Table of Contents

Context

Online retail is a transnational data set 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.

Objectives:

Our main objective is to segment customers based on RFM(Recency, Frequency and Monetary) which can be later used in targeting campaigns.

The following are the plan of action to be employed for our task:

1. Data Dictionary: Reading and Defining the data with summary statistics
2. Exploratory Data Analysis: Data Cleaning and feature creation.
3. Compare models to find the best segmentation.

Data Dictionary:

Libraries used:

```python
# 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 os
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

os.chdir('data')
from colorsetup import colors, palette
sns.set_palette(palette)

# For clustering
import sklearn
from sklearn.preprocessing import MinMaxScaler
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
```

- Reading the Data

```python
data = pd.read_csv('OnlineRetail.csv',sep=",",  encoding="ISO-8859-1")
data.head()
```

	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

- Data dictionary

```python
data.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
```

- Summary Statistics

```
data.describe()
```

	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.describe(include='O')
```

	InvoiceNo	StockCode	Description	InvoiceDate	Country
count	541909	541909	540455	541909	541909
unique	25900	4070	4223	23260	38
top	573585	85123A	WHITE HANGING HEART T-LIGHT HOLDER	31-10-2011 14:41	United Kingdom
freq	1114	2313	2369	1114	495478

From the summary statistics, we can observe that 'CustomerID' need to be defined as a string and InvoiceDate need to be converted to Datetime due to the nature of the variables.

Exploratory Data Analysis:

- Checking for Nulls

```
(data.isna().sum()*100)/data.shape[0]
```

```
InvoiceNo       0.000000
StockCode       0.000000
Description     0.268311
Quantity        0.000000
InvoiceDate     0.000000
UnitPrice       0.000000
CustomerID     24.926694
Country         0.000000
dtype: float64
```

We observe that ~25% of the records are missing 'CustomerID' . Due to the high count, it would have made sense to impute this variable but I am choosing not to do any imputing because it may cause some spurious groupings to be formed during the modelling phase.

```
data = data.dropna()
```

After dropping the records, we see the nulls are gone.

