Predicting Customer Lifetime Value - Beginner

KeytoDataScience.com

For any Data Science project, we will follow the approach of Data Science Project Lifecycle:

💡💡💡💡💡 Data Science Project Life Cycle [1] 💡💡💡💡💡

---

1. **Understand Problem/Objective**
2. **Data Collection**
3. **Data Preparation**
   - 3.1 Data Preprocessing
   - 3.2 EDA[3]
   - 3.3 Train/Validation/Test Split
   - 3.4 Feature Engineering
   - 3.5 Feature Selection
4. **Modeling:** Regression [6]
5. **Evaluation:** Regression
   - RMSE, RSE, MAE, RAE, Coefficient of Determination (R2)
6. **Model Deployment**
   - Model Deployment in pipeline or tool

In this case study, we will use past purchase history of customers to build a model that can predict the **Customer Lifetime Value (CLV or CLTV)** for new customers.

Table of Contents

1. Understand Objective

Customer Lifetime Value(CLTV)[2]

"Customer Lifetime Value is a monetary value that represents the amount of revenue or profit a customer will give the company over the period of the relationship" (Source). CLTV demonstrates the implications of acquiring long-term customers compare to short-term customers. Customer lifetime value (CLV) can help you to answers the most important questions about sales to every company:

1. How to Identify the most profitable customers?
2. How can a company offer the best product and make the most money?
3. How to segment profitable customers?
4. How much budget need to spend to acquire customers?

Business Terms

- **Average Order Value(AOV):** The Average Order value is the ratio of your total revenue and the total number of orders. AOV represents the mean amount of revenue that the customer spends on an order.
  - Average Order Value = Total Revenue / Total Number of Orders
- **Purchase Frequency:** Purchase Frequency is the ratio of the total number of orders and the total number of customer. It represents the average number of orders placed by each customer.
  - Purchase Frequency = Total Number of Orders / Total Number of Customers
- **Churn Rate:** Percentage of customers who have not ordered again.
- **Customer Lifetime:** Customer Lifetime is the period of time that the customer has been continuously ordering.
  - Customer lifetime = 1 / Churn Rate
- **Repeat Rate:** Repeat rate can be defined as the ratio of the number of customers with more than one order to the number of unique customers. Example: If you have 10 customers in a month out of who 4 come back, your repeat rate is 40%.

- Repeat Rate = 1 - Churn Rate

↥ back to top

2. Data Collection

Import Libraries

In [1]:
```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
import os

import matplotlib.pylab as plt
plt.style.use('ggplot')
import seaborn as sns

from sklearn.model_selection  import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler

import sklearn.metrics
from sklearn.metrics import r2_score,mean_absolute_error, mean_squared_error

import joblib
```

Download Data[2]

Load Data

In [2]:
```
df = pd.read_csv("history.csv")
```

We will load the data file for this example and checkout summary statistics and columns for that file.

Check out the Data

In [3]:
```
df.shape
```

Out[3]:
```
(100, 8)
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   CUST_ID   100 non-null    int64
 1   MONTH_1   100 non-null    int64
 2   MONTH_2   100 non-null    int64
 3   MONTH_3   100 non-null    int64
 4   MONTH_4   100 non-null    int64
 5   MONTH_5   100 non-null    int64
```

```
 6   MONTH_6  100 non-null    int64
 7   CLV      100 non-null    int64
dtypes: int64(8)
memory usage: 6.4 KB
```

The dataset consists of the customer ID, the amount the customer spent on your website for the first months of his relationship with your business and his ultimate life time value (say 3 years worth)

In [5]:

```
df.describe()
```

Out[5]:

