

```
from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
data = pd.read_csv("/content/drive/MyDrive/Online_Retail.csv", encoding="unicode_escape", parse_dates=['InvoiceDate'])
print("Total number of transactions happened in the given period: "+ str(data.shape[0]))
features = ['CustomerID', 'InvoiceNo', 'InvoiceDate', 'Quantity', 'UnitPrice']
data_clv = data[features]
data_clv['TotalSales'] = data_clv['Quantity'].multiply(data_clv['UnitPrice'])
print(data_clv.shape)
print(data_clv.head())
```

Total number of transactions happened in the given period: 541909
(541909, 6)

	CustomerID	InvoiceNo	InvoiceDate	Quantity	UnitPrice	TotalSales
0	17850.0	536365	2010-12-01 08:26:00	6	2.55	15.30
1	17850.0	536365	2010-12-01 08:26:00	6	3.39	20.34
2	17850.0	536365	2010-12-01 08:26:00	8	2.75	22.00
3	17850.0	536365	2010-12-01 08:26:00	6	3.39	20.34
4	17850.0	536365	2010-12-01 08:26:00	6	3.39	20.34

data_clv.head

pandas.core.generic.NDFrame.head
def head(n: int=5) -> NDFrameT

</usr/local/lib/python3.10/dist-packages/pandas/core/generic.py>

Return the first `n` rows.

This function returns the first `n` rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

```
# Printing the details of the dataset
maxdate = data_clv['InvoiceDate'].dt.date.max()
mindate = data_clv['InvoiceDate'].dt.date.min()
unique_cust = data_clv['CustomerID'].nunique()
tot_quantity = data_clv['Quantity'].sum()
tot_sales = data_clv['TotalSales'].sum()
```

```
tot_sales = data_clv['TotalSales'].sum()
```

```
print(f"The Time range of transactions is: {mindate} to {maxdate}")
print(f"Total number of unique customers: {unique_cust}")
print(f"Total Quantity Sold: {tot_quantity}")
print(f"Total Sales for the period: {tot_sales}")
```

➞ The Time range of transactions is: 2010-12-01 to 2011-12-09
 Total number of unique customers: 4372
 Total Quantity Sold: 5176450
 Total Sales for the period: 9747747.933999998

```
# Transforming the data to customer level for the analysis
customer = data_clv.groupby('CustomerID').agg({'InvoiceDate': lambda x: (x.max() - x.min()).days,
                                              'InvoiceNo': lambda x: len(x),
                                              'TotalSales': lambda x: sum(x)})

customer.columns = ['Age', 'Frequency', 'TotalSales']
customer.head()
```

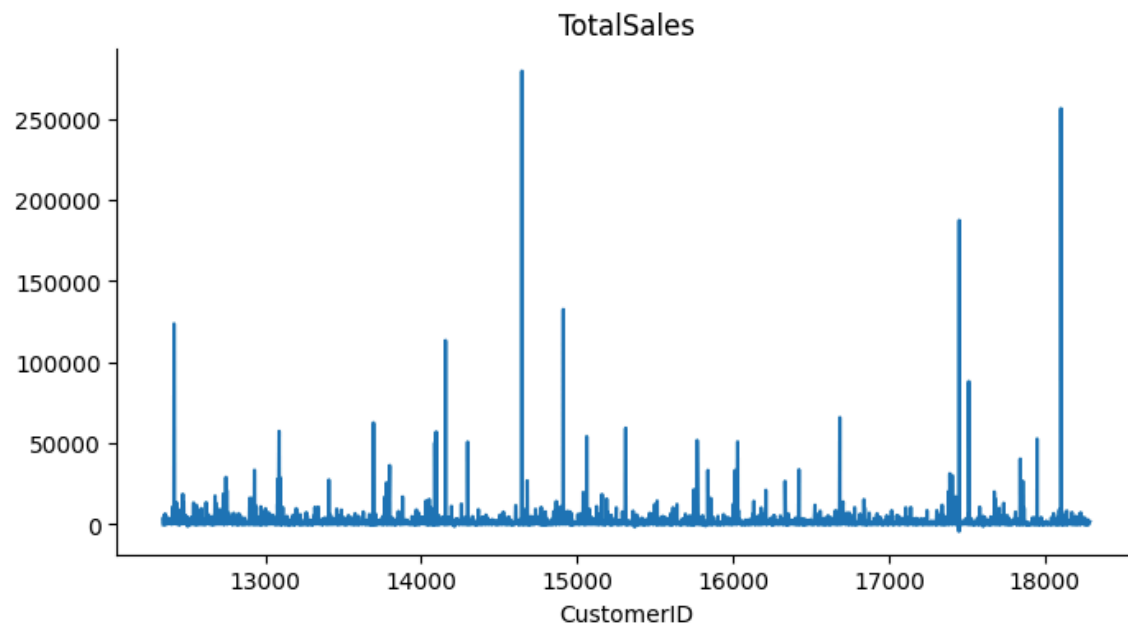
	Age	Frequency	TotalSales
CustomerID			
12346.0	0	2	0.00
12347.0	365	182	4310.00
12348.0	282	31	1797.24
12349.0	0	73	1757.55
12350.0	0	17	334.40