```
(data.isna().sum()*100)/data.shape[0]
```

```
InvoiceNo      0.0
StockCode      0.0
Description    0.0
Quantity       0.0
InvoiceDate    0.0
UnitPrice      0.0
CustomerID     0.0
Country        0.0
dtype: float64
```

- Changing data type for 'CustomerID' and 'InvoiceDate'.

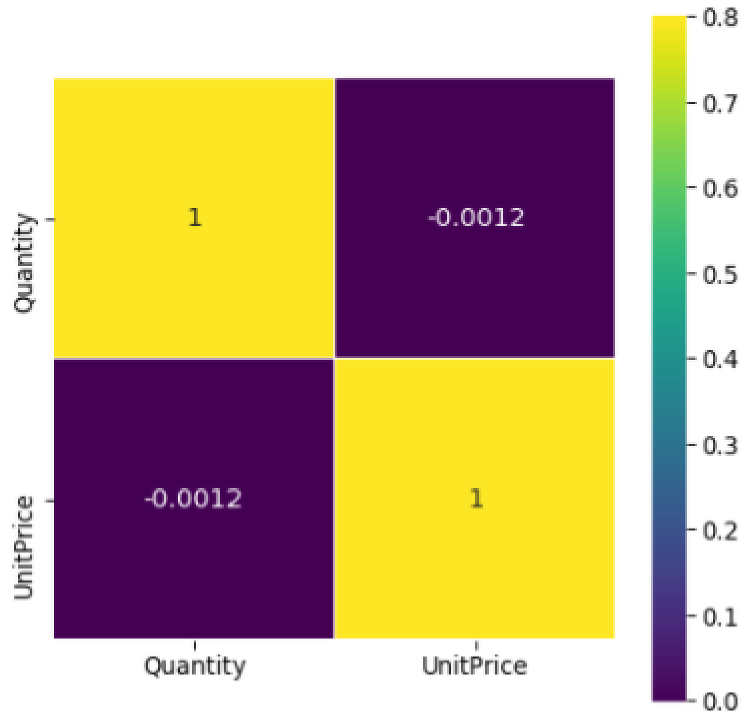
```
data['CustomerID'] = data['CustomerID'].astype(str)
```

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

- Correlation Matrix show the variables are not very weakly correlated to one another.

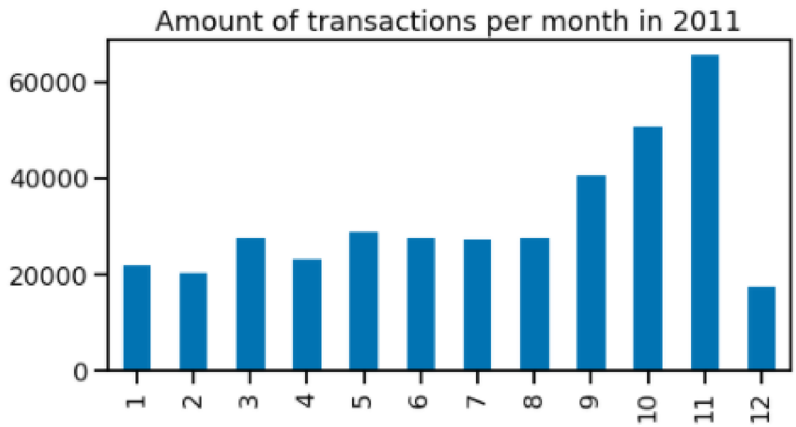
```
corr = data.corr()
plt.figure(figsize=(7, 7))
sns.color_palette("viridis")
sns.heatmap(corr, vmax=.8, linewidths=0.01,
            square=True,annot=True,cmap='viridis',linecolor="white")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x27588c69308>
```



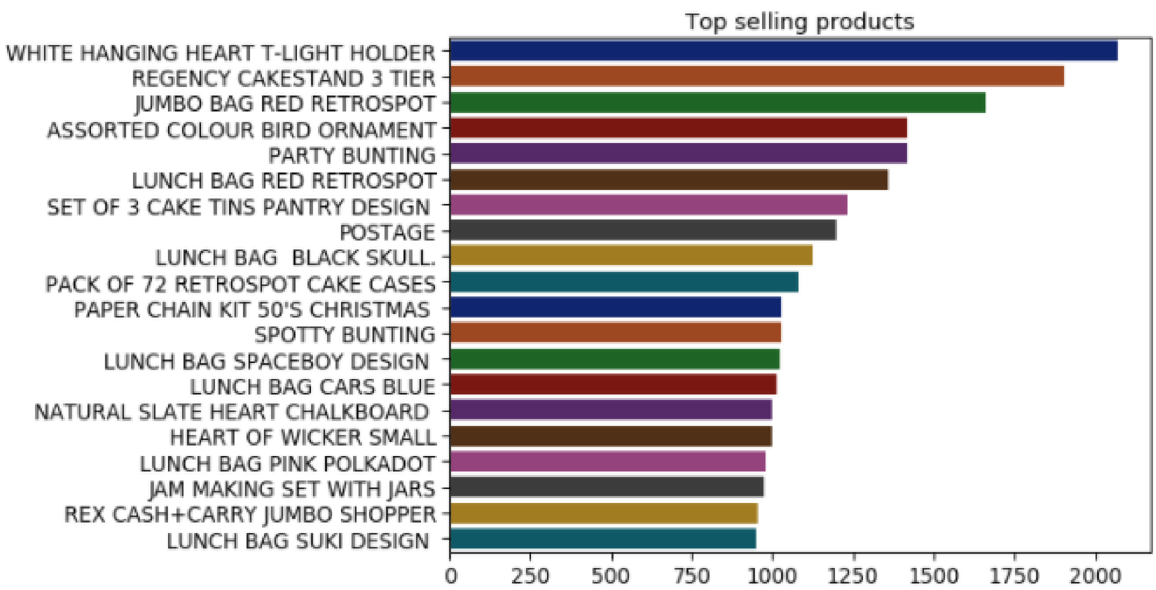
- Looking at transactional volume by month doesn't indicate anything unusual as it is expected that sales usually go up around the holidays.