	CUST_ID	MONTH_1	MONTH_2	MONTH_3	MONTH_4	MONTH_5	MONTH_6	CL
count	100.000000	100.00000	100.000000	100.000000	100.000000	100.000000	100.000000	100.00000
mean	1050.500000	113.25000	115.750000	106.250000	106.750000	106.250000	108.500000	9421.19000
std	29.011492	55.32902	64.221739	63.601406	62.649317	59.816111	66.021499	2664.44317
min	1001.000000	25.00000	0.000000	0.000000	0.000000	0.000000	0.000000	4125.00000
25%	1025.750000	75.00000	75.000000	50.000000	50.000000	50.000000	50.000000	7816.00000
50%	1050.500000	100.00000	125.000000	100.000000	100.000000	100.000000	100.000000	9344.00000
75%	1075.250000	150.00000	175.000000	175.000000	150.000000	156.250000	175.000000	10719.25000
max	1100.000000	200.00000	200.000000	200.000000	200.000000	200.000000	200.000000	17100.00000

In [6]:

```
df.head()
```

Out[6]:

	CUST_ID	MONTH_1	MONTH_2	MONTH_3	MONTH_4	MONTH_5	MONTH_6	CLV
0	1001	150	75	200	100	175	75	13125
1	1002	25	50	150	200	175	200	9375
2	1003	75	150	0	25	75	25	5156
3	1004	200	200	25	100	75	150	11756
4	1005	200	200	125	75	175	200	15525

↥ back to top

3. Data Preparation

3.1 Data Preprocessing

In [7]:

```
# drop CUST_ID
df=df.drop("CUST_ID",axis=1)

# or
# df.drop("CUST_ID",axis=1,inplace=True)
```

Check Null Values

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
MONTH_1    0
MONTH_2    0
MONTH_3    0
MONTH_4    0
MONTH_5    0
MONTH_6    0
CLV        0
dtype: int64
```

Perform Correlation Analysis

In [9]:

```
df.corr()['CLV']
# -1 to 1
```

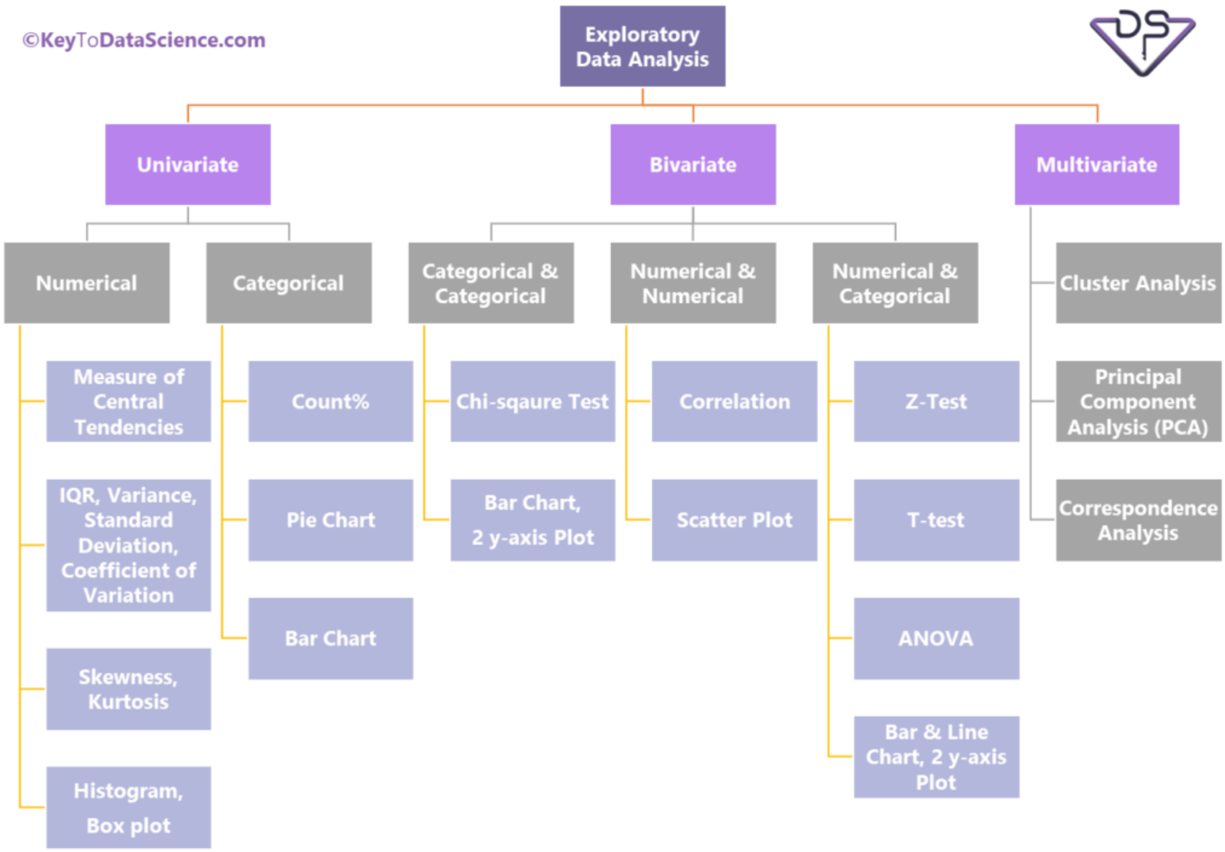
Out[9]:

