VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF ECONOMICS AND LAW

FINAL REPORT

PREDICT THE CUSTOMER LIFETIME VALUE USING MACHINE LEARNING

Lecturer: Nguyễn Anh Phong – Phan Huy Tâm

Student: Trần Thanh Phúc

Student ID: K194141740

Subject: Machine Learning

Ho Chi Minh City, June 23th. 2022

Abstract:

The concept of viewing customers as resources that must be managed and valued is now widely accepted and accepted by academics and professionals. This focus on client relationship management necessitates an understanding of Customer Lifetime Value (CLV), as CLV models are a productive and viable approach to assessing an organization's relationship with its clients. CLV evaluation is especially important for businesses that provide client-focused services. I used the CLV dataset created by Prediction Consultants, a data science firm based in Rishon, Lezion. This dataset was created using transactions that occurred between September 2020 and September 2021. I chose the regression approach, which states that CLV can be calculated by creating a machine learning model for existing customers. The idea is that we can build a regression model using recent six-month data as independent variables and total revenue over three years as a dependent variable.

1. INTRODUCTION

What is Customer Lifetime Value?

Customer lifetime value (CLV or customer LTV) is the predicted total of all future revenues (or profits) generated by a specific customer for a business. Using accurate CLV estimates as the foundation for marketing decisions will increase the company's revenue (or profits). The catch is that calculating accurate CLV predictions is extremely difficult.

The Benefits of Using Customer Lifetime Value

It is easy to stay in the present moment, but this is not always the best way to realize the full potential value of each customer. Looking at conversion rates and first purchases, for example, while ignoring long-term customer value, may lead marketers to invest resources in acquiring "cheap" customers with low total revenue value, rather than paying more to acquire customers who will provide a consistent stream of income for years to come. Similarly, marketers and retention experts should devote resources to nurturing customer relationships with those customers who will continue to generate significant revenue in the long run, while saving resources that would otherwise be wasted on low-value customers.

When a company has a reliable method for predicting CLV, it can better manage customer relationships, maximize the effectiveness of marketing and retention actions, optimize the resources invested in retaining each customer, and achieve more accurate customer metrics.

In summary, the goal of using CLV is to focus on finding, nurturing, and retaining customers who add the most value to the company over time.

The Challenge of Calculating Customer Lifetime Value

Calculating customer lifetime value (CLV) is difficult because it requires accurate estimates of future events. It is difficult to predict parameters like how long a customer will stay with a company and how much the customer will spend in each time period, especially when the customer is new. The fact that the data required to perform the calculations may be hidden deep within multiple databases adds to the difficulty.

Some simplistic approaches attempt to categorize customers into logical groups, such as all customers who came from a specific source, live in a specific location, or purchased a specific product/service. More sophisticated approaches, such as linear regression or Bayesian probabilistic models, attempt to predict the future using well-known statistical techniques. All of these, however, have large (or huge!) margins of error and thus present a very risky basis for important marketing decisions.

Estimating the Customer Lifetime Value

In general, several methods for estimating the CLV have been proposed. The first approach is based on empirical data analysis and putting it into an analytical formula. Another technique computes CLV by adding customer profit in a given cycle and then averaging it annually, semi-annually, or monthly. Another approach was to ask how recently, how frequently, and how much a customer purchased. These methods are problematic because they cannot assist in the development of a predictive model for new customers.

Use Machine Learning to forecast Customer Lifetime Value

Machine Learning (ML) is the study of algorithms and statistics combined. This method is used by computer engines to perform a specific task without the need for additional instructions, relying on patterns found in large data sets. This data analysis method is a subset of artificial intelligence (or AI, if short). It implies that systems can learn and make decisions while human intervention is minimized. It can significantly simplify your life when combined with predictive analytics. This is significant because when measuring CLV, you must deal with a large number of numbers.

I chose the regression approach, which states that CLV can be calculated by creating a machine learning model for existing customers. The idea is that we can build a regression model using recent six-month data as independent variables and total revenue over three years as a dependent variable.

The Customer Lifetime Value model

CLV is usually defined and estimated at the customer or segment level. This enables us to distinguish between customers who are more profitable than others rather than simply looking at average profitability. The problem is predicting future profits when the timing and benefit of future

transactions are unknown, as Mulhern and Bell et al. discuss. Gupta and other scholars propose that a customer's CLV is:

$$\mathrm{CLV} = \sum_{t=0}^{T} \frac{(P_t - C_t) r_t}{(1+i)^t} - AC$$

Gupta and other scholars propose that a customer's CLV is:

where:

= the price paid by a customer at time t

= the direct cost of providing service to the customer at time t

= the firm's discount rate or cost of capital

= likelihood of customer repeat purchase or being 'alive' at time t

AC stands for acquisition cost.

T denotes the time horizon for calculating CLV.

2. METHODOLOGY

2.1. The dataset

To predict the CLV of a company's customers, I used the CLV dataset created by Prediction Consultants, a data science firm based in Rishon, Lezion. This dataset was created using transactions that occurred between September 2020 and September 2021. It has many learning characteristics, and the dataset was attached to this report or can be download here.

2.2. Data Understanding

First, I will import the libraries that I think I will use in this task.

```
# Import Libraries
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

Then, I will import the dataset.

```
# Import Data
df = pd.read_csv('company.csv', encoding='unicode_escape' )
```

OK, let's take a quick look at the dataset.

```
print(df.shape)
print(list(df.columns))
```

```
(495478, 7)
['CustomerID', 'InvoiceNo', 'InvoiceDate', 'StockCode', 'Description', 'Quantity', 'UnitPrice']
```

It is easy to see that the data has 7 variables and 495478 observations.

Next, I will look at the top 5 and bottom observations of the data. It is useful for quickly verifying data. For example, after sorting or appending rows.

