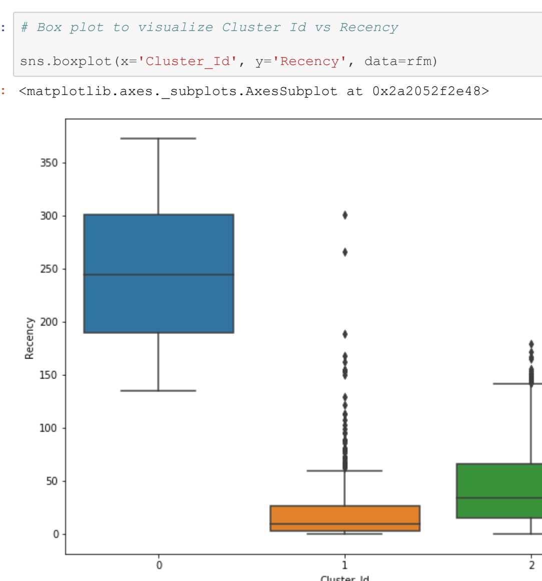
In [30]:
```
# Box plot to visualize Cluster Id vs Frequency

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

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x2a20179b608>



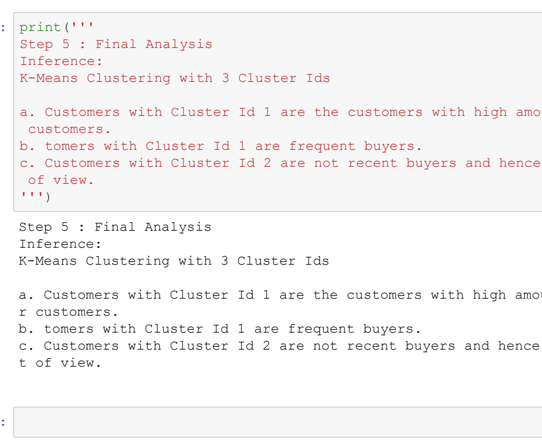
In [31]:
```
# Box plot to visualize Cluster Id vs Frequency

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

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x2a202a27488>



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

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

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x2a2052f2e48>



In [33]:
```
print('''
Step 5 : Final Analysis
Inference:
K-Means Clustering with 3 Cluster Ids

a. Customers with Cluster Id 1 are the customers with high amount of transactions as compared to other
 customers.
b. tomers with Cluster Id 1 are frequent buyers.
c. Customers with Cluster Id 2 are not recent buyers and hence least of importance from business point
 of view.
''')
```

```
Step 5 : Final Analysis
Inference:
K-Means Clustering with 3 Cluster Ids

a. Customers with Cluster Id 1 are the customers with high amount of transactions as compared to othe
r customers.
b. tomers with Cluster Id 1 are frequent buyers.
c. Customers with Cluster Id 2 are not recent buyers and hence least of importance from business poin
t of view.
```

In [ ]: