Name: Punit Jain
Batch: June2021_05 (PIET DA)

Python Capstone - Task 2 _ E-Commerce Market Segmentation

"The term Market Segmentation refers to aggregating prospective buyers into groups or segments with common needs and who respond similarly to a marketing action. This enables companies to target different categories of consumers who perceive the full value of certain products and services differently from one another."

The Procedure is Explained as follows:

1. Dataset:

   First of all, there are some operations which are needed to be performed on the given Dataset.

   The dataset (.xlsx file) can be used as it is or it can converted to the ".csv" format and then, the data frame can be created.

Dataset Operations

```
! wget https://github.com/punitjain-jp/E-Com_Market-Segmentation/raw/main/E-commerce/Datasets/Online_Retail.xlsx
```

For Excel File

[If this is used, then go Directly to 'NaN Values.]

```
[ ]  df = pd.DataFrame(pd.read_excel("Online_Retail.xlsx"))
     df.dtypes
```

[OR]

.xlsx to .csv Conversion

```
[3]  r_file = pd.read_excel("Online_Retail.xlsx")

     r_file.to_csv("dataset.csv", index=None, header=True)
```

```
[ ]  df = pd.DataFrame(pd.read_csv("dataset.csv"))
```

2. Datatype Change:

   If we convert the dataset to .csv format then, the datatype of the “Invoice Date” column must be modified as per the requirement or the step can be simply omitted if the column is not required.

DataType Change (of Invoice Date Column)

```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
df.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  datetime64[ns]
 5   UnitPrice    541909 non-null  float64       
 6   CustomerID   406829 non-null  float64       
 7   Country      541909 non-null  object        
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

3. Cleaning the Data:

   It means the removal of ‘NaN’ (Not a Number) Values from the dataset.

```
[6] df.isna().sum()
```

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

```
[7] df.dropna(subset=['CustomerID'],inplace=True)      # Dropping the 'na' Values
```

```
df.head(3)
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom

4. Creating Amount Column:

   This column indicates the grouping of customers based on the total amount spent by them.

```
df['Total_Amount']=df['Quantity']*df['UnitPrice']
df.head(5)
```

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

5. Grouping the Data:

   This grouping of data is important so as to define various types of parameters which then further lead to the better segmentation.

   The Group 1:

   It determines the total number of Invoices as per unique Customer

   i.e. based on Customer ID.

```
[13] inv_frq = df.groupby('CustomerID')['InvoiceNo'].count()
     inv_frq = inv_frq.reset_index()
     inv_frq.head(3)
```

	CustomerID	InvoiceNo
0	12346.0	2
1	12347.0	182
2	12348.0	31

The Group 2:

It determines the Total Amount spent by a Customer.

```
consumer_amt = df.groupby('CustomerID')['Total_Amount'].sum()
consumer_amt = consumer_amt.reset_index()
consumer_amt.head(3)
```

	CustomerID	Total_Amount
0	12346.0	0.00
1	12347.0	4310.00
2	12348.0	1797.24

The Group 3:

It determines the Recency of the Customer i.e. how many days before the Customer was last active.

```
[15] last_date = max(df['InvoiceDate'])
     last_date
```

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

```
[16] df['Recency'] = last_date - df['InvoiceDate']
```

```
ttl_day = df.groupby('CustomerID')['Recency'].min()
ttl_day = ttl_day.reset_index()
ttl_day['Recency'] = ttl_day['Recency'].dt.days
ttl_day.head(3)
```

	CustomerID	Recency
0	12346.0	325
1	12347.0	1
2	12348.0	74

6. Final Dataframe:

In this step, the data frames generated in the process of grouping are combined to form the final data frame which can then be used further.

```
trans_details = pd.merge(inv_frq,consumer_amt,on='CustomerID')
trans_details = pd.merge(trans_details,ttl_day,on='CustomerID')
trans_details
```

	CustomerID	InvoiceNo	Total_Amount	Recency
0	12346.0	2	0.00	325
1	12347.0	182	4310.00	1
2	12348.0	31	1797.24	74
3	12349.0	73	1757.55	18
4	12350.0	17	334.40	309
...	...	...	...	...
4367	18280.0	10	180.60	277
4368	18281.0	7	80.82	180
4369	18282.0	13	176.60	7
4370	18283.0	756	2094.88	3
4371	18287.0	70	1837.28	42

4372 rows × 4 columns

7. Statistics:

Statistics is the science concerned with developing and studying methods for collecting, analyzing, interpreting and presenting empirical data.

In this, we use the Inter Quartile Range for the removal of outliers present in the dataset.

[Note] : This is very useful as these outliers affect the statistical measures like mean, median etc. in a very bad manner.