```
df.head()
```

	CustomerID	InvoiceNo	InvoiceDate	StockCode	Description	Quantity	UnitPrice
0	17850.0	536365	25/09/2020	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	8.415
1	17850.0	536365	25/09/2020	71053	WHITE METAL LANTERN	6	11.187
2	17850.0	536365	25/09/2020	84406B	CREAM CUPID HEARTS COAT HANGER	8	9.075
3	17850.0	536365	25/09/2020	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	11.187
4	17850.0	536365	25/09/2020	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	11.187

```
df.tail()
```

	CustomerID	InvoiceNo	InvoiceDate	StockCode	Description	Quantity	UnitPrice
495473	15804.0	581585	03/10/2021	22466	FAIRY TALE COTTAGE NIGHT LIGHT	12	6.435
495474	13113.0	581586	04/10/2021	22061	LARGE CAKE STAND HANGING STRAWBERY	8	9.735
495475	13113.0	581586	04/10/2021	23275	SET OF 3 HANGING OWLS OLLIE BEAK	24	4.125
495476	13113.0	581586	04/10/2021	21217	RED RETROSPOT ROUND CAKE TINS	24	29.535
495477	13113.0	581586	04/10/2021	20685	DOORMAT RED RETROSPOT	10	23.364

Let's prints information about our dataset including the index dtype and columns, non-null values and memory usage.

```
df.info()
```

```
RangeIndex: 495478 entries, 0 to 495477
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   CustomerID   361878 non-null  float64
 1   InvoiceNo    495478 non-null  object
 2   InvoiceDate  495478 non-null  object
 3   StockCode    495478 non-null  object
 4   Description  494024 non-null  object
 5   Quantity     495478 non-null  int64
 6   UnitPrice    495478 non-null  float64
dtypes: float64(2), int64(1), object(4)
```

We see that there are 3 numerical variables and 4 categorical variables. I added them to 2 separate variables because they can be useful later.

```
numerical_features = ['CustomerID', 'Quantity', 'UnitPrice']
categorical_features = [x for x in df.columns if (x not in numerical_features)]
print(numerical_features)
print(categorical_features)
```

✓ 0.1s

```
['CustomerID', 'Quantity', 'UnitPrice']
['InvoiceNo', 'InvoiceDate', 'StockCode', 'Description']
```

Let's generate descriptive statistics, which summarize the central tendency, dispersion, and shape of a dataset's distribution while excluding null values.

```
# Describe Data
df.describe()
```

✓ 0.2s

	CustomerID	Quantity	UnitPrice
count	361878.000000	495478.000000	495478.000000
mean	15547.871368	8.605486	14.956993
std	1594.402590	227.588756	327.740946
min	12346.000000	-80995.000000	-36504.798000
25%	14194.000000	1.000000	4.125000
50%	15514.000000	3.000000	6.930000
75%	16931.000000	10.000000	13.629000
max	18287.000000	80995.000000	128601.000000

Let's visualize 3 numerical variables with graphs to make it easier to understand.

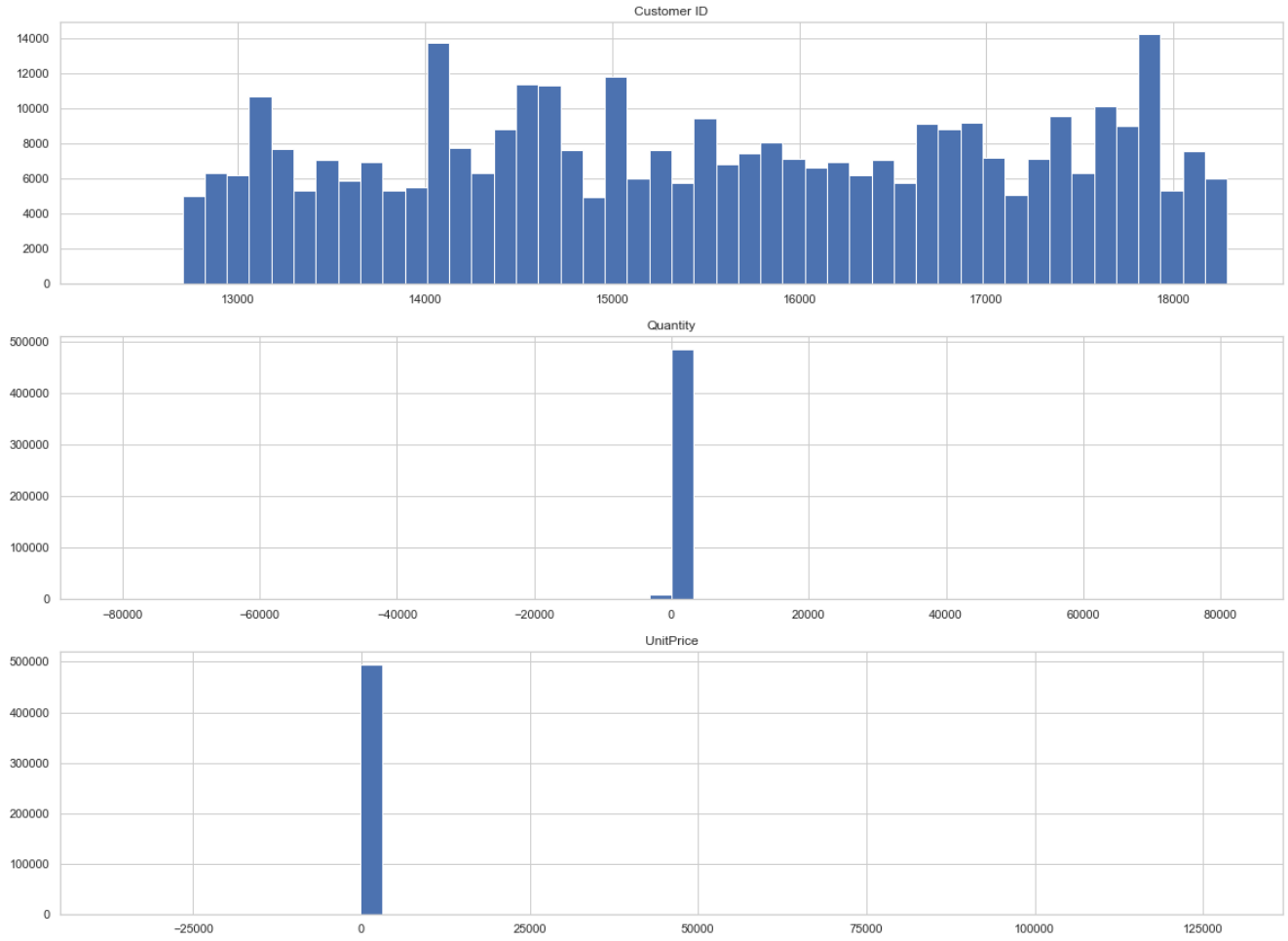
```
plt.figure(figsize=(20,15))
ax1 = plt.subplot(3,1,1)
ax1 = df['CustomerID'].hist(bins=50)
ax1.set_title('Customer ID')

