Q.Calculate the Customer Life Time Value (CLTV) Using 2 Different methods

1. RFM Method
2. Predictive Modelling

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
df = pd.read_csv('customer_purchases.csv')
```

In [3]:

```
df
```

Out[3]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
...	...	...	...	...	...	...	...	...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/2011 12:50	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	12/9/2011 12:50	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/2011 12:50	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/2011 12:50	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/2011 12:50	4.95	12680.0	France

541909 rows × 8 columns

In [4]:

```
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  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]:
```
df.isnull().sum()
```

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

RFM METHOD

- Recency
- Frequency
- Monetary

In [6]:
```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate']).dt.date
```

In [7]:
```
df = df.dropna(subset=['CustomerID']) #dropping null values
```

In [8]:
```
df['Total_Sales'] = df['Quantity'] * df['UnitPrice'] # calculating total sales
```

```
C:\Users\91775\AppData\Local\Temp\ipykernel_800\160927535.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df['Total_Sales'] = df['Quantity'] * df['UnitPrice'] # calculating total sales
```

CALCULATING RECENCY

In [9]:
```
recency_df = df.groupby(by='CustomerID', as_index=False)['InvoiceDate'].max()
recency_df.columns = ['CustomerID','LastPurshaceDate']
```

In [10]:
```
recent_date=recency_df.LastPurshaceDate.max()
print(recent_date)
```

```
2011-12-09
```

In [11]:
```
recency_df['Recency'] = recency_df['LastPurshaceDate'].apply(lambda x: (recent_date - x).
recency_df.head()
```

Out[11]:

	CustomerID	LastPurshaceDate	Recency
0	12346.0	2011-01-18	325
1	12347.0	2011-12-07	2
2	12348.0	2011-09-25	75
3	12349.0	2011-11-21	18
4	12350.0	2011-02-02	310

CALCULATING FREQUENCY

In [12]:

```python
df1= df
df1.drop_duplicates(subset=['InvoiceNo', 'CustomerID'], keep="first", inplace=True)
frequency_df = df1.groupby(by=['CustomerID'], as_index=False)['InvoiceNo'].count()
frequency_df.columns = ['CustomerID','Frequency']
frequency_df.head()
```

```
C:\Users\91775\AppData\Local\Temp\ipykernel_800\956064957.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df1.drop_duplicates(subset=['InvoiceNo', 'CustomerID'], keep="first", inplace=True)
```

Out[12]:

	CustomerID	Frequency
0	12346.0	2
1	12347.0	7
2	12348.0	4
3	12349.0	1
4	12350.0	1

CALCULATING MONETARY

In [13]:

```python
monetary_df = df.groupby(by='CustomerID',as_index=False)['Total_Sales'].sum()
monetary_df.columns = ['CustomerID','Monetary']
monetary_df.head()
```

Out[13]:

	CustomerID	Monetary
0	12346.0	0.00
1	12347.0	163.16
2	12348.0	331.36
3	12349.0	15.00
4	12350.0	25.20

MERGING ALL THE TABLES

In [14]:
```
temp_df = recency_df.merge(frequency_df,on='CustomerID')
rfm_df = temp_df.merge(monetary_df,on='CustomerID')
rfm_df
```

Out[14]:

	CustomerID	LastPurshaceDate	Recency	Frequency	Monetary
0	12346.0	2011-01-18	325	2	0.00
1	12347.0	2011-12-07	2	7	163.16
2	12348.0	2011-09-25	75	4	331.36
3	12349.0	2011-11-21	18	1	15.00
4	12350.0	2011-02-02	310	1	25.20
...	...	...	...	...	...
4367	18280.0	2011-03-07	277	1	23.70
4368	18281.0	2011-06-12	180	1	5.04
4369	18282.0	2011-12-02	7	3	36.80
4370	18283.0	2011-12-06	3	16	66.75
4371	18287.0	2011-10-28	42	3	80.40

4372 rows × 5 columns

RANKING

In [15]:
```
# (lower is better)
rfm_df['RecencyRank'] = (rfm_df['Recency'].rank(ascending=True) - 1) / (len(rfm_df) - 1)

# (higher is better)
rfm_df['FrequencyRank'] = (rfm_df['Frequency'].rank(ascending=False) - 1) / (len(rfm_df)

# (higher is better)
rfm_df['MonetaryRank'] = (rfm_df['Monetary'].rank(ascending=False) - 1) / (len(rfm_df) -
```

In [16]:
```
rfm_df
```

Out[16]:

	CustomerID	LastPurshaceDate	Recency	Frequency	Monetary	RecencyRank	FrequencyRank	MonetaryRank
0	12346.0	2011-01-18	325	2	0.00	96.156486	60.626859	95.950583
1	12347.0	2011-12-07	2	7	163.16	4.598490	18.576985	18.142302
2	12348.0	2011-09-25	75	4	331.36	62.445665	35.758408	9.059712
3	12349.0	2011-11-21	18	1	15.00	27.327843	84.991993	75.623427
4	12350.0	2011-02-02	310	1	25.20	94.577900	84.991993	61.816518
...	...	...	...	...	...	...	...	...
4367	18280.0	2011-03-07	277	1	23.70	91.672386	84.991993	62.971860
4368	18281.0	2011-06-12	180	1	5.04	80.256234	84.991993	89.533288
4369	18282.0	2011-12-02	7	3	36.80	12.422787	45.676047	51.681537
4370	18283.0	2011-12-06	3	16	66.75	6.977808	5.159003	36.879433
4371	18287.0	2011-10-28	42	3	80.40	46.282315	45.676047	31.892016

4372 rows × 8 columns

NOW CALCULATING SCORES:

AS THE WEIGTHS ARE NOT POVIDED I WILL USE :

`0.15*RecencyRank + 0.28*FrequencyRank + 0.57*MonetaryRank`

In [17]:
```
rfm_df['RFM_Score']= 0.15*rfm_df['RecencyRank'] + 0.28*rfm_df['FrequencyRank'] + 0.57*rfm_
```

In [18]:
```
rfm_df=rfm_df.round(0)
rfm_df.head()
```

Out[18]:

	CustomerID	LastPurshaceDate	Recency	Frequency	Monetary	RecencyRank	FrequencyRank	MonetaryRank	RF
0	12346.0	2011-01-18	325	2	0.0	96.0	61.0	96.0	
1	12347.0	2011-12-07	2	7	163.0	5.0	19.0	18.0	
2	12348.0	2011-09-25	75	4	331.0	62.0	36.0	9.0	
3	12349.0	2011-11-21	18	1	15.0	27.0	85.0	76.0	
4	12350.0	2011-02-02	310	1	25.0	95.0	85.0	62.0	

PERFORMING SEGMENTATION:

- 0 - 50 - Low valued customer
- 50 - 75 - Medium valued customer
- 76 - 100 - High valued customer

In [19]:
```
rfm_df["Customer_segment"]=np.where(rfm_df['RFM_Score'] > 75 ,"High Value Customer",(np.wh
```

In [34]:
```python
final1 =rfm_df[['CustomerID',"RFM_Score",'Customer_segment']]
final1
```

Out[34]:

	CustomerID	RFM_Score	Customer_segment
0	12346.0	86.0	High Value Customer
1	12347.0	16.0	Low value Customer
2	12348.0	25.0	Low value Customer
3	12349.0	71.0	Medium Value Customer
4	12350.0	73.0	Medium Value Customer
...	...	...	...
4367	18280.0	73.0	Medium Value Customer
4368	18281.0	87.0	High Value Customer
4369	18282.0	44.0	Low value Customer
4370	18283.0	24.0	Low value Customer
4371	18287.0	38.0	Low value Customer

4372 rows × 3 columns

Predictive Modelling

In [21]:
```python
pip install lifetimes
```

...

In [22]:
```python
from lifetimes import BetaGeoFitter, GammaGammaFitter
from lifetimes.utils import summary_data_from_transaction_data
from sklearn.preprocessing import MinMaxScaler

```

In [23]:
```python
pm = df[['CustomerID','Total_Sales','InvoiceDate']]
```

In [24]:
```python
lf_tx_data = summary_data_from_transaction_data(pm, 'CustomerID', 'InvoiceDate', monetary_
lf_tx_data.reset_index().head()
```

Out[24]:

	CustomerID	frequency	recency	T	monetary_value
0	12346.0	0.0	0.0	325.0	0.000000
1	12347.0	6.0	365.0	367.0	22.993333
2	12348.0	3.0	283.0	358.0	97.253333
3	12349.0	0.0	0.0	18.0	0.000000
4	12350.0	0.0	0.0	310.0	0.000000

In [25]:
```python
bgf = BetaGeoFitter(penalizer_coef=0.0)
bgf.fit(lf_tx_data['frequency'], lf_tx_data['recency'], lf_tx_data['T'])
```

Out[25]:
```
<lifetimes.BetaGeoFitter: fitted with 4372 subjects, a: 0.02, alpha: 55.62, b: 0.49, r: 0.84>
```

In [26]:

```
t = 10
lf_tx_data['pred_num_txn'] = round(bgf.conditional_expected_number_of_purchases_up_to_time
lf_tx_data.sort_values(by='pred_num_txn', ascending=False).head(10).reset_index()
```