```
sns.set_context("talk")
sns.set_palette("colorblind")
plt.figure(figsize=(8,4))
data[data.InvoiceDate.dt.year==2011].InvoiceDate.dt.month.value_counts(sort=False).plot(kind='bar')
plt.title("Amount of transactions per month in 2011")
plt.show()
```



- Looking at the highest sold products:

```
top_products = data['Description'].value_counts()[:20]
plt.figure(figsize=(8,6))
sns.set_context("paper", font_scale=1.5)
sns.barplot(y = top_products.index,
            x = top_products.values,
           palette='dark')
plt.title("Top selling products")
plt.show()
```



- For our RFM analysis, we need to create new variables to account for the Last-time customer made purchase in days, what is the frequency for customer making purchase in count and what is the Amount they have purchased so far in summation.

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

	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

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

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

```
max_dt = max(data['InvoiceDate'])
data['Diff'] = max_dt - data['InvoiceDate']
data['Diff']=data['Diff'].dt.days
data.head()

LatestDt = data.groupby('CustomerID')['Diff'].min()
LatestDt = LatestDt.reset_index()
LatestDt.columns = ['CustomerID','Recency']
LatestDt.head()
```

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

Combining them with CustomerID

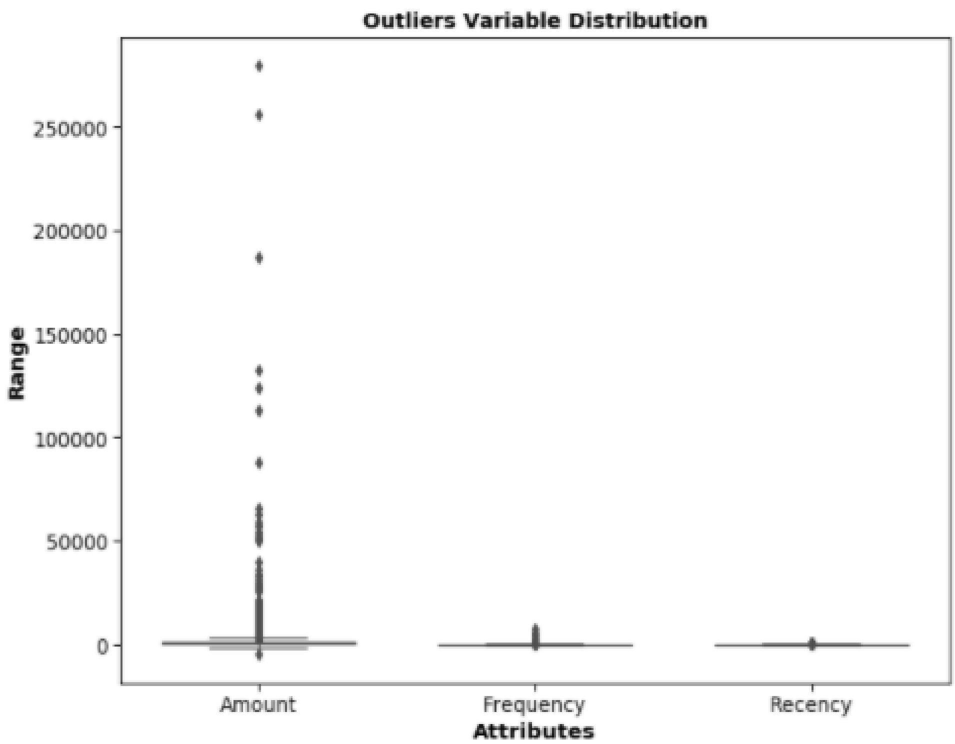
```python
data_new = pd.merge(Amount, Freq, how='inner', on ='CustomerID')
data_new = pd.merge(data_new, LatestDt, how='inner', on ='CustomerID')
data_new.head()
```

	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

- Checking for Outliers