ax2 = plt.subplot(3,1,2)
ax2 = df['Quantity'].hist(bins=50)
ax2.set_title('Quantity')

ax3 = plt.subplot(3,1,3)
ax3 = df['UnitPrice'].hist(bins=50)
ax3.set_title('UnitPrice')

plt.savefig('attribute_histogram_plots')
plt.show()
```

✓ 1.5s



There are some negative values in the "Quantity" and "UnitPrice" features, as seen in both the statistical summary and the histogram above. Because it makes no sense for quantity to be negative, I will remove these values in the next phase.

Let’s check the number of missing values in each column.

```
# Check minssing values
df.isna().sum()
```

✓ 0.1s

```
CustomerID     133600
InvoiceNo           0
InvoiceDate         0
StockCode           0
Description      1454
Quantity            0
UnitPrice           0
dtype: int64
```

These missing values cannot be filled using methods because they will be wrong in nature and meaning. Therefore, I will remove them in later section.

Let's compute pairwise correlation of columns, excluding null values.

```
print(df.corr())  # view correlation
plt.figure(figsize=(7,5))
sns.heatmap(df.corr(),annot=True)
plt.savefig('corr_ori')
plt.show()
```

✓ 0.5s

```
             CustomerID  Quantity  UnitPrice
CustomerID     1.000000 -0.000844  -0.000828
Quantity      -0.000844  1.000000  -0.000994
UnitPrice     -0.000828 -0.000994   1.000000
```

It is easy to see that the variables are not correlated at all.

2.3. Data Preparation

Let's make a copy dataset for this section.

```
df2 = df.copy()
df2.info()
```

✓ 0.2s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 495478 entries, 0 to 495477
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   CustomerID   361878 non-null  float64
 1   InvoiceNo    495478 non-null  object 
 2   InvoiceDate  495478 non-null  object 
 3   StockCode    495478 non-null  object 
 4   Description  494024 non-null  object 
 5   Quantity     495478 non-null  int64  
 6   UnitPrice    495478 non-null  float64
dtypes: float64(2), int64(1), object(4)
```

I will remove missing values for the new dataset as I said before.

```
# Drop missing values and check
df2 = df2.dropna()
print(df2.isna().sum())
df2.info()
```

✓ 0.4s

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 361878 entries, 0 to 495477
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   CustomerID   361878 non-null  float64
 1   InvoiceNo    361878 non-null  object 
 2   InvoiceDate  361878 non-null  object 
 3   StockCode    361878 non-null  object 
 4   Description  361878 non-null  object 
 5   Quantity     361878 non-null  int64  
 6   UnitPrice    361878 non-null  float64
dtypes: float64(2), int64(1), object(4)
```

Our database has shrunk by nearly 27%, from 495,478 to 361,878 records. It makes no difference because we still have a few hundred thousand records.

Next, let's remove any negative values and zero value in Quantity and UnitPrice columns.

```
# Replace negative values with NaN
df2['Quantity'][df2['Quantity'] <= 0] = np.nan
df2['UnitPrice'][df2['UnitPrice'] <= 0] = np.nan
# Drop NaN
df2 = df2.dropna()
df2.head()
```

✓ 0.1s

	CustomerID	InvoiceNo	InvoiceDate	StockCode	Description	Quantity	UnitPrice
0	17850.0	536365	25/09/2020	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6.0	8.415
1	17850.0	536365	25/09/2020	71053	WHITE METAL LANTERN	6.0	11.187
2	17850.0	536365	25/09/2020	84406B	CREAM CUPID HEARTS COAT HANGER	8.0	9.075
3	17850.0	536365	25/09/2020	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6.0	11.187
4	17850.0	536365	25/09/2020	84029E	RED WOOLLY HOTTIE WHITE HEART.	6.0	11.187

Print out the summary information and statistics to check if the goal has been achieved.

```
print(df2.info())
print(df2.describe())
```

✓ 0.2s

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 354321 entries, 0 to 495477
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   CustomerID   354321 non-null  float64
 1   InvoiceNo    354321 non-null  object 
 2   InvoiceDate  354321 non-null  object 
 3   StockCode    354321 non-null  object 
 4   Description  354321 non-null  object 
 5   Quantity     354321 non-null  float64
 6   UnitPrice    354321 non-null  float64
dtypes: float64(3), object(4)
```

	CustomerID	Quantity	UnitPrice
count	354321.000000	354321.000000	354321.000000
mean	15552.486392	12.013795	9.781179
std	1594.527150	189.267956	58.946762
min	12346.000000	1.000000	0.003300
25%	14194.000000	2.000000	4.125000
50%	15522.000000	4.000000	6.435000
75%	16931.000000	12.000000	12.375000
max	18287.000000	80995.000000	26871.075000

After removing nulls, there is not any negative values and my new dataset has 354,321 records and 7 columns.

Several features, such as "StockCode" and "Description," are unnecessary. As a result, I'm going to drop them.

```
# Delete 2 columns
del df2['StockCode']
del df2['Description']
df2.head()
```

✓ 0.9s

	CustomerID	InvoiceNo	InvoiceDate	Quantity	UnitPrice
0	17850.0	536365	25/09/2020	6.0	8.415
1	17850.0	536365	25/09/2020	6.0	11.187
2	17850.0	536365	25/09/2020	8.0	9.075
3	17850.0	536365	25/09/2020	6.0	11.187
4	17850.0	536365	25/09/2020	6.0	11.187

I want to give new names to the columns of the data to make it easier to use.

```
# Rename columns
df2 = df2.rename(columns={'CustomerID': 'ID_OF_CUSTOMER', 'InvoiceNo': 'NUMBER_OF_INVOICE', \
    'InvoiceDate': 'DATE_OF_INVOICE', 'Quantity': 'QUANTITY', 'UnitPrice': 'PRICE_OF_UNIT'})
df2.head()
```

✓ 0.1s

	ID_OF_CUSTOMER	NUMBER_OF_INVOICE	DATE_OF_INVOICE	QUANTITY	PRICE_OF_UNIT
0	17850.0	536365	25/09/2020	6.0	8.415
1	17850.0	536365	25/09/2020	6.0	11.187
2	17850.0	536365	25/09/2020	8.0	9.075
3	17850.0	536365	25/09/2020	6.0	11.187
4	17850.0	536365	25/09/2020	6.0	11.187

Let's create visualizations for features.