Next steps: [View recommended plots](#)

▼ TotalSales

```
# @title TotalSales
```

```
from matplotlib import pyplot as plt
customer['TotalSales'].plot(kind='line', figsize=(8, 4), title='TotalSales')
plt.gca().spines[['top', 'right']].set_visible(False)
```



```
# Calculating the necessary variables for CLV calculation
```

```
Average_sales = round(np.mean(customer['TotalSales']),2)
```

```
print(f"Average sales: ${Average_sales}")
```

```
Purchase_freq = round(np.mean(customer['Frequency']), 2)
```

```
print(f"Purchase Frequency: {Purchase_freq}")
```

```
Retention_rate = customer[customer['Frequency']>1].shape[0]/customer.shape[0]
```

```
churn = round(1 - Retention_rate, 2)
```

```
print(f"Churn: {churn}%")
```

```
Average sales: $1898.46
```

```
Purchase Frequency: 93.05
```

```
Churn: 0.02%
```

```
# Calculating the CLV
```

```
Profit_margin = 0.05
```

```
CLV = round(((Average_sales * Purchase_freq/churn)) * Profit_margin, 2)
```

```
print(f"The Customer Lifetime Value (CLV) for each customer is: ${CLV}")
```

The Customer Lifetime Value (CLV) for each customer is: \$441629.26

```
customer = data_clv.groupby('CustomerID').agg({  
    'InvoiceDate': lambda x: x.min().month,  
    'InvoiceNo': lambda x: len(x),  
    'TotalSales': lambda x: np.sum(x)  
})
```

```
customer.columns = ['Start_Month', 'Frequency', 'TotalSales']  
customer.head()
```

	Start_Month	Frequency	TotalSales
CustomerID			
12346.0	1	2	0.00
12347.0	12	182	4310.00
12348.0	12	31	1797.24
12349.0	11	73	1757.55
12350.0	2	17	334.40

Next steps:

 [View recommended plots](#)

```
# Calculating CLV for each cohort
months = ['Jan', 'Feb', 'March', 'Apr', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
Monthly_CLV = []

for i in range(1, 13):
    customer_m = customer[customer['Start_Month']==i]

    Average_sales = round(np.mean(customer_m['TotalSales']),2)

    Purchase_freq = round(np.mean(customer_m['Frequency']), 2)

    Retention_rate = customer_m[customer_m['Frequency']>1].shape[0]/customer_m.shape[0]
    churn = round(1 - Retention_rate, 2)

    CLV = round(((Average_sales * Purchase_freq/churn)) * Profit_margin, 2)

    Monthly_CLV.append(CLV)
```

```
pip install lifetimes
```

Collecting lifetimes

Downloading Lifetimes-0.11.3-py3-none-any.whl (584 kB)

584.2/584.2 kB 10.7 MB/s eta 0:00:00

Requirement already satisfied: numpy>=1.10.0 in /usr/local/lib/python3.10/dist-packages (from lifetimes) (1.25.2)

Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from lifetimes) (1.11.4)

Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.10/dist-packages (from lifetimes) (2.0.3)

Requirement already satisfied: autograd>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from lifetimes) (1.6.2)

Collecting dill>=0.2.6 (from lifetimes)

Downloading dill-0.3.8-py3-none-any.whl (116 kB)