```
MONTH_1    0.734122
MONTH_2    0.250397
MONTH_3    0.371742
MONTH_4    0.297408
MONTH_5    0.376775
MONTH_6    0.327064
CLV        1.000000
Name: CLV, dtype: float64
```

We can see that the months do show strong correlation to the target variable (CLV). That should give us confidence that we can build a strong model to predict the CLV

3.2 Exploratory Data Analysis (EDA)

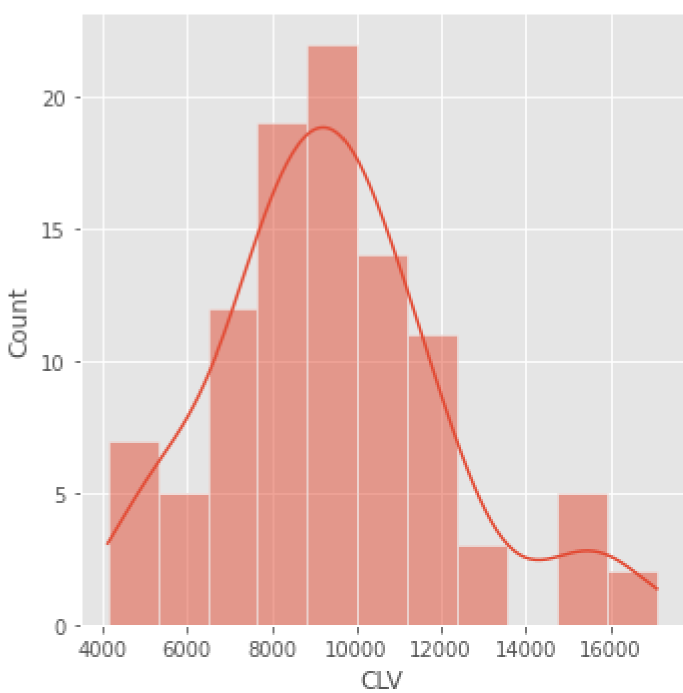
Exploratory Data Analysis (EDA)[3]



Univariate Analysis [4]

In [10]:
```
sns.displot(df['CLV'],kde=True)
```

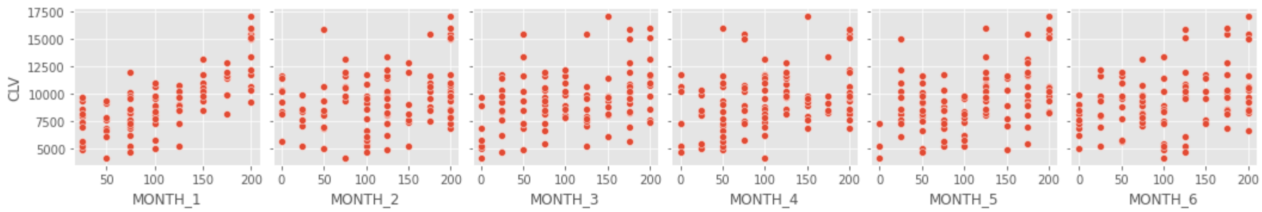
Out[10]:
```
<seaborn.axisgrid.FacetGrid at 0x18521bcfb50>
```



Bivariate Analysis [5]

In [11]:
```python
# df.columns[:-1]
sns.pairplot(df,x_vars=df.columns[:-1],y_vars=['CLV'])
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x18525cf9fd0>
```



3.3 Train and Test Split

Prepare X(independent) and y(dependent) variables

In [12]:
```python
#Drop columns with low correlation
X = df.drop('CLV',axis=1)
y = df.CLV
```

Let us split the data into training and testing datasets in the **ratio 70:30**

In [13]:
```python
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size=.3)
print("X_train - Training : ", X_train.shape)
print("X_test - Testing   : ", X_test.shape )
```