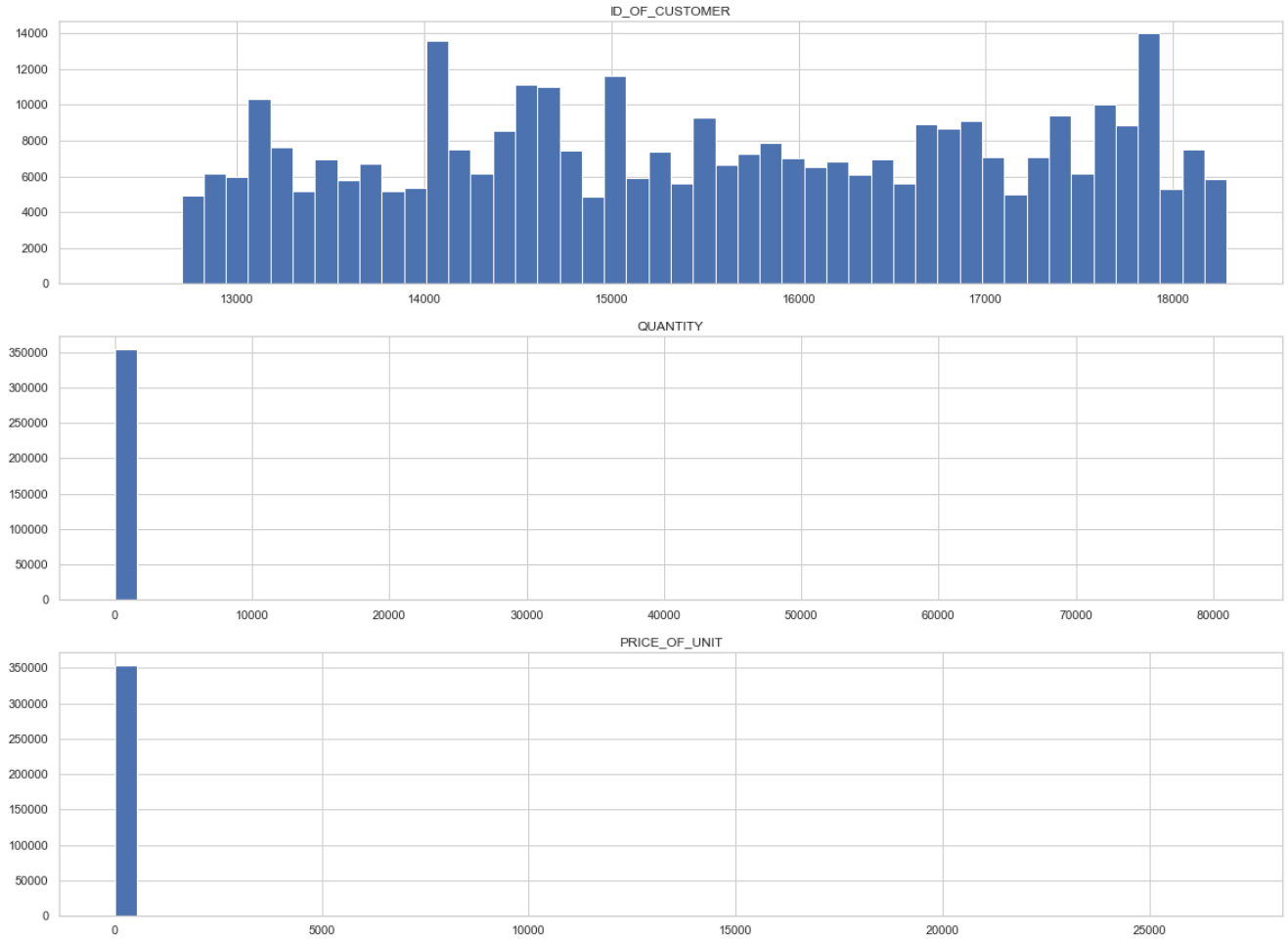
```
plt.figure(figsize=(20,15))
ax1 = plt.subplot(3,1,1)
ax1 = df2['ID_OF_CUSTOMER'].hist(bins=50)
ax1.set_title('ID_OF_CUSTOMER')

ax2 = plt.subplot(3,1,2)
ax2 = df2['QUANTITY'].hist(bins=50)
ax2.set_title('QUANTITY')

ax3 = plt.subplot(3,1,3)
ax3 = df2['PRICE_OF_UNIT'].hist(bins=50)
ax3.set_title('PRICE_OF_UNIT')

plt.savefig('attribute_histogram_plots_2')
plt.show()
```

✓ 1.2s



Let's add a new feature that represents the total cost of the purchase.

```
df2['TOTAL_PURCHASE'] = df2['QUANTITY'] * df2['PRICE_OF_UNIT']
df2.head()
```

✓ 0.7s

	ID_OF_CUSTOMER	NUMBER_OF_INVOICE	DATE_OF_INVOICE	QUANTITY	PRICE_OF_UNIT	TOTAL_PURCHASE
0	17850.0	536365	25/09/2020	6.0	8.415	50.490
1	17850.0	536365	25/09/2020	6.0	11.187	67.122
2	17850.0	536365	25/09/2020	8.0	9.075	72.600
3	17850.0	536365	25/09/2020	6.0	11.187	67.122
4	17850.0	536365	25/09/2020	6.0	11.187	67.122

Convert 'DATE OF INVOICE' column to real date data type.

```
df2['DATE_OF_INVOICE'] = pd.to_datetime(df2['DATE_OF_INVOICE'])
print(df2.info())
df2.head()
```