116.3/116.3 kB 14.4 MB/s eta 0:00:00

Requirement already satisfied: future>=0.15.2 in /usr/local/lib/python3.10/dist-packages (from autograd>=1.2.0->lifetimes) (0.18.3)

Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.0->lifetimes) (2.8.2)

Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.0->lifetimes) (2023.4)

Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.0->lifetimes) (2024.1)



Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=0.24.0->lifetimes) (1.16.0)

Installing collected packages: dill, lifetimes

Successfully installed dill-0.3.8 lifetimes-0.11.3

```
import lifetimes
```

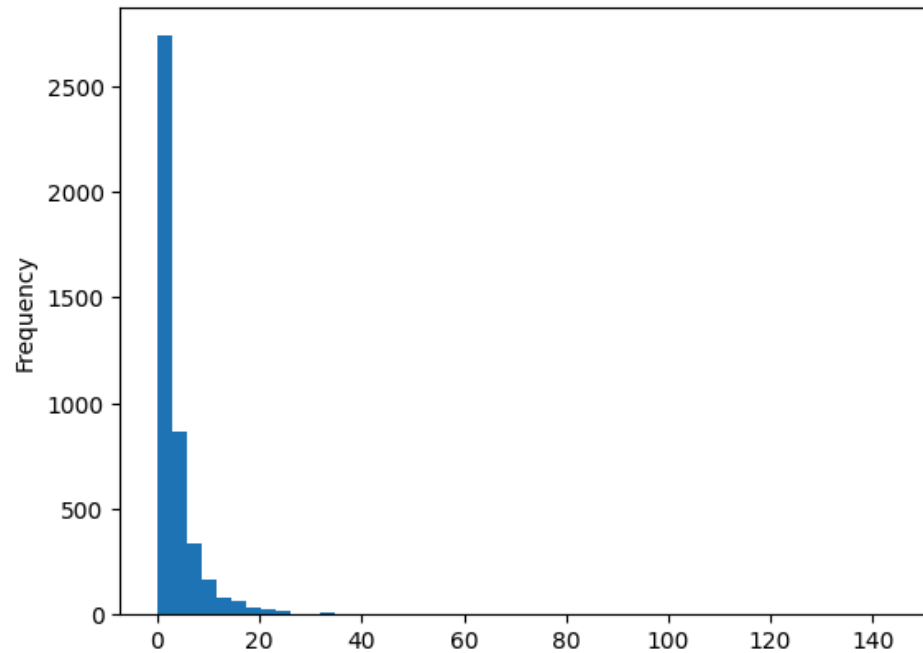
```
summary = lifetimes.utils.summary_data_from_transaction_data(data_clv, 'CustomerID', 'InvoiceDate', 'TotalSales')
summary = summary.reset_index()
summary.head()
```

	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	599.701667	
2	12348.0	3.0	283.0	358.0	301.480000	
3	12349.0	0.0	0.0	18.0	0.000000	
4	12350.0	0.0	0.0	310.0	0.000000	

Next steps: [View recommended plots](#)

```
summary['frequency'].plot(kind='hist', bins=50)
print(summary['frequency'].describe())
one_time_buyers = round(sum(summary['frequency'] == 1)/float(len(summary))*(100),2)
print("Percentage of customers purchase the item only once:", one_time_buyers,"%")
```



```
count    4372.000000
mean      3.413541
std       6.674343
min       0.000000
25%       0.000000
50%       1.000000
75%       4.000000
max      145.000000
Name: frequency, dtype: float64
Percentage of customers purchase the item only once: 19.62 %
```





```
bgf = lifetimes.BetaGeoFitter(penalizer_coef=0.0)
bgf.fit(summary['frequency'], summary['recency'], summary['T'])
print(bgf)
```

```
# Model summary
bgf.summary
```

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

	coef	se(coef)	lower 95% bound	upper 95% bound	
r	0.843025	0.026206	0.791661	0.894389	
alpha	55.619383	2.088118	51.526671	59.712095	
a	0.021519	0.006381	0.009012	0.034026	
b	0.488673	0.176970	0.141812	0.835534	

```
summary['probability_alive'] = bgf.conditional_probability_alive(summary['frequency'], summary['recency'], summary['T'])
summary.head(10)
```

