```
# 1. Importing the necessary libraries and Modules

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import random


# 2. Importing the Dataset

# Read the Excel file without specifying usecols to inspect the actual column names
dataset = pd.read_excel("/content/default_of_credit_card_clients.xls" , skiprows = 1)
dataset
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6
0	1	20000	2	2	1	24	2	2	-1	-1	...	689	0	0	0
1	2	120000	2	2	2	26	-1	2	0	0	...	2682	3272	3455	3261
2	3	90000	2	2	2	34	0	0	0	0	...	13559	14331	14948	15549
3	4	50000	2	2	1	37	0	0	0	0	...	49291	28314	28959	29547
4	5	50000	1	2	1	57	-1	0	-1	0	...	35835	20940	19146	19131
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	29996	220000	1	3	1	39	0	0	0	0	...	208365	88004	31237	15980
29996	29997	150000	1	3	2	43	-1	-1	-1	-1	...	3502	8979	5190	0
29997	29998	30000	1	2	2	37	4	3	2	-1	...	2758	20878	20582	19357
29998	29999	80000	1	3	1	41	1	-1	0	0	...	76304	52774	11855	48944
29999	30000	50000	1	2	1	46	0	0	0	0	...	49764	36535	32428	15313

30000 rows × 24 columns

```
# 3. Geting the Statistics of a Dataset

dataset.describe()
```

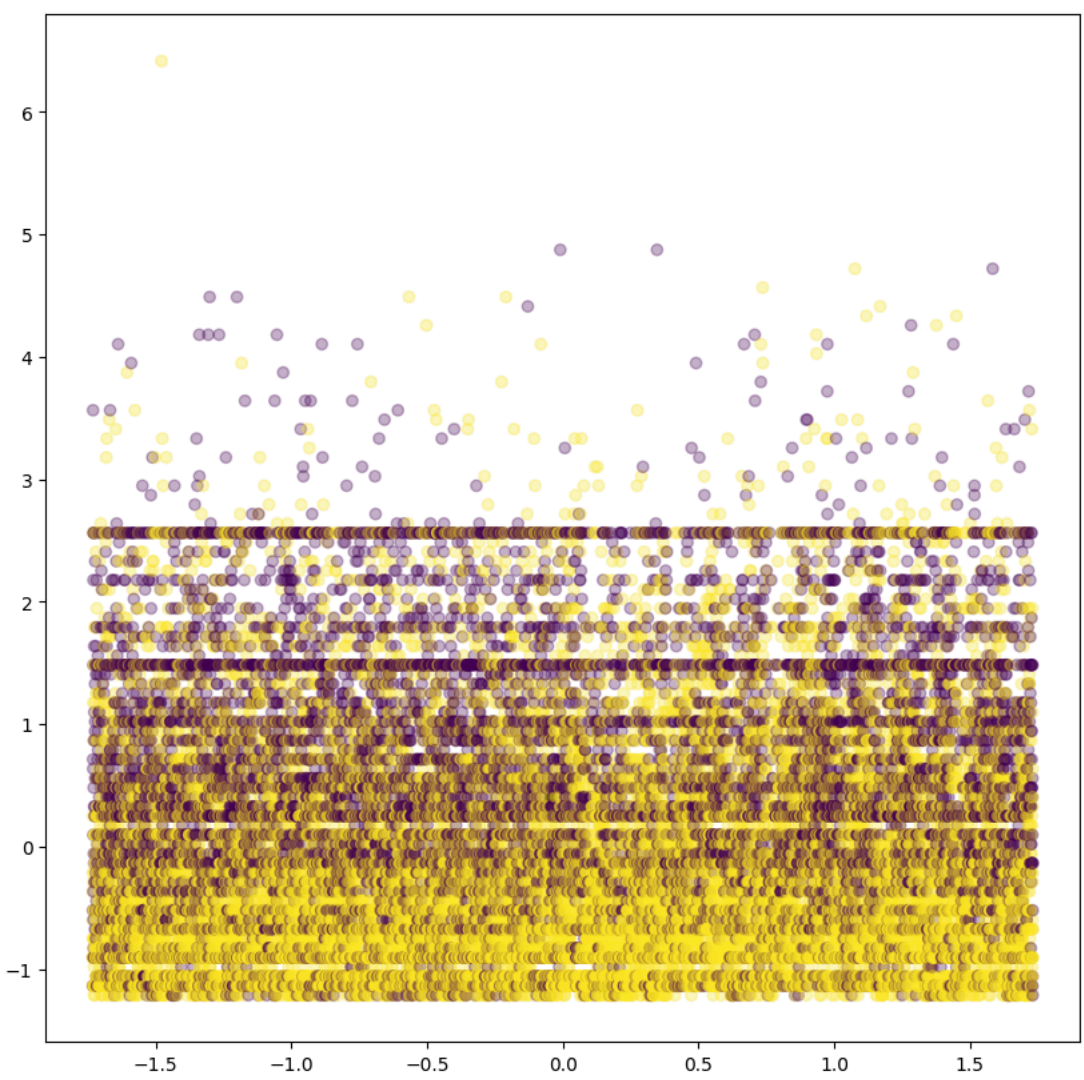
	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	
	00.000000	30000.000000	30000.000000	30000.000000	30000.000000	3
	1.551867	35.485500	-0.016700	-0.133767	-0.166200	
	0.521970	9.217904	1.123802	1.197186	1.196868	
	0.000000	21.000000	-2.000000	-2.000000	-2.000000	
	1.000000	28.000000	-1.000000	-1.000000	-1.000000	
	2.000000	34.000000	0.000000	0.000000	0.000000	
	2.000000	41.000000	0.000000	0.000000	0.000000	
	3.000000	79.000000	8.000000	8.000000	8.000000	

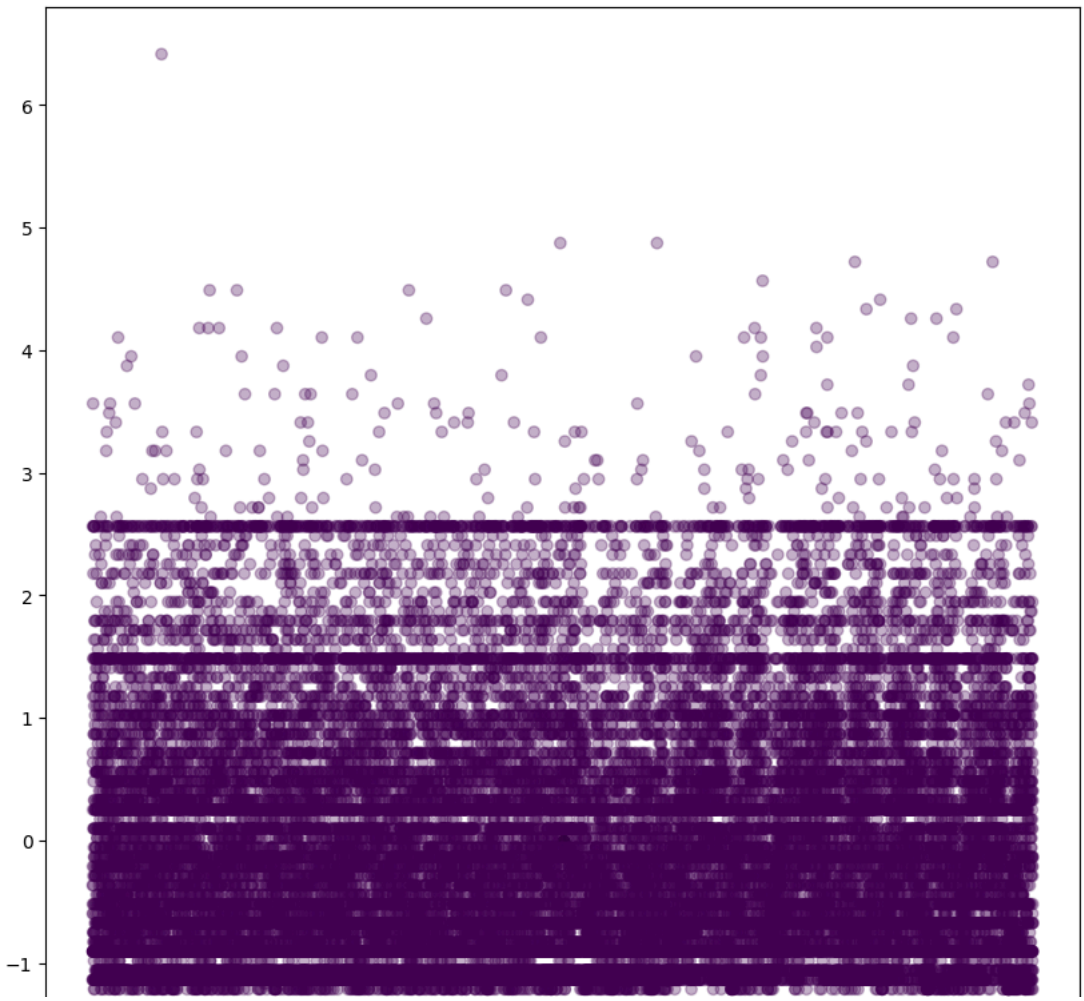
```
# 4.    Pre-process the data