```
X_train - Training :  (70, 6)
X_test - Testing   :  (30, 6)
```

↥ back to top

4. Modeling

Create a function to quickly build models and evaluate

In [14]:
```python
def model_builder(algo,X_train,y_train,X_test):
    algo.fit(X_train,y_train)

    y_train_pred = algo.predict(X_train)
    y_test_pred = algo.predict(X_test)

    print("The model performance for training set")
    print("--------------------------------------")
    print(f'R2  : {round(r2_score(y_true=y_train,y_pred=y_train_pred),2)}')
    print('MAE :', mean_absolute_error(y_train, y_train_pred))
    print('MSE :', mean_squared_error(y_train, y_train_pred))
    print('RMSE:', np.sqrt(mean_squared_error(y_train, y_train_pred)))

    print("\n")

    print("The model performance for testing set")
    print("--------------------------------------")
    print(f'R2  : {round(r2_score(y_true=y_test,y_pred=y_test_pred),2)}')
    print('MAE :', mean_absolute_error(y_test, y_test_pred))
    print('MSE :', mean_squared_error(y_test, y_test_pred))
    print('RMSE:', np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

4.1 Linear Regression [6]

Build Model

We build a Linear Regression equation for predicting CLV and then check its accuracy by predicting against the test dataset

In [15]:

```
# #Build model on training data
# lr = LinearRegression()
# lr.fit(X_train,y_train)
# print("Coefficients :", lr.coef_)
# print("Intercept    :", lr.intercept_)
```

In [16]:

```
# use model_builder function
lr = LinearRegression()
model_builder(lr,X_train,y_train,X_test)
```

```
The model performance for training set
--------------------------------------
R2  : 0.93
MAE : 595.5826744945408
MSE : 509998.45508921274
RMSE: 714.1417611995623


The model performance for testing set
--------------------------------------
R2  : 0.9
MAE : 637.6508538506049
MSE : 598689.529200415
RMSE: 773.7503015834081
```

4.2 DecisionTreeRegressor [7]

In [17]:

```
# from sklearn.tree import DecisionTreeRegressor

dec_tree = DecisionTreeRegressor(random_state=0)
model_builder(dec_tree,X_train,y_train,X_test)
```

```
The model performance for training set
--------------------------------------
R2  : 1.0
MAE : 0.0
MSE : 0.0
RMSE: 0.0


The model performance for testing set
--------------------------------------
R2  : 0.57
MAE : 1286.8333333333333
MSE : 2599523.3666666667
RMSE: 1612.3037451630096
```

4.3 RandomForestRegressor [8]

In [18]:

```
# from sklearn.ensemble import RandomForestRegressor

rf_tree = RandomForestRegressor(random_state=0)
model_builder(rf_tree,X_train,y_train,X_test)
```