	CustomerID	frequency	recency	T	monetary_value	probability_alive	
0	12346.0	0.0	0.0	325.0	0.000000	1.000000	
1	12347.0	6.0	365.0	367.0	599.701667	0.995966	
2	12348.0	3.0	283.0	358.0	301.480000	0.981687	
3	12349.0	0.0	0.0	18.0	0.000000	1.000000	
4	12350.0	0.0	0.0	310.0	0.000000	1.000000	
5	12352.0	6.0	260.0	296.0	208.151667	0.991857	
6	12353.0	0.0	0.0	204.0	0.000000	1.000000	
7	12354.0	0.0	0.0	232.0	0.000000	1.000000	
8	12355.0	0.0	0.0	214.0	0.000000	1.000000	
9	12356.0	2.0	303.0	325.0	269.905000	0.983167	

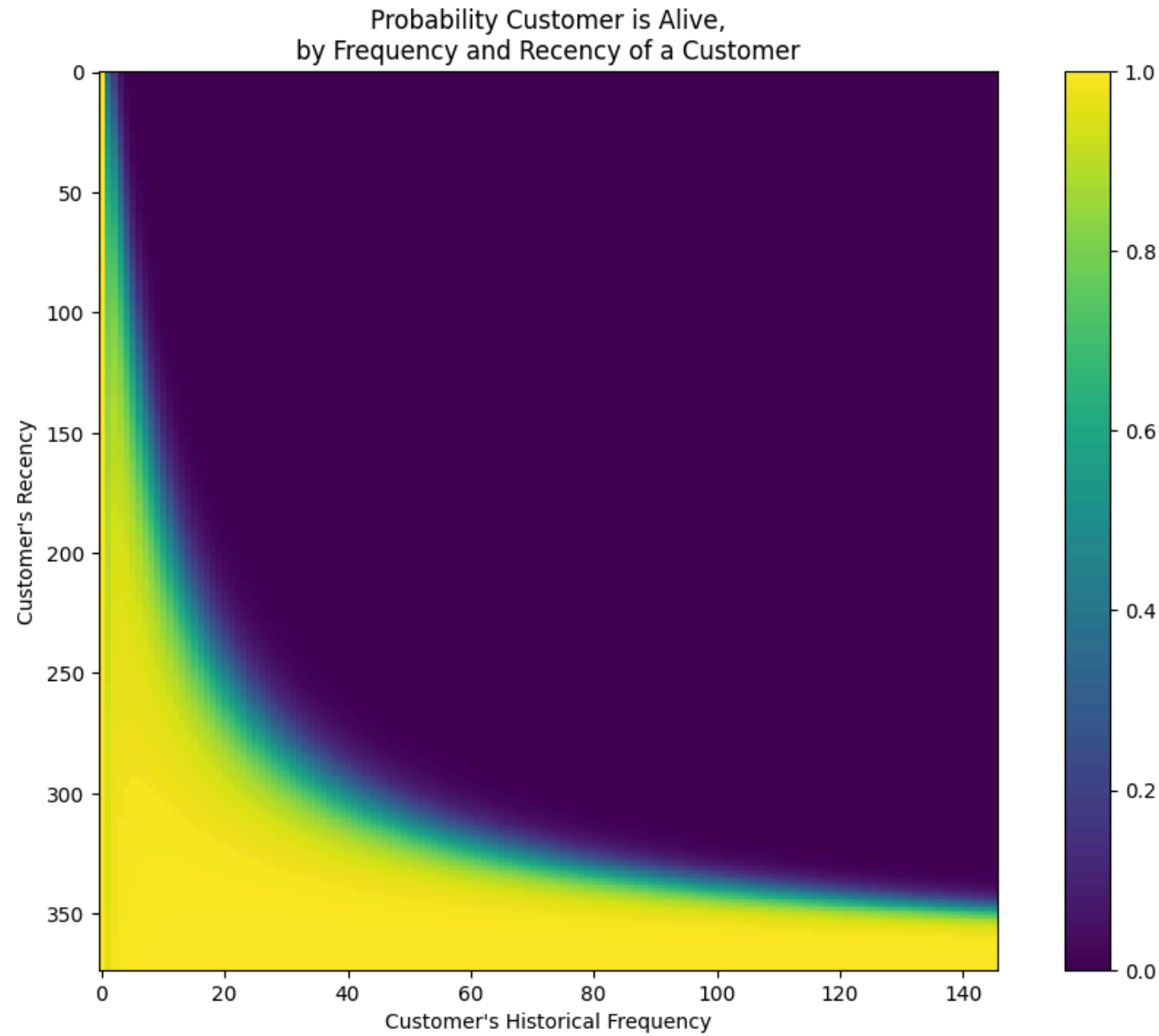
Next steps:  [View recommended plots](#)

```
# Visual representation of relationship between recency and frequency
from lifetimes.plotting import plot_probability_alive_matrix
```

```
fig = plt.figure(figsize=(12,8))
plot_probability_alive_matrix(bgf)
```



```
<Axes: title={'center': 'Probability Customer is Alive,\nby Frequency and Recency of a Customer'}, xlabel="Customer's Historical Frequency", ylabel="Customer's Recency">
```





```
# Predict future transaction for the next 30 days based on historical data
```

```
t = 30
```

```
summary['pred_num_txn'] = round(bgf.conditional_expected_number_of_purchases_up_to_time(t, summary['frequency'], summary['recency'], summary['T']), 0)
```

```
summary.sort_values(by='pred_num_txn', ascending=False).head(10).reset_index()
```