def Feature_Normalization(X):
  X = (X - np.mean(X)) / np.std(X)  # Calculate mean and std across the entire 1D array
  return X

x = dataset.iloc[: , :].values

# Initialize x_norm as a list to store normalized features
x_norm = []

for i in range(x.shape[1]):  # Iterate through columns of x
    norm_feature = Feature_Normalization(x[:, i])
    x_norm.append(norm_feature)  # Append normalized feature to the list
```

```
# 5. Find the the no. of Clusters

# Randomly initialize the centroids by selecting 3 unique points from the dataset
# `random.sample(range(0, len(dataset)), 3)` will select 3 random indices without replacement from the dataset.
# Each selected index represents a row in the dataset, which will serve as an initial centroid.
init_centroids = random.sample(range(0, len(dataset)), 2)

# Create an empty list to store the initial centroid points.
centroids = []
for i in init_centroids:
    # Append the data point at each randomly chosen index to the centroids list.
    centroids.append(dataset.iloc[i])

# Convert the list of centroids to a NumPy array for easier manipulation in further calculations.
centroids = np.array(centroids)

# Print the initialized centroids
centroids
```

```
array([[-1.49343194,  0.63598104, -1.23432296,  0.18582826,  0.85855728,
         0.48976158,  0.90471219, -1.55887596, -1.53219171, -1.52194355,
        -1.53004603, -1.48604076, -0.69564183, -0.69098343, -0.67792868,
        -0.67249727, -0.66305853, -0.65272422, -0.34194162, -0.25698952,
        -0.29680127, -0.30806256, -0.31413612, -0.29338206],
       [-0.96746585, -0.90549825,  0.81016074,  0.18582826, -1.05729503,
         0.59824792,  1.79456386,  1.78234817,  1.8099213 ,  0.18874609,
         0.23491652,  0.25313738, -0.20227344, -0.17518087, -0.26818499,
        -0.3855629 , -0.39861741, -0.44403656, -0.1363026 , -0.25698952,
        -0.26045141, -0.26593284, -0.29476192, -0.29000694]])
```

```
# 5. Preprocessing the Data

X=np.array(x_norm).T


# 6. Function to Claculate the Euclidian Distance

def  Calculate_Distance(i,j):
  return np.sqrt(np.sum((i-j)**2))


# 7. Function to find the Nearest Centroid

def Find_Nerest_Centroid(centroids,X):
  assigned_cluster=[]
  for i in X:
    dist=[]#list of calced distances
    for j in centroids:
      dist.append(Calculate_Distance(i,j))
    assigned_cluster.append(np.argmin(dist))
  return assigned_cluster


# 8. Function For Getting the New Centroids i.e. Updating the Old Centroids

def Calculating_Centroids(cluster_number,X):
  new_centorids=[]
  new_df=pd.concat([pd.DataFrame(X),pd.Series(cluster_number,name='cluster')],axis=1)
  for c in set(new_df['cluster']):
    current_cluster=new_df[new_df['cluster']==c][new_df.columns[:-1]]
    cluster_mean=current_cluster.mean(axis=0)
    new_centorids.append(cluster_mean)
    return new_centorids


# 9. Training the Model

epochs=10
for i in range(epochs):
  get_centroids=Find_Nerest_Centroid(centroids,X)
  centroids=Calculating_Centroids(get_centroids,X)
  if i == 0 or i == epochs - 1 :
    plt.figure(figsize=(10,10))
    plt.scatter(np.array(centroids)[:,0],np.array(centroids)[:,1],color="black")
    plt.scatter(X[:,0],X[:,1],c=get_centroids,alpha =0.3)
    plt.show()
```


6
5
4
3
2
1
0
−1
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


6
5
4
3
2
1
0
−1