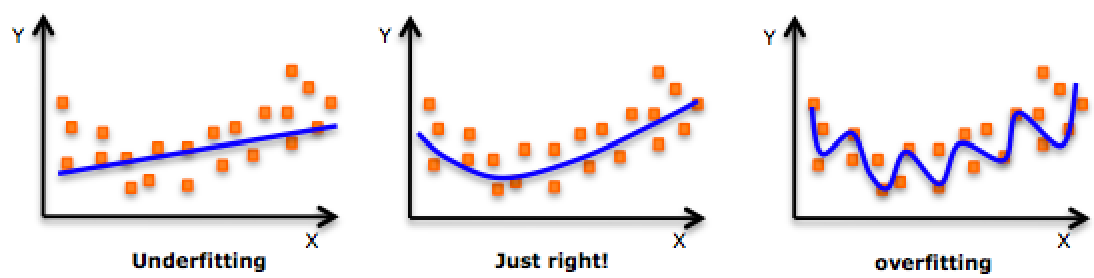
```
The model performance for training set
--------------------------------------
R2   : 0.97
MAE  : 392.26885714285714
MSE  : 260275.18048571429
RMSE: 510.17171666578525


The model performance for testing set
--------------------------------------
R2   : 0.75
MAE  : 920.8510000000002
MSE  : 1483953.4223700003
RMSE: 1218.1762690062553
```

Observation:

Linear Regression model has the highest: R2; and lowest: MAE, MSE, RMSE on the test set.

Both Decision Tree Regressor and Random Forest Regerssor are overfitting



Reference Image for Overfitting

↥ back to top

5. Evaluation [9]

📌 **Regression Evaluation Metrics**

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

5.1 Select Final Model

Final Model Selected for production is: Linear Regression

Linear Regression model has the highest: R2; and lowest: MAE, MSE, RMSE on the test set.

In [19]:

```
y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)

print("The model performance for training set")
print("--------------------------------------")
print(f'R2  : {round(r2_score(y_true=y_train,y_pred=y_train_pred),2)}')
print('MAE :', mean_absolute_error(y_train, y_train_pred))
print('MSE :', mean_squared_error(y_train, y_train_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_train, y_train_pred)))

print("\n")

print("The model performance for testing set")
print("--------------------------------------")
print(f'R2  : {round(r2_score(y_true=y_test,y_pred=y_test_pred),2)}')
print('MAE :', mean_absolute_error(y_test, y_test_pred))
print('MSE :', mean_squared_error(y_test, y_test_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

```
The model performance for training set
--------------------------------------
R2  : 0.93
MAE : 595.5826744945408
MSE : 509998.45508921274
RMSE: 714.1417611995623


The model performance for testing set
--------------------------------------
R2  : 0.9
MAE : 637.6508538506049
```

```
MSE : 598689.529200415
RMSE: 773.7503015834081
```

Linear Regression shows a R-squared of 0.91% on Testing set. This is an excellent model for predicting CLV.

5.2 Save Model to Disk

In [20]:
```
# get current directory path
# os.getcwd()
```

In [21]:
```
# import joblib
filename = os.getcwd()+'/CLV_LinearRegression.joblib.pkl'
joblib.dump(lr, filename, compress=9)
```

Out[21]:
```
['F:\\Work\\Site\\KDS - Career Now Program\\DS\\Syllabus\\5. Case Studies\\Business\\Cus
tome Lifetime Value (CLV)/CLV_LinearRegression.joblib.pkl']
```

5.3 Interpret the Output

In [22]:
```
# print the intercept
print("Intercept:", lr.intercept_)
```