✓ 0.2s

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 354321 entries, 0 to 495477
Data columns (total 6 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   ID_OF_CUSTOMER     354321 non-null  float64
 1   NUMBER_OF_INVOICE  354321 non-null  object
 2   DATE_OF_INVOICE    354321 non-null  datetime64[ns]
 3   QUANTITY           354321 non-null  float64
 4   PRICE_OF_UNIT      354321 non-null  float64
 5   TOTAL_PURCHASE     354321 non-null  float64
dtypes: datetime64[ns](1), float64(4), object(1)
```

	ID_OF_CUSTOMER	NUMBER_OF_INVOICE	DATE_OF_INVOICE	QUANTITY	PRICE_OF_UNIT	TOTAL_PURCHASE
0	17850.0	536365	2020-09-25	6.0	8.415	50.490
1	17850.0	536365	2020-09-25	6.0	11.187	67.122
2	17850.0	536365	2020-09-25	8.0	9.075	72.600
3	17850.0	536365	2020-09-25	6.0	11.187	67.122
4	17850.0	536365	2020-09-25	6.0	11.187	67.122

Now let's extract the month and year from the “DATE OF INVOICE” column.

```
# Extract month and year from DATE_OF_INVOICE.
df2['MONTH_BY_YEAR'] = df2['DATE_OF_INVOICE'].apply(lambda x: x.strftime('%b-%Y'))
df2.head()
```

✓ 3.3s

	ID_OF_CUSTOMER	NUMBER_OF_INVOICE	DATE_OF_INVOICE	QUANTITY	PRICE_OF_UNIT	TOTAL_PURCHASE	MONTH_BY_YEAR
0	17850.0	536365	2020-09-25	6.0	8.415	50.490	Sep-2020
1	17850.0	536365	2020-09-25	6.0	11.187	67.122	Sep-2020
2	17850.0	536365	2020-09-25	8.0	9.075	72.600	Sep-2020
3	17850.0	536365	2020-09-25	6.0	11.187	67.122	Sep-2020
4	17850.0	536365	2020-09-25	6.0	11.187	67.122	Sep-2020

Let's make a pivot table that takes the columns as input and groups the entries into a two-dimensional table in such a way that the data is summarized in multiple dimensions.

```
# Create a pivot table with index is ID_OF_CUSTOMER, column is MONTH_BY_YEAR and values is TOTAL_PURCHASE
SALES = df2.pivot_table(index=['ID_OF_CUSTOMER'], columns=['MONTH_BY_YEAR'], \
    values='TOTAL_PURCHASE', aggfunc='sum', fill_value=0).reset_index()
SALES.head()
```

✓ 0.2s

MONTH_BY_YEAR	ID_OF_CUSTOMER	Apr-2020	Apr-2021	Aug-2020	Aug-2021	Dec-2020	Dec-2021	Feb-2020	Feb-2021	Jan-2020	...
0	12346.0	0.000	0.000	0.000	0.000	254705.880	0.000	0.000	0.000	0.000	...
1	12747.0	0.000	1241.790	0.000	0.000	1025.574	1032.009	0.000	2908.818	0.000	...
2	12748.0	2729.067	3541.263	9.405	6900.795	2576.013	4693.788	1012.572	3990.558	270.996	...
3	12749.0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...
4	12820.0	0.000	0.000	0.000	1134.408	0.000	0.000	0.000	0.000	0.000	...

...	Mar-2020	Mar-2021	May-2020	May-2021	Nov-2020	Nov-2021	Oct-2020	Oct-2021	Sep-2020	Sep-2021
...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
...	0.000	1083.555	0.000	0.000	1000.032	0.000	0.000	0.000	1183.248	0.000
...	1549.086	6474.072	2060.157	6448.761	113.751	3299.670	2232.186	7421.436	4433.649	22251.801
...	0.000	0.000	0.000	9092.259	0.000	1889.547	0.000	0.000	0.000	2518.098
...	0.000	0.000	0.000	0.000	562.518	0.000	0.000	0.000	0.000	0.000

Let's add all of the month's sales to a new column.

```
# Create CLV column
SALES['CLV']=SALES.iloc[:,2:].sum(axis=1)
SALES.head()
```

✓ 0.9s

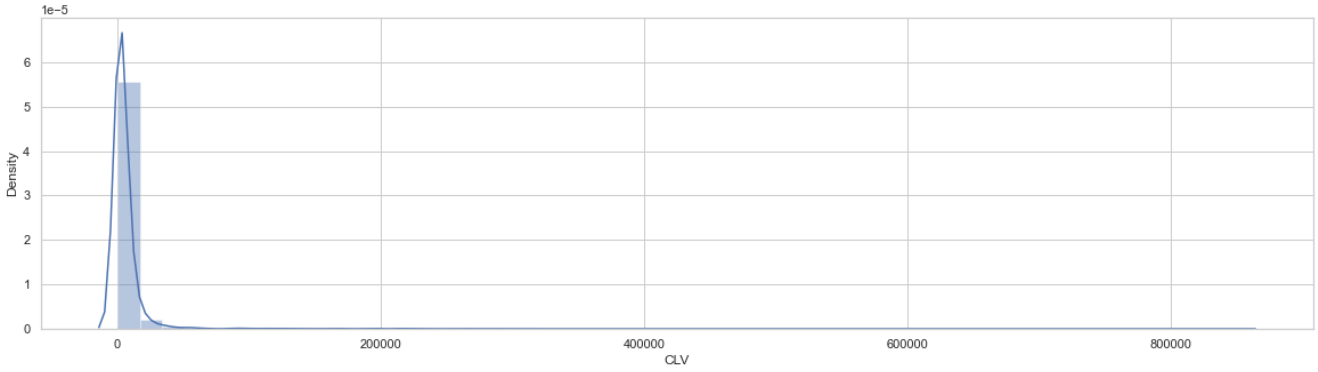
...	Mar-2021	May-2020	May-2021	Nov-2020	Nov-2021	Oct-2020	Oct-2021	Sep-2020	Sep-2021	CLV
...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	254705.880
...	1083.555	0.000	0.000	1000.032	0.000	0.000	0.000	1183.248	0.000	13846.833
...	6474.072	2060.157	6448.761	113.751	3299.670	2232.186	7421.436	4433.649	22251.801	108546.042
...	0.000	0.000	9092.259	0.000	1889.547	0.000	0.000	0.000	2518.098	13499.904
...	0.000	0.000	0.000	562.518	0.000	0.000	0.000	0.000	0.000	3109.722

Take a look at our new column CLV.

