To understand the logistic Regression that includes non-linearity to linear regression

```
# 1.Load the basic libraries and packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

```
# 2.    Load the dataset

data = pd.read_excel("/content/default_of_credit_card_clients.xls" , skiprows=1)
data
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	..
0	1	20000	2	2	1	24	2	2	-1	-1	
1	2	120000	2	2	2	26	-1	2	0	0	
2	3	90000	2	2	2	34	0	0	0	0	
3	4	50000	2	2	1	37	0	0	0	0	
4	5	50000	1	2	1	57	-1	0	-1	0	
...	...	...	...	...	...	...	...	...	...	...	
29995	29996	220000	1	3	1	39	0	0	0	0	
29996	29997	150000	1	3	2	43	-1	-1	-1	-1	
29997	29998	30000	1	2	2	37	4	3	2	-1	
29998	29999	80000	1	3	1	41	1	-1	0	0	
29999	30000	50000	1	2	1	46	0	0	0	0	

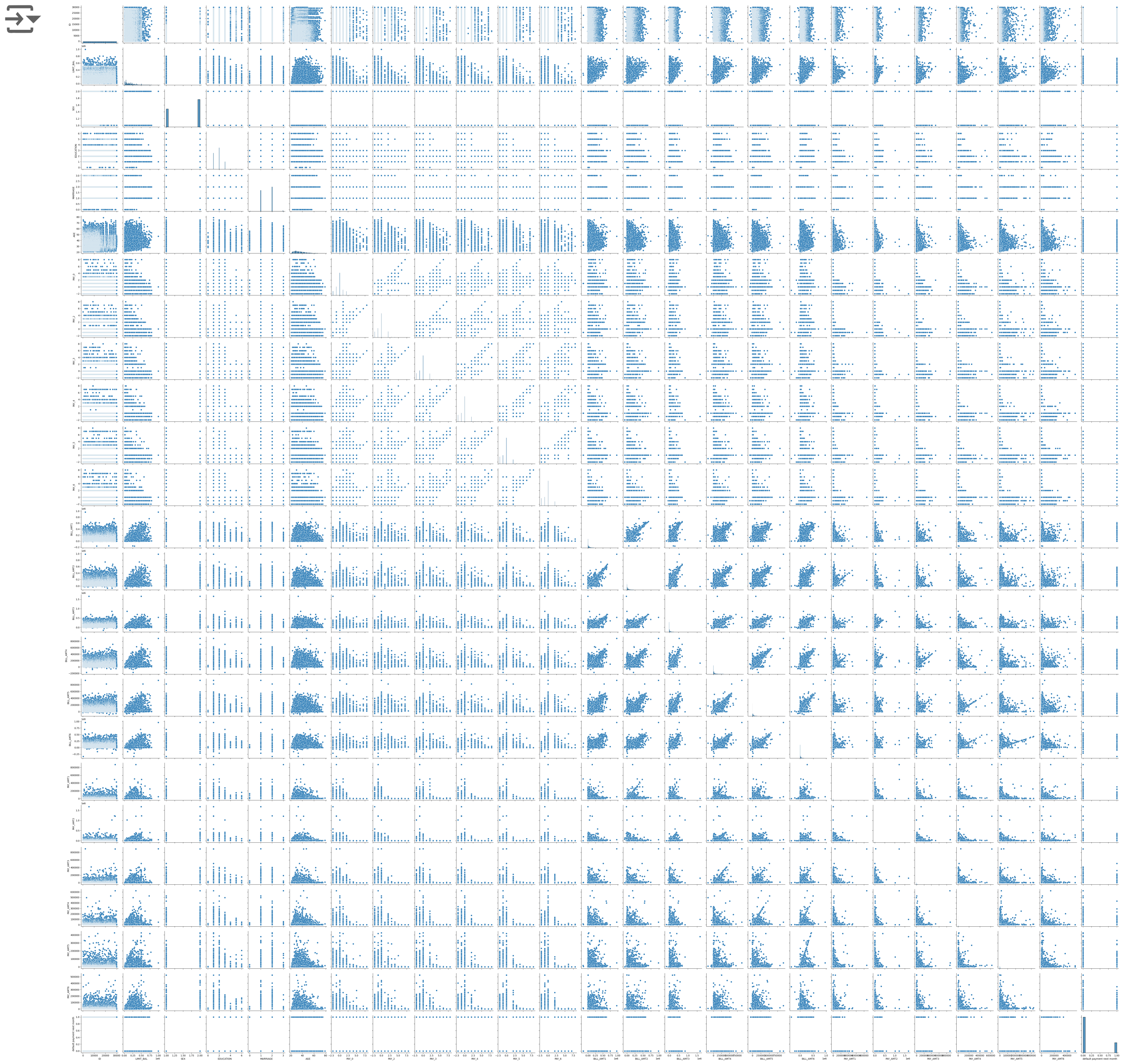
30000 rows × 25 columns

```
# 3.    Analyze the dataset

data.describe()
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.00
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867	35.48
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.21
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.00
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.00
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.00
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.00
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.00

8 rows × 25 columns

```
# 4.    Normalize the data

def Feature_Normalization(X):
  X = (X - np.mean(X , axis = 0)) / np.std(X , axis = 0)
  return X , np.mean(X , axis = 0) ,  np.std(X , axis = 0)


# 5.    Pre-process the data

x_train , x_test , y_train , y_test = train_test_split(x , y , test_size = 0.2 , random_s


# 6.    Visualize the Data

sns.pairplot(data)
plt.show()
```



```
# 7.Separate the feature and prediction value columns

x = data.iloc[: , :-1].values
y = data.iloc[: , -1].values


def Sigmoid(z):
    return 1 / (1 + np.exp(-z))


# 8.    Write the Hypothesis Function

def Hypothesis(theta, X):
    return Sigmoid(np.dot(X, theta))


# 9. Write the Cost Function

def Cost_function(theta, X, y):
    m = len(y)  # number of training examples
    h = Sigmoid(np.dot(X, theta))  # hypothesis (predicted probabilities)
    cost = (-1/m) * (np.dot(y.T, np.log(h)) + np.dot((1 - y).T, np.log(1 - h)))
    return cost


# 10.   Write the Gradient Descent optimization algorithm

def Gradient_Descent(X, y, theta_array, alpha, epochs):
    m = len(y)
    cost_history = []

    for i in range(epochs):
        h = Sigmoid(np.dot(X, theta_array))
        gradient = (1/m) * np.dot(X.T, (h - y))
        theta_array = theta_array - alpha * gradient
```

```
        cost = Cost_function(theta_array, X, y)
        cost_history.append(cost)

    return theta_array, cost_history


# 11.   Apply Feature Normalization technique over the data

x_train , train_mean , train_std = Feature_Normalization(x_train)
x_test , test_mean , test_std = Feature_Normalization(x_test)


# 12.   Apply the training over the dataset to minimize the loss

def Training(X, y, alpha, epochs):
    theta_array = np.zeros(X.shape[1])  # Initialize theta as a zero array with shape mat
    cost_history = []

    theta_array, cost_history = Gradient_Descent(X, y, theta_array, alpha, epochs)

    return theta_array, cost_history


# 13.   Observe the cost function vs iterations learning curve

x = np.arange(0, epochs)
plt.plot(x, cost_history)
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.show()
```