```python
attributes = ['Amount','Frequency','Recency']
plt.rcParams['figure.figsize'] = [10,8]
sns.boxplot(data = data_new[attributes], orient="v", palette="deep" ,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')
```

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



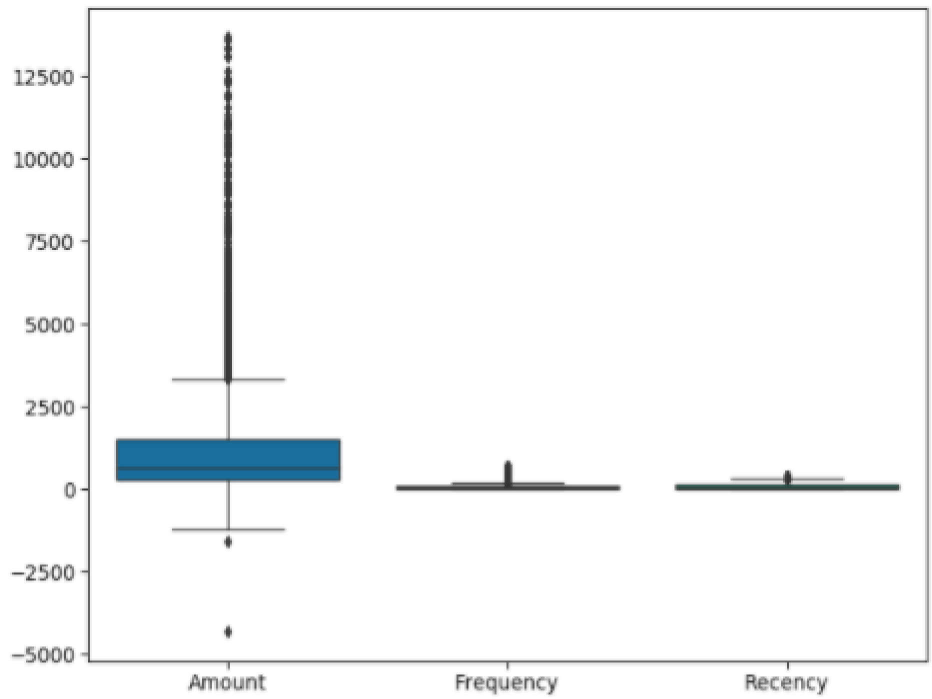
Observing the box plot we have to do some outlier treatment, which we will do by filtering values within 1.5 times the inter-quantile range

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

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

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

Doing so we have removed most of our extreme outliers.

```
sns.boxplot(data=data_new[attributes])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x275c02d7888>
```



Now we are ready to start with the modelling process.

Model Evaluation:

- Feature Scaling:

```python
from sklearn.preprocessing import MinMaxScaler

ScaleVar = data_new[['Amount', 'Frequency', 'Recency']]

mms = StandardScaler()

for col in attributes:
    data_new[col] = mms.fit_transform(ScaleVar)

data_new.head()
```

	CustomerID	Amount	Frequency	Recency
0	12346.0	-0.723738	-0.723738	-0.723738
1	12347.0	1.731617	1.731617	1.731617
2	12348.0	0.300128	0.300128	0.300128
3	12349.0	0.277517	0.277517	0.277517
4	12350.0	-0.533235	-0.533235	-0.533235

We will keep only the feature we want for our model.

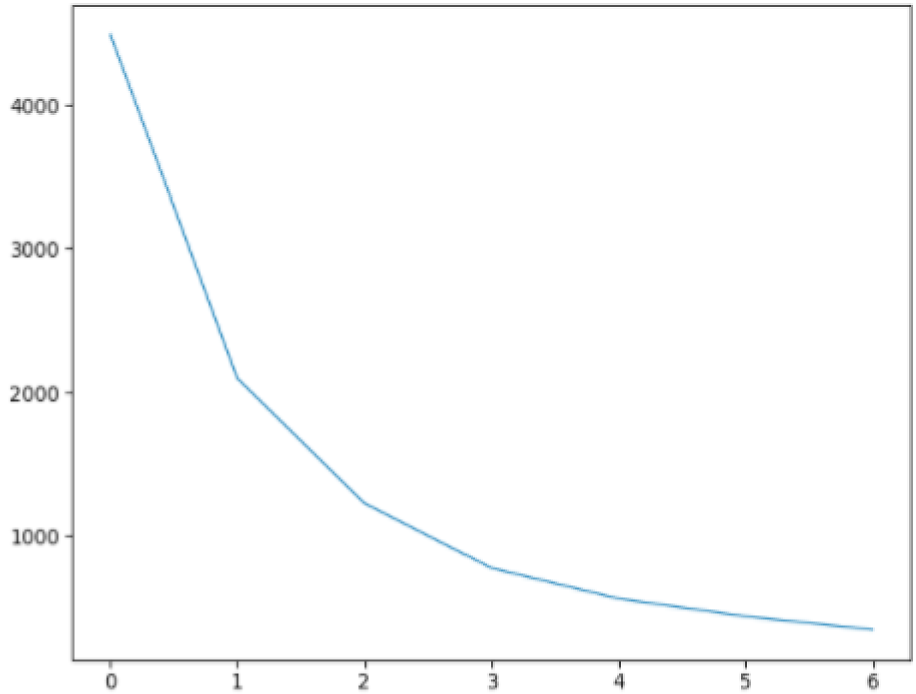
```python
cluster_df = data_new.drop('CustomerID', axis=1)
cluster_df.head()
```

	Amount	Frequency	Recency
0	-0.723738	-0.723738	-0.723738
1	1.731617	1.731617	1.731617
2	0.300128	0.300128	0.300128
3	0.277517	0.277517	0.277517
4	-0.533235	-0.533235	-0.533235

- K-Means
    - Finding Optimal number of clusters using inertia

```
inertia = []
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(cluster_df)

    inertia.append(kmeans.inertia_)

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

```
[<matplotlib.lines.Line2D at 0x275c0cd26c8>]
```



The graph suggests the optimal cluster is 3.