```
print(SALES['CLV'].describe())
plt.figure(figsize=(20,5))
sns.distplot(SALES['CLV'], kde=True)
plt.show()
```

✓ 0.3s

```
count      3920.000000
mean       6108.827978
std       24578.215043
min           0.000000
25%         979.060500
50%        2132.229000
75%        5172.857250
max      850552.098000
Name: CLV, dtype: float64
```



I'll create a new data set that only contains the columns need to be interested in.

```
df3 = SALES[['ID_OF_CUSTOMER','Sep-2021','Aug-2021','Jul-2021','Jun-2021','May-2021','Apr-2021','CLV']]
df3.head()
```

✓ 0.9s

MONTH_BY_YEAR	ID_OF_CUSTOMER	Sep-2021	Aug-2021	Jul-2021	Jun-2021	May-2021	Apr-2021	CLV
0	12346.0	0.000	0.000	0.000	0.000	0.000	0.000	254705.880
1	12747.0	0.000	0.000	2228.754	995.610	0.000	1241.790	13846.833
2	12748.0	22251.801	6900.795	18851.184	3869.151	6448.761	3541.263	108546.042
3	12749.0	2518.098	0.000	0.000	0.000	9092.259	0.000	13499.904
4	12820.0	0.000	1134.408	718.641	0.000	0.000	0.000	3109.722

Create a last dataset that I will use it to build the model.

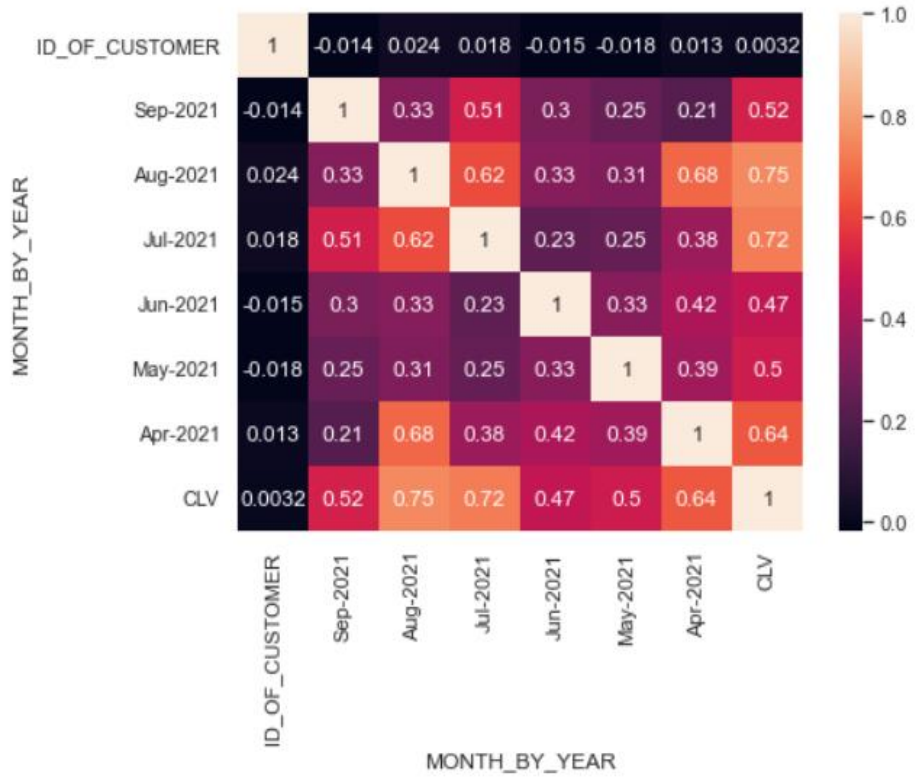
```
dflast = df3.copy()
```

✓ 0.7s

Check the correlation between the columns of the last dataset.

```
plt.figure(figsize=(7,5))
sns.heatmap(dflast.corr(),annot=True)
plt.savefig('heatmap')
plt.show()
```

✓ 1.8s

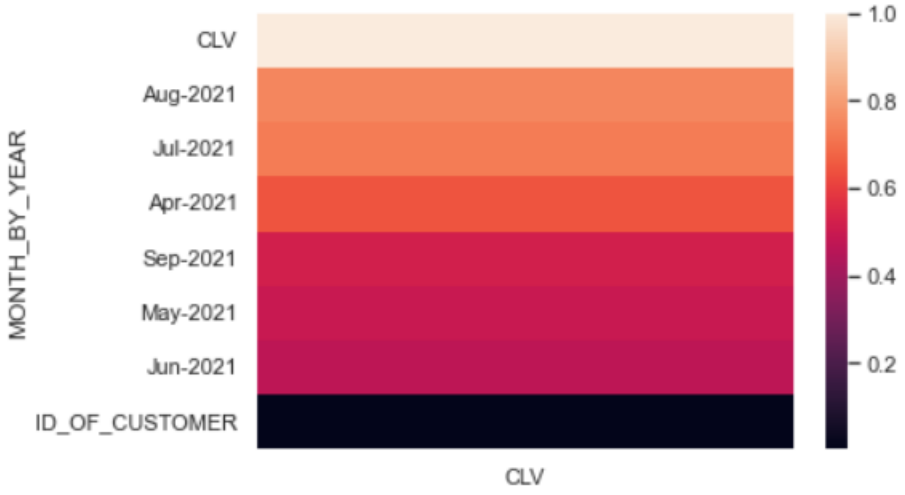


The "CLV" is the feature we will predict. So, how much does each independent variable correlate with this target variable?