	index	CustomerID	frequency	recency	T	monetary_value	probability_alive	pred_num_txn	
0	1895	14911.0	145.0	372.0	373.0	910.463724	0.999791	10.0	
1	4042	17841.0	112.0	372.0	373.0	355.421429	0.999749	8.0	
2	330	12748.0	114.0	373.0	373.0	254.975000	0.999810	8.0	
3	2192	15311.0	90.0	373.0	373.0	655.266778	0.999760	6.0	
4	568	13089.0	82.0	367.0	369.0	690.871707	0.999610	6.0	
5	1674	14606.0	88.0	372.0	373.0	130.858295	0.999697	6.0	
6	487	12971.0	71.0	369.0	372.0	153.311831	0.999494	5.0	
7	3014	16422.0	66.0	352.0	369.0	486.931667	0.994981	5.0	
8	803	13408.0	54.0	372.0	373.0	490.050556	0.999543	4.0	
9	2099	15189.0	45.0	331.0	332.0	341.859111	0.999456	4.0	

```
return_customers_summary = summary[summary['frequency']>0]
```

```
print(return_customers_summary.shape)
```



```
return_customers_summary.head()
```

```
(2991, 7)
```

	CustomerID	frequency	recency	T	monetary_value	probability_alive	pred_num_txn	
1	12347.0	6.0	365.0	367.0	599.701667	0.995966	0.0	
2	12348.0	3.0	283.0	358.0	301.480000	0.981687	0.0	
5	12352.0	6.0	260.0	296.0	208.151667	0.991857	1.0	
9	12356.0	2.0	303.0	325.0	269.905000	0.983167	0.0	
11	12358.0	1.0	149.0	150.0	683.200000	0.957457	0.0	

Next steps: [View recommended plots](#)

```
return_customers_summary[['frequency', 'monetary_value']].corr()
```



	frequency	monetary_value	
frequency	1.000000	0.198027	
monetary_value	0.198027	1.000000	

```
# Modeling the monetary value using Gamma-Gamma Model
```

```
positive_monetary_value = return_customers_summary[return_customers_summary['monetary_value'] > 0]
from lifetimes import GammaGammaFitter
```

```
bgf = GammaGammaFitter(penalizer_coef=0.01)
bgf.fit(positive_monetary_value['frequency'], positive_monetary_value['monetary_value'])
```

```
# Summary of the fitted parameters
bgf.summary
```

	coef	se(coef)	lower 95% bound	upper 95% bound	
p	3.792282	0.096405	3.603328	3.981236	
q	0.357821	0.007701	0.342727	0.372915	
v	3.689081	0.097430	3.498119	3.880043	

```
# Calculating the conditional expected average profit for each customer per transaction
```

```
summary['exp_avg_sales'] = bgf.conditional_expected_average_profit(
    summary['frequency'],
    summary['monetary_value']
)
summary.head()
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.