We need to find the inter quartile range separately for all the three groups and then, modify the dataset by removing the outliers using the obtained inter-quartile range.

For Recency

```
[19] a = trans_details.Recency.quantile(0.05)
     b = trans_details.Recency.quantile(0.95)
     iqr1 = b-a   # Inter Quartile Range
     trans_details = trans_details[(trans_details['Recency']>=a-1.5*iqr1) & (trans_details['Recency']<=b+1.5*iqr1)]
```

For Total Amount

```
[20] c = trans_details.Total_Amount.quantile(0.05)
     d = trans_details.Total_Amount.quantile(0.95)
     iqr2 = d-c   # Inter Quartile Range
     trans_details = trans_details[(trans_details['Total_Amount']>=c-1.5*iqr2) & (trans_details['Total_Amount']<=d+1.5*iqr2)]
```

For Invoices

```
e = trans_details.InvoiceNo.quantile(0.05)
f = trans_details.InvoiceNo.quantile(0.95)
iqr3 = f-e   # Inter Quartile Range
trans_details = trans_details[(trans_details['Total_Amount']>=e-1.5*iqr3) & (trans_details['Total_Amount']<=f+1.5*iqr3)]
```

8. Scaling:

Feature scaling is essential for machine learning algorithms that calculate distances between data. Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions do not work correctly without normalization.

'Standard Scaler' is that it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1.

In case of multivariate data, this is done feature-wise (in other words independently for each column of the data).

The ultimate goal to perform standardization is to bring down all the features to a common scale without distorting the differences in the range of the values.

The 'fit_transform' is used on the training data so that we can scale the training data and also learn the scaling parameters of that data.

```
scale = StandardScaler().fit_transform(
    trans_details[['InvoiceNo', 'Total_Amount', 'Recency']]
    )
scaling = pd.DataFrame(scale)
scaling.columns = ['InvoiceNo', 'Total_Amount', 'Recency']
scaling
```

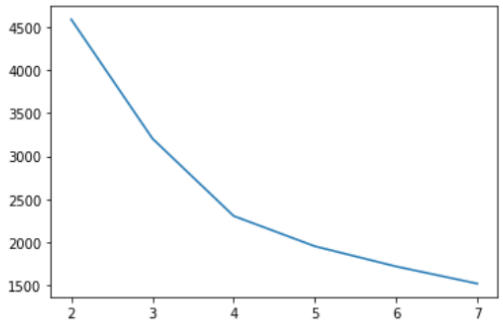
	InvoiceNo	Total_Amount	Recency
0	-0.900845	-1.729304	1.766422
1	-0.404976	-0.007580	1.622783
2	-0.834729	-1.271070	0.671173
3	-0.537208	0.636007	0.760948
4	-0.636382	-0.751567	1.416302

9. Elbow Graph:

In cluster analysis, the elbow method is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use.

The same method can be used to choose the number of parameters in other data-driven models, such as the number of principal components to describe a data set.

```
plt.plot(range(2,8,1),value)
plt.xticks(ticks=range(2,8))
plt.show()
```



10. Silhouette Scores:

Silhouette Coefficient or silhouette score is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to 1.

1: Means clusters are well apart from each other and clearly distinguished.

0: Means clusters are indifferent, or we can say that the distance between clusters is not significant.

-1: Means clusters are assigned in the wrong way.

Silhouette Score = (b-a)/max(a,b)

where

a= average intra-cluster distance i.e the average distance between each point within a cluster.

b= average inter-cluster distance i.e the average distance between all clusters.

```
for i in range(2,8):
  kmeans = KMeans(n_clusters=i,max_iter=40).fit(scaling)
  cluster_labels=kmeans.labels_
  score_avg = silhouette_score(scaling,cluster_labels)
  print('For n_Cluster{}, the Silhouette Score is {}'.format(i,score_avg))
```

```
For n_Cluster2, the Silhouette Score is 0.34164951594999515
For n_Cluster3, the Silhouette Score is 0.35655749784360646
For n_Cluster4, the Silhouette Score is 0.3805769567752563
For n_Cluster5, the Silhouette Score is 0.36155170203203507
For n_Cluster6, the Silhouette Score is 0.3651982231357887
For n_Cluster7, the Silhouette Score is 0.31653292314326026
```

11. Model Building:

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.

“The objective of K-means is simple: group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset.”

A cluster refers to a collection of data points aggregated together because of certain similarities.

You'll define a target number k, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster. Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares.