```
clv_corr = dflast.corr()["CLV"].sort_values(ascending=False)
print(clv_corr)
sns.heatmap(pd.DataFrame(clv_corr))
```

✓ 0.3s

```
MONTH_BY_YEAR
CLV                1.000000
Aug-2021           0.750516
Jul-2021           0.724179
Apr-2021           0.642231
Sep-2021           0.517859
May-2021           0.495913
Jun-2021           0.467872
ID_OF_CUSTOMER     0.003167
Name: CLV, dtype: float64
```



It is easy to see that all the features have a great influence on the CLV feature except the "ID_OF_CUSTOMER" feature. Therefore, I will not use it for the model.

2.4. Modeling

I will import some models to check which ones is best.

```
# Import models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
```

✓ 0.4s

Define standalone features and Target features.

```
X = dflast[ ['Sep-2021', 'Aug-2021', 'Jul-2021', 'Jun-2021','May-2021','Apr-2021'] ]
y = dflast[ ['CLV']]
```

✓ 0.6s

Split the data to train set and test set.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=40)
```

✓ 0.2s

OK, I will run the models to compare which one is the best fit.

```
models = {
  'LinearRegression': LinearRegression(),
  'K-Nearest Neighbors': KNeighborsRegressor(),
  'Decision Tree': DecisionTreeRegressor(),
  'Random Forest': RandomForestRegressor(),
  'GradientBoostingRegressor' : GradientBoostingRegressor(),
  'Support Vector Machine' : SVR()
}

from sklearn.metrics import mean_squared_error

for name, model in models.items():
  model.fit(X_train, y_train)

for name, model in models.items():
  result = model.score(X_test, y_test)
  y_pred_regressor = model.predict(X_test)
  regressor_mse = mean_squared_error(y_pred_regressor, y_test)
  regressor_rmse = np.sqrt(regressor_mse)
  print(name + f': {round(result*100, 2)}% R_square and ' + f'{round(regressor_rmse, 2)} RMSE')
```

✓ 2.5s

```
LinearRegression: 90.66% R_square and 3793.0 RMSE
K-Nearest Neighbors: 84.89% R_square and 4825.16 RMSE
Decision Tree: 71.95% R_square and 6573.86 RMSE
Random Forest: 89.09% R_square and 4100.53 RMSE
GradientBoostingRegressor: 89.26% R_square and 4067.79 RMSE
Support Vector Machine: -4.86% R_square and 12709.47 RMSE
```

After comparing the R squares and RMSEs of the models, I found that the output of the Linear Regression model is the best. Therefore, in the following section we will use this model for forecasting.

If you don't know what R square or RMSE is, I will a link here for you to click and read.

R_square.

RMSE.

Next, I will run a regression on the entire dataset to see if there is any kind of model.

```
import statsmodels.api as st
X1 = st.add_constant(X)
reg_model=st.OLS(y,X1)
result=reg_model.fit()
print(result.summary())
```

✓ 0.5s

OLS Regression Results

Dep. Variable:	CLV	R-squared:	0.786
Model:	OLS	Adj. R-squared:	0.786
Method:	Least Squares	F-statistic:	2399.
Date:	Thu, 23 Jun 2022	Prob (F-statistic):	0.00
Time:	07:51:00	Log-Likelihood:	-42167.
No. Observations:	3920	AIC:	8.435e+04
Df Residuals:	3913	BIC:	8.439e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	981.8549	189.455	5.183	0.000	610.415	1353.295
Sep-2021	1.1641	0.078	14.866	0.000	1.011	1.318
Aug-2021	1.8467	0.078	23.533	0.000	1.693	2.001
Jul-2021	1.7621	0.054	32.921	0.000	1.657	1.867
Jun-2021	1.9971	0.137	14.606	0.000	1.729	2.265
May-2021	1.4530	0.066	21.922	0.000	1.323	1.583
Apr-2021	1.8677	0.117	15.967	0.000	1.638	2.097

Omnibus:	10748.729	Durbin-Watson:	1.874
Prob(Omnibus):	0.000	Jarque-Bera (JB):	383138267.108
Skew:	33.813	Prob(JB):	0.00

X can explain 78.6 percent of the variability in Y, and the p-values for all variables are less than 0.05. This is good.

LINEAR REGRESSION MODEL

I will fit the data into the linear regression model.

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

✓ 0.8s

```
X_train1 = st.add_constant(X_train)
reg_model = st.OLS(y_train,X_train1)
result = reg_model.fit()
print(result.summary())
```

✓ 0.1s

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    CLV   R-squared:                       0.776
Model:                            OLS   Adj. R-squared:                  0.775
Method:                 Least Squares   F-statistic:                     1579.
Date:                Thu, 23 Jun 2022   Prob (F-statistic):               0.00
Time:                        09:09:29   Log-Likelihood:                -29961.
No. Observations:                2744   AIC:                         5.994e+04
Df Residuals:                    2737   BIC:                         5.998e+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1159.6547    265.436      4.369      0.000     639.180    1680.129
Sep-2021       1.1107      0.096     11.597      0.000       0.923       1.299
Aug-2021       1.9277      0.100     19.333      0.000       1.732       2.123
Jul-2021       1.7724      0.065     27.193      0.000       1.645       1.900
Jun-2021       1.8981      0.173     10.946      0.000       1.558       2.238
May-2021       1.4753      0.082     17.992      0.000       1.314       1.636
Apr-2021       1.7482      0.150     11.671      0.000       1.455       2.042
==============================================================================
Omnibus:                     7175.414   Durbin-Watson:                   2.003
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        149452721.667
Skew:                          29.687   Prob(JB):                         0.00
```

Evaluation

Check the linear regression model's precision

```
print(f'Precision of training set: {regressor.score(X_train, y_train)}')
print(f'Precision of on test set: {regressor.score(X_test, y_test)}')
```

✓ 0.9s

```
Precision of training set: 0.7758919124964715
Precision of on test set: 0.9066089274834814
```

We can see that the precision on the training set is 77.59%. It is good and we can be satisfied with it. The precision on the test set is 90.07%, so it is great.

Next to, I will use the R squared metric, assess the precision of our linear regression model.

```
from sklearn.metrics import mean_squared_error
y_pred_regressor = regressor.predict(X_test)
regressor_r = regressor.score(X_test, y_test)
print(f'Linear Regression R squared: {regressor_r}')
```

✓ 0.7s

```
Linear Regression R squared: 0.9066089274834814
```

This means my model can explain 90.07% of the variability in the 'CLV' feature using our 6 independent features.

Let's build a scatter plot to compare actual and predicted values for a test set.