- Finding Optimal number of cluster using the silhouette score

```
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]

for num in range_n_clusters:

    kmeans = KMeans(n_clusters=num, max_iter=50)
    kmeans.fit(cluster_df)

    cluster_labels = kmeans.labels_

    silhouette_avg = silhouette_score(cluster_df, 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 0.4646444032280179
For n_clusters=6, the silhouette score is 0.4171229822428261
For n_clusters=7, the silhouette score is 0.4163434484832087
For n_clusters=8, the silhouette score is 0.4097764149832758
```

The silhouette score suggest 2 cluster to be optimal but we will choose to go with 3 cluster for our modelling.

- Training K-Means with 3 clusters

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

```
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)
```

```
kmeans.labels_
```

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

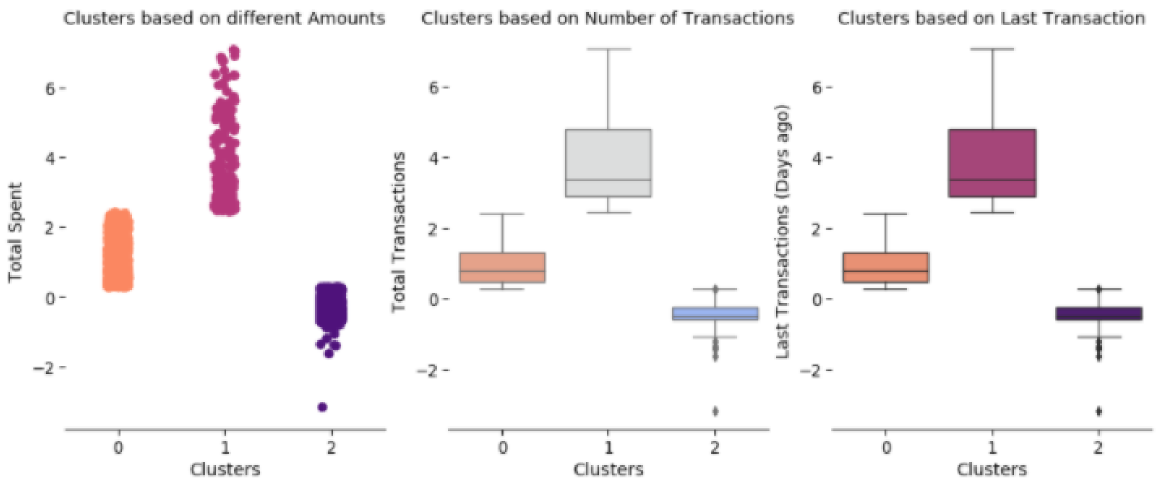
```
data_new['Cluster_Id'] = kmeans.labels_
data_new.head()
```

	CustomerID	Amount	Frequency	Recency	Cluster_Id
0	12346.0	-0.723738	-0.723738	-0.723738	2
1	12347.0	1.731617	1.731617	1.731617	0
2	12348.0	0.300128	0.300128	0.300128	0
3	12349.0	0.277517	0.277517	0.277517	0
4	12350.0	-0.533235	-0.533235	-0.533235	2

- Plotting to view the clusters, we can see that K-Means does a fairly good job of segmenting the customers.