In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

The 'means' in the K-means refers to averaging of the data; that is, finding the centroid.

```
kmeans = KMeans(n_clusters=4,max_iter=50).fit(scaling)
clusters = kmeans.labels_

trans_details['Clusters'] = clusters
trans_details
```

	CustomerID	InvoiceNo	Total_Amount	Recency	Clusters
0	12346.0	2	0.00	325	0
4	12350.0	17	334.40	309	0
6	12353.0	4	89.00	203	0
8	12355.0	13	459.40	213	0
14	12361.0	10	189.90	286	0
...	...	...	...	...	...
4365	18277.0	9	97.63	57	3
4366	18278.0	9	173.90	73	3
4367	18280.0	10	180.60	277	0
4368	18281.0	7	80.82	180	0
4369	18282.0	13	176.60	7	3

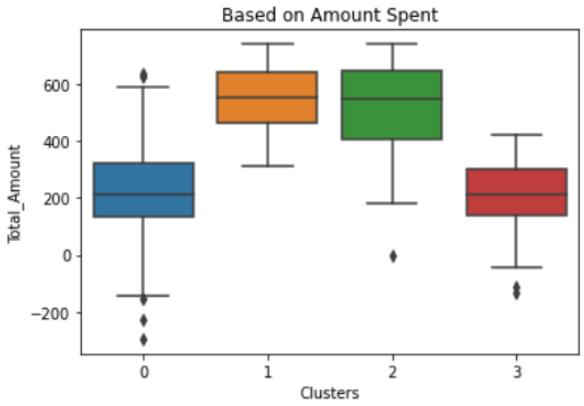
2371 rows × 5 columns

12. Visualizations:

A box and whisker plot—also called a box plot—displays the five-number summary of a set of data. The five-number summary is the minimum, first quartile, median, third quartile, and maximum. In a box plot, we draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median.

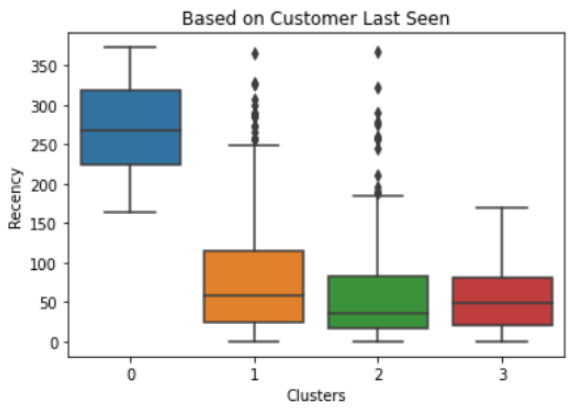
i) The plot shows the Total Amount spent by the customers which are present in a Cluster.

```
sab.boxplot(x='Clusters', y='Total_Amount', data=trans_details).set(title='Based on Amount Spent')
plt.show()
```



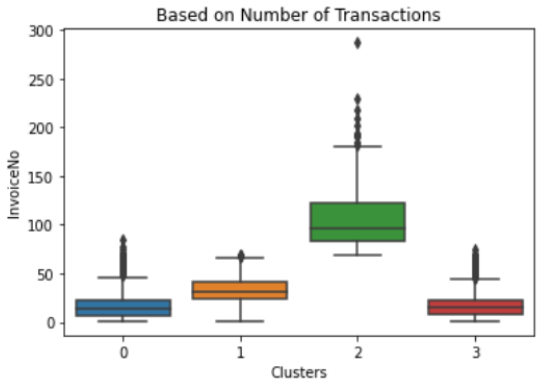
ii) The plot shows the Recency of the Customers as per various Clusters formed.

```
sab.boxplot(x='Clusters', y='Recency', data=trans_details).set(title='Based on Customer Last Seen')
plt.show()
```



iii) The plot shows the Total Number of Transactions Done by the customers respective to their Cluster.

```
sab.boxplot(x='Clusters', y='InvoiceNo', data=trans_details).set(title='Based on Number of Transactions')
plt.show()
```



13. Profiling:

The pandas_profiling library in Python include a method named as Profile Report () which generate a basic report on the input Data Frame.

The report consists of the Data Frame overview, sample, each attribute on which Data Frame is defined & correlations between them.

```
[33] ! pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
```

```
[34] from pandas_profiling import ProfileReport
```

[If you don't want to see the embedded profiling, then close the output and download the file to view it in your browser.]

> The download option is available at the end.

```
[45] report = ProfileReport(trans_details,title='Profile Report', html= {'style':{'full_width':True}})
     report.to_notebook_iframe()
```

Download the Profiling File

```
report.to_file(output_file='E_Comm_Profiling.html')
from google.colab import files
files.download('/content/E_Comm_Profiling.html')
```

[Please refer to the “Python Code File” & the “Profiling File (.html)” for more details.]