```
plt.figure(figsize=(8, 6), dpi=80)
plt.scatter(y_test, y_pred_regressor)
plt.xlabel('Actual Values', fontsize=14)
plt.ylabel('Predicted Values', fontsize=14)
plt.title('Actual vs. Predicted for Test Set', fontsize=16)
```

✓ 0.3s

Next, the RMSE metric is used to assess the accuracy of our linear regression model.

```
regressor_mse = mean_squared_error(y_pred_regressor, y_test)
regressor_rmse = np.sqrt(regressor_mse)
print(f'Linear Regression RMSE: {regressor_rmse}')
```

✓ 0.7s

```
Linear Regression RMSE: 3792.9973836278928
```

The RMSE estimates the deviation of the actual y-values from the regression line. This means the model predicted the CLV value of each customer in the test set to within $3792.99 of the true CLV value.

2.5. Deployment

This part, I want to test my model actually work in reallife, so I will import a data of a new customer into my model to predict a new value.

```
# import a new data
new_data = np.array([1000, 2000, 4000, 7000, 9000, 10000]).reshape(1, -1)
new_pred=regressor.predict(new_data)
new_pred1 = int(new_pred[0][0])
print(f"The mean customer CLV is: ${new_pred1}")
```

✓ 0.6s

```
The mean customer CLV is: $57261
```

Since we are dealing with a two-tailed test, let's calculate the Z value for a 99% confidence interval, which is 0.5 percent probability. A two-tailed test is a method that tests whether a sample is greater or less than a range of values by using a two-sided critical area of a distribution. It is used in null-hypothesis testing and statistical significance testing.

If you don’t know about Z value, the Z-value is the regression coefficient divided by standard error. If the z-value is too big in magnitude, it indicates that the corresponding true regression coefficient is not 0 and the corresponding X-variable matters. A good rule of thumb is to use a cut-off value of 2 which approximately corresponds to a two-sided hypothesis test with a significance level of \alpha=0.05.

```
StandardError = regressor_rmse
ExpectedValue = new_pred1
import scipy.stats as si
alpha = 1 - 0.99
zvalue = abs(si.norm.ppf(0.5*alpha))
print(f'Z value: {zvalue}')
```

✓ 0.6s

```
Z value: 2.5758293035489004
```

```
Upper = int(ExpectedValue + zvalue * StandardError)  # compute the upper bound of the range
Lower = int(ExpectedValue - zvalue*StandardError)  # compute the lower bound of the range
print(f"The Customer's CLV is between ${Lower} and ${Upper} with a confidence level of 99%.")
```

✓ 0.7s

```
The Customer's CLV is between $47490 and $67031 with a confidence level of 99%.
```

3. CONCLUSION

In this report, I defined Customer Lifetime Value (CLV), as well as the benefits and challenges of predicting it. I also demonstrated the approach to estimating Customer Lifetime Value using Machine Learning models. I used the CLV dataset developed by Prediction Consultants, a data science firm based in Rishon, Lezion, about transactions that took place between September 2020 and September 2021. Then, using Machine Learning, I went on to explore the data, process it, and build models for CLV prediction. After comparing the models, I decided on Linear Regression as the best fit. I kept evaluating the model, and the results were very promising. Finally, I retest the model with new data to determine the CLV forecast.

4. REFERENCES

[1] Chen, S., 2018. *Estimating Customer Lifetime Value Using Machine Learning Techniques*. [ebook] p.https://www.researchgate.net/. Available at: <https://www.researchgate.net/publication/327182561_Estimating_Customer_Lifetime_Value_Using_Machine_Learning_Techniques> [Accessed 23 June 2022].

[2] Prasasti, N., Okada, M., Kanamori, K. and Ohwada, H., 2014. *Customer Lifetime Value and Defection Possibility Prediction Model Using Machine Learning: An Application to a Cloud-Based Software Company*. [ebook] p.https://link.springer.com/. Available at: <https://link.springer.com/chapter/10.1007/978-3-319-05458-2_7> [Accessed 23 June 2022].

[3] Reddy, K., Swain, D., Shukla, S. and Jacob, L., 2021. *Prediction of Customer Lifetime Value Using Machine Learning*. [ebook] p.https://link.springer.com/. Available at: <https://link.springer.com/chapter/10.1007/978-981-16-3346-1_22> [Accessed 23 June 2022].

[4] Caldwell, A., 2022. *How to Calculate Customer Lifetime Value*. [online] Oracle NetSuite. Available at: <https://www.netsuite.com/portal/resource/articles/ecommerce/customer-lifetime-value-clv.shtml?mc24943=v1> [Accessed 23 June 2022].

[5] Shopenova, A., 2022. *Predict Customer Lifetime Value with Machine Learning*. [online] Medium. Available at: <https://medium.com/swlh/predict-customer-lifetime-value-with-machine-learning-545624073d14> [Accessed 23 June 2022].

[6] Polanitzer, R., 2022. *CLV in Python; Predict the Customer Lifetime Value E2E, from Research to Production*. [online] Medium. Available at: <https://medium.com/@polanitzer/clv-in-python-predict-the-customer-lifetime-value-e2e-from-research-to-production-f07182dadbe7> [Accessed 23 June 2022].

[7] Optimove. 2022. *Customer Lifetime Value (CLV) Definition & Software | Customer LTV | Optimove*. [online] Available at: <https://www.optimove.com/resources/learning-center/customer-lifetime-value#:~:text=Customer%20lifetime%20value%20(CLV%20or,company's%20revenues%20(or%20profits).> [Accessed 23 June 2022].

[8] FERNANDO, J., 2022. *What Is R-Squared?*. [online] Investopedia. Available at: <https://www.investopedia.com/terms/r/r-squared.asp#:~:text=R%2Dsquared%20(R2),variables%20in%20a%20regression%20model.> [Accessed 23 June 2022].

[9] Zach, V., 2022. *How to Interpret Root Mean Square Error (RMSE)*. [online] Statology. Available at: <https://www.statology.org/how-to-interpret-rmse/> [Accessed 23 June 2022].