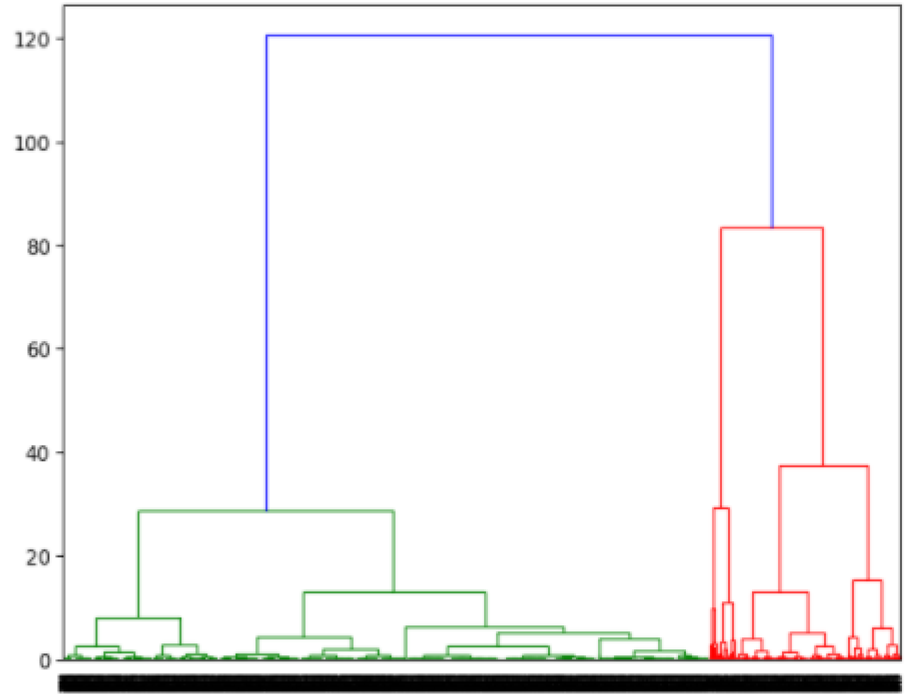
```
fig, ax =plt.subplots(nrows= 1, ncols = 3, figsize= (16,6))
ty=sns.stripplot(x='Cluster_Id', y='Amount', data=data_new, s=8, ax = ax[0], palette='magma_r')
sns.despine(left=True)
ty.set_title('Clusters based on different Amounts')
ty.set_ylabel('Total Spent')
ty.set_xlabel('Clusters')

tt=sns.boxplot(x='Cluster_Id', y='Frequency', data=data_new, ax = ax[1], palette='coolwarm_r')
tt.set_title('Clusters based on Number of Transactions')
tt.set_ylabel('Total Transactions')
tt.set_xlabel('Clusters')

tr=sns.boxplot(x='Cluster_Id', y='Recency', data=data_new, ax = ax[2], palette='magma_r')
tr.set_title('Clusters based on Last Transaction')
tr.set_ylabel('Last Transactions (Days ago)')
tr.set_xlabel('Clusters')
```

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



- Hierarchical Clustering
  - With 'Ward' linkage and computing with the 'euclidean' distance metric, we observe from the dendrogram that cutting the tree into 3 branches will make good segments.

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




```
cluster_labels = cut_tree(mergings, n_clusters=3).reshape(-1, )
cluster_labels
```

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

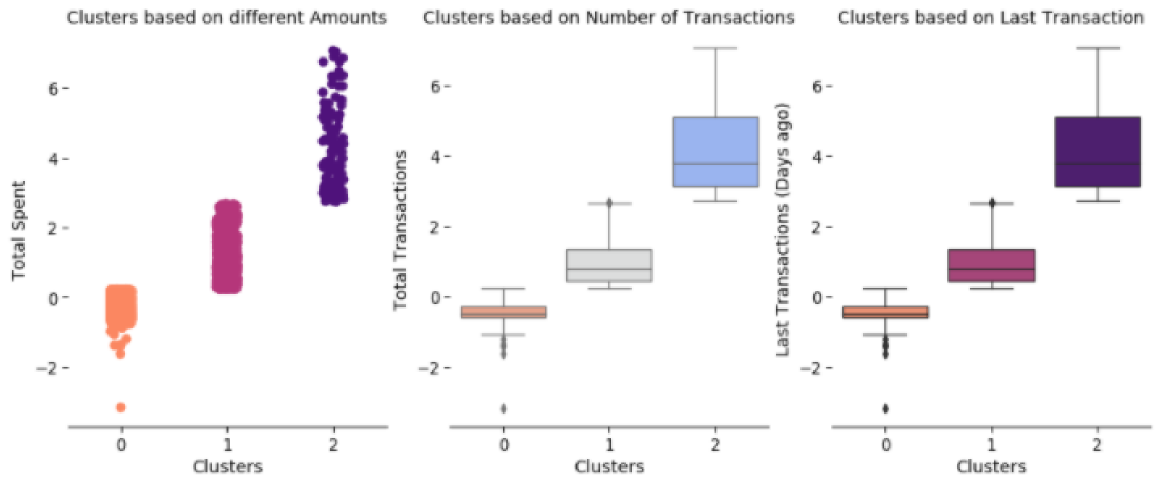
```
data_new['Cluster_Labels'] = cluster_labels
data_new.head()
```