Out[26]:

	CustomerID	frequency	recency	T	monetary_value	pred_num_txn
0	14911.0	145.0	372.0	373.0	45.236483	3.40
1	12748.0	114.0	373.0	373.0	5.457982	2.68
2	17841.0	112.0	372.0	373.0	9.544821	2.63
3	15311.0	90.0	373.0	373.0	43.113222	2.12
4	14606.0	88.0	372.0	373.0	10.123523	2.07
5	13089.0	82.0	367.0	369.0	44.713902	1.95
6	12971.0	71.0	369.0	372.0	51.797042	1.68
7	16422.0	66.0	352.0	369.0	61.474394	1.57
8	14527.0	63.0	371.0	373.0	-0.784444	1.49
9	13408.0	54.0	372.0	373.0	76.712407	1.28

In [27]:

```
shortlisted_customers = lf_tx_data[lf_tx_data['frequency']>0]
shortlisted_customers = shortlisted_customers[shortlisted_customers['monetary_value'] > 0
shortlisted_customers.head().reset_index()
```

Out[27]:

	CustomerID	frequency	recency	T	monetary_value	pred_num_txn
0	12347.0	6.0	365.0	367.0	22.993333	0.16
1	12348.0	3.0	283.0	358.0	97.253333	0.09
2	12356.0	2.0	303.0	325.0	25.950000	0.07
3	12358.0	1.0	149.0	150.0	142.800000	0.09
4	12359.0	5.0	324.0	331.0	1.380000	0.15

In [28]:

```
ggf = GammaGammaFitter(penalizer_coef = 0)
ggf.fit(shortlisted_customers['frequency'],
        shortlisted_customers['monetary_value'])
```

Out[28]: <lifetimes.GammaGammaFitter: fitted with 2681 subjects, p: 0.95, q: 1.59, v: 25.06>

In [29]:

```
## PREDICTING average transaction value

lf_tx_data['pred_txn_value'] = round(ggf.conditional_expected_average_profit(
        lf_tx_data['frequency'],
        lf_tx_data['monetary_value']), 2)
lf_tx_data.reset_index().head()
```

Out[29]:

	CustomerID	frequency	recency	T	monetary_value	pred_num_txn	pred_txn_value
0	12346.0	0.0	0.0	325.0	0.000000	0.02	40.35
1	12347.0	6.0	365.0	367.0	22.993333	0.16	24.62
2	12348.0	3.0	283.0	358.0	97.253333	0.09	87.49
3	12349.0	0.0	0.0	18.0	0.000000	0.11	40.35
4	12350.0	0.0	0.0	310.0	0.000000	0.02	40.35

In [30]:
```
lf_tx_data['CLV'] = round(ggf.customer_lifetime_value(
    bgf,
    lf_tx_data['frequency'],
    lf_tx_data['recency'],
    lf_tx_data['T'],
    lf_tx_data['monetary_value'],
    time=12,
    discount_rate=0.01
), 2)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\arraylike.py:402: RuntimeWarning: inva
lid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [53]:
```
final2= lf_tx_data.drop(lf_tx_data.iloc[:, 0:6], axis=1)

```

In [55]:
```

final2 = final2.sort_values(by='CLV', ascending=False).head(10).reset_index()

```

In [58]:
```
min_v = final2['CLV'].min()
max_v = final2['CLV'].max()
```

In [60]:
```
final2['CLV'] =((final2['CLV'] - min_v) / (max_v- min_v)) * 100
```

In [61]:
```
final2
```

Out[61]:

	CustomerID	CLV
0	18102.0	100.000000
1	17949.0	53.080220
2	14646.0	34.859091
3	17450.0	34.842560
4	16333.0	23.401344
5	16013.0	17.204289
6	13868.0	10.586562
7	12901.0	2.375153
8	12798.0	2.208514
9	15769.0	0.000000

RESULT OF BOTH RFM AND Predective modelling

In [62]:
```
final1.merge(final2,on='CustomerID')
```

Out[62]:

	CustomerID	RFM_Score	Customer_segment	CLV
0	12798.0	16.0	Low value Customer	2.208514
1	12901.0	3.0	Low value Customer	2.375153
2	13868.0	19.0	Low value Customer	10.586562
3	14646.0	0.0	Low value Customer	34.859091
4	15769.0	2.0	Low value Customer	0.000000
5	16013.0	1.0	Low value Customer	17.204289
6	16333.0	3.0	Low value Customer	23.401344
7	17450.0	2.0	Low value Customer	34.842560
8	17949.0	0.0	Low value Customer	53.080220
9	18102.0	0.0	Low value Customer	100.000000

In [ ]:
```

```