```
Intercept: -250.46672566544476
```

In [23]:
```
coeff_df = pd.DataFrame(lr.coef_,X.columns,columns=['Coefficient'])
coeff_df
```

Out[23]:

	Coefficient
MONTH_1	34.765858
MONTH_2	10.694871
MONTH_3	15.682293
MONTH_4	12.583314
MONTH_5	8.425176
MONTH_6	5.615516

📌 **Interpreting the coefficients:**

- Holding all other features fixed, a 1 unit increase in **MONTH_1** is associated with an **increase of $34.7** in Customer Lifetime Value (CLTV).
- Holding all other features fixed, a 1 unit increase in **MONTH_2** is associated with an **increase of $10.6** in Customer Lifetime Value (CLTV).
- Holding all other features fixed, a 1 unit increase in **MONTH_3** is associated with an **increase of $15.6** in Customer Lifetime Value (CLTV).
- Holding all other features fixed, a 1 unit increase in **MONTH_4** is associated with an **increase of $12.5** in Customer Lifetime Value (CLTV).

- Holding all other features fixed, a 1 unit increase in **MONTH_5** is associated with an **increase of $8.4** in Customer Lifetime Value (CLTV).
- Holding all other features fixed, a 1 unit increase in **MONTH_6** is associated with an **increase of $5.6** in Customer Lifetime Value (CLTV).

5.4 Linear Regression with StandardScaler (Optional)

In [24]:

```
# from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaling = scaler.fit_transform(X_train)
```

💡 To preventing information about the distribution of the test set leaking into the model.

Fit the scaler on your training data only, then standardise both training and test sets with that scaler.

In [25]:

```
# we have to scale X_test also, before predicting
# Use transform() on the test data, not fit_transform(), as fit is done on training set
X_test_scaling = scaler.transform(X_test)
```

In [26]:

```
lr_scaling = LinearRegression()
model_builder(lr_scaling,X_train_scaling,y_train,X_test_scaling)
```

```
The model performance for training set
--------------------------------------
R2  : 0.93
MAE : 595.5826744945401
MSE : 509998.4550892126
RMSE: 714.1417611995623


The model performance for testing set
--------------------------------------
R2  : 0.9
MAE : 637.6508538506054
MSE : 598689.5292004154
RMSE: 773.7503015834084
```

In [27]:

```
coeff_df_scaling = pd.DataFrame(lr_scaling.coef_,X.columns,columns=['Coefficient'])
coeff_df_scaling
```

Out[27]:

	Coefficient
MONTH_1	1964.026074
MONTH_2	678.352205
MONTH_3	980.639262
MONTH_4	773.654048
MONTH_5	501.735199

	Coefficient
MONTH_6	368.845216

↥ back to top

6. Model Deployment

The code in model deployment should be able to run independently, i.e. the below code should be independent of all the above performed steps.

Predicting for a new Customer

Let's use the model to predict his CLV.

6.1 Import Libraries

In [28]:

```
# import os
# import joblib
# from sklearn.linear_model import LinearRegression
```

6.2 Load Model from Disk

In [29]:

```
filename = os.getcwd()+'/CLV_LinearRegression.joblib.pkl'
```

In [30]:

```
model = joblib.load(filename)
```

6.3 Real Time Prediction

New Customer Data

Say we have a new customer who in his first 3 months have spend **100,0,50** on the website.

In [31]:

```
new_data = np.array([100,0,50,0,0,0]).reshape(1, -1)
```

Real Time Prediction

In [32]:

```
new_pred=model.predict(new_data)
print("The CLV for the new customer is : $",new_pred[0])
```

```
The CLV for the new customer is : $ 4010.2337033822596
```

↥ back to top

Great Job!

Links Used in this notebook:

1. Data Science Project Life Cycle [1]
2. Customer Lifetime Value(CLTV) [2]
3. Exploratory Data Analysis (EDA) [3]
4. Univariate Analysis [4]
5. Bivariate Analysis [5]
6. Regression [6]
7. Decision Tree [7]
8. Random Forest [8]
9. Model Evaluation [9]

KeytoDataScience.com