	CustomerID	Amount	Frequency	Recency	Cluster_Id	Cluster_Labels
0	12346.0	-0.723738	-0.723738	-0.723738	2	0
1	12347.0	1.731617	1.731617	1.731617	0	1
2	12348.0	0.300128	0.300128	0.300128	0	1
3	12349.0	0.277517	0.277517	0.277517	0	1
4	12350.0	-0.533235	-0.533235	-0.533235	2	0

- Plotting to view the clusters, we can see that Hierarchical Clustering also does a good job of segmenting the customers.

```
fig, ax =plt.subplots(nrows= 1, ncols = 3, figsize= (16,6))
ty=sns.stripplot(x='Cluster_Labels', y='Amount', data=data_new, s=8, ax = ax[0], palette='magma_r')
sns.despine(left=True)
ty.set_title('Clusters based on different Amounts')
ty.set_ylabel('Total Spent')
ty.set_xlabel('Clusters')

tt=sns.boxplot(x='Cluster_Labels', y='Frequency', data=data_new, ax = ax[1], palette='coolwarm_r')
tt.set_title('Clusters based on Number of Transactions')
tt.set_ylabel('Total Transactions')
tt.set_xlabel('Clusters')

tr=sns.boxplot(x='Cluster_Labels', y='Recency', data=data_new, ax = ax[2], palette='magma_r')
tr.set_title('Clusters based on Last Transaction')
tr.set_ylabel('Last Transactions (Days ago)')
tr.set_xlabel('Clusters')
```

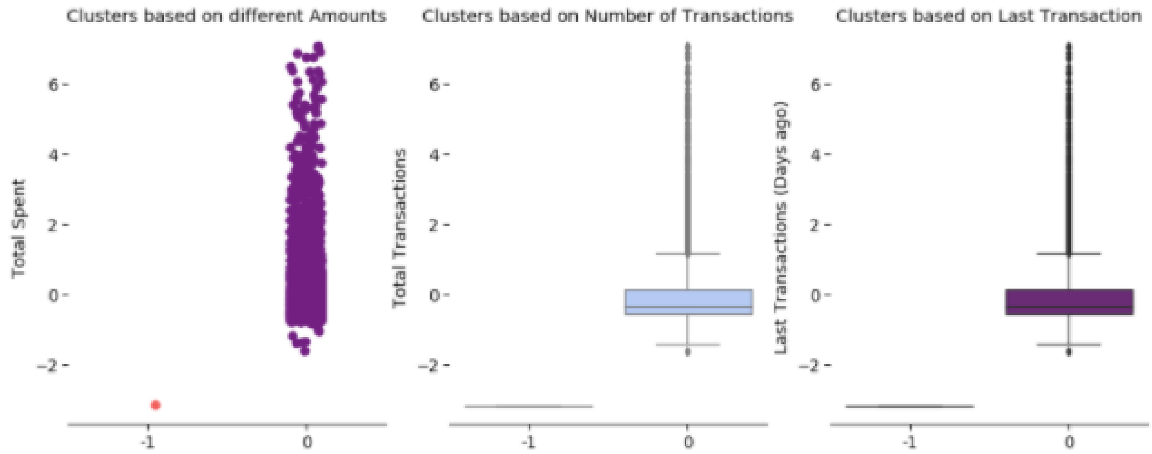
```
Text(0.5, 0, 'Clusters')
```



- DBSCAN

```
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=1, min_samples=3).fit(cluster_df)

data_new['dbscan'] = db.labels_
data_new.head()

fig, ax =plt.subplots(nrows= 1, ncols = 3, figsize= (16,6))
ty=sns.stripplot(x='dbscan', y='Amount', data=data_new, s=8, ax = ax[0], palette='magma_r')
sns.despine(left=True)
ty.set_title('Clusters based on different Amounts')
ty.set_ylabel('Total Spent')
ty.set_xlabel('Clusters')

tt=sns.boxplot(x='dbscan', y='Frequency', data=data_new, ax = ax[1], palette='coolwarm_r')
tt.set_title('Clusters based on Number of Transactions')
tt.set_ylabel('Total Transactions')
tt.set_xlabel('Clusters')

tr=sns.boxplot(x='dbscan', y='Recency', data=data_new, ax = ax[2], palette='magma_r')
tr.set_title('Clusters based on Last Transaction')
tr.set_ylabel('Last Transactions (Days ago)')
tr.set_xlabel('Clusters')
```

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



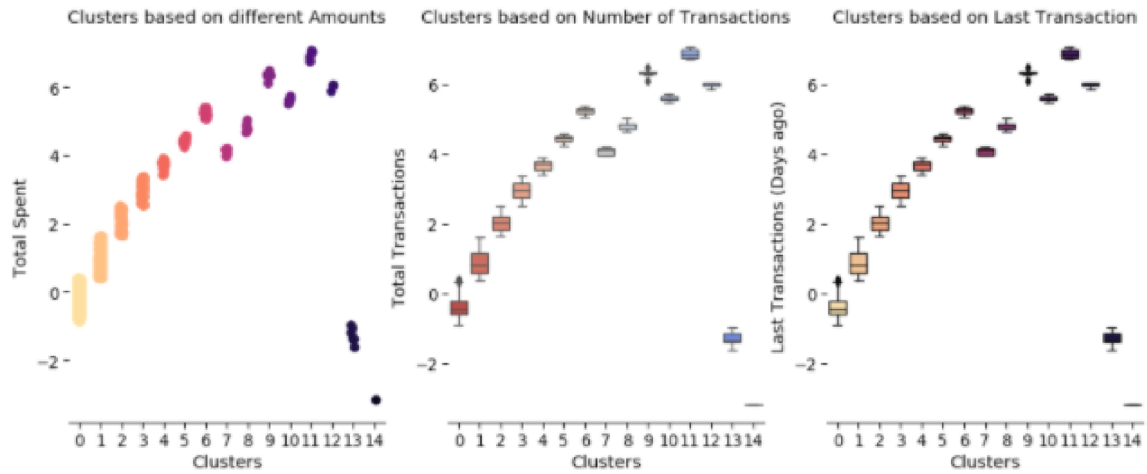
As you can see, even with an epsilon value of 1, DBSCAN is unable to categorize the data. My assumption is that the density of the cluster are very close together which isn't allowing form the right n_clu numbers.

- MeanShift

  Here we used estimate_bandwidth to find the right number of bandwidth which came up to 0.4306 and Meanshift returns 14 different clusters.

```
from sklearn.cluster import MeanShift, estimate_bandwidth

# The following bandwidth can be automatically detected using
bandwidth = estimate_bandwidth(cluster_df, quantile=0.1)
ms = MeanShift(bandwidth).fit(cluster_df)
data_new['meanshift'] = ms.labels_

fig, ax =plt.subplots(nrows= 1, ncols = 3, figsize= (16,6))
ty=sns.stripplot(x='meanshift', y='Amount', data=data_new, s=8, ax = ax[0], palette='magma_r')
sns.despine(left=True)
ty.set_title('Clusters based on different Amounts')
ty.set_ylabel('Total Spent')
ty.set_xlabel('Clusters')

tt=sns.boxplot(x='meanshift', y='Frequency', data=data_new, ax = ax[1], palette='coolwarm_r')
tt.set_title('Clusters based on Number of Transactions')
tt.set_ylabel('Total Transactions')
tt.set_xlabel('Clusters')

tr=sns.boxplot(x='meanshift', y='Recency', data=data_new, ax = ax[2], palette='magma_r')
tr.set_title('Clusters based on Last Transaction')
tr.set_ylabel('Last Transactions (Days ago)')
tr.set_xlabel('Clusters')
```

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



Conclusion and Next Steps:

- Though we had a sufficient number of records for building our model, the segmentation could have been better if we had those 25% records with missing Customer IDs.
- Both Kmeans and Hierarchical Clustering show that we can segment customers based on their transaction counts with the frequency of their purchase into three cluster, where Frequent buyers provide more value to the business and can be prioritized
- DBSCAN could have made some improvement if we try with a larger number of epsilon values.
- MeanShift will help in cases we have a very large customer base and would help targeting smaller batches. But our customer base is only about 4k and with this many segmentation, it is difficult to drive effective campaigns against them.
- With the above points in consideration, it is more efficient to choose KMeans over the remaining after accounting for the computation and the cost of running it.