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

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

```
# 2.  Load the dataset

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_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	P
0	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	0	0	
1	2	120000	2	2	2	26	-1	2	0	0	...	3272	3455	3261	0	
2	3	90000	2	2	2	34	0	0	0	0	...	14331	14948	15549	1518	
3	4	50000	2	2	1	37	0	0	0	0	...	28314	28959	29547	2000	
4	5	50000	1	2	1	57	-1	0	-1	0	...	20940	19146	19131	2000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
29995	29996	220000	1	3	1	39	0	0	0	0	...	88004	31237	15980	8500	
29996	29997	150000	1	3	2	43	-1	-1	-1	-1	...	8979	5190	0	1837	
29997	29998	30000	1	2	2	37	4	3	2	-1	...	20878	20582	19357	0	
29998	29999	80000	1	3	1	41	1	-1	0	0	...	52774	11855	48944	85900	
29999	30000	50000	1	2	1	46	0	0	0	0	...	36535	32428	15313	2078	

30000 rows × 25 columns

```
# 3.    Analyse the dataset

dataset.describe()
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700	-0.133767	-0.166200
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802	1.197186	1.196868
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.000000	-2.000000
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.000000	-1.000000
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.000000	0.000000
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.000000	0.000000
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.000000	8.000000

8 rows × 25 columns

```
# 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[: , :-1].values
y = dataset.iloc[: , -1].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
```

```
# Convert the list of normalized features to a NumPy array
x_norm = np.array(x_norm).T  # Transpose to get the desired shape
x_norm
```

```
array([[-1.73199307, -1.13672015,  0.81016074, ..., -0.30806256,
        -0.31413612, -0.29338206],
       [-1.7318776 , -0.3659805 ,  0.81016074, ..., -0.24422965,
        -0.31413612, -0.18087821],
       [-1.73176213, -0.59720239,  0.81016074, ..., -0.24422965,
        -0.24868274, -0.01212243],
       ...,
       [ 1.73176213, -1.05964618, -1.23432296, ..., -0.03996431,
        -0.18322937, -0.11900109],
       [ 1.7318776 , -0.67427636, -1.23432296, ..., -0.18512036,
         3.15253642, -0.19190359],
       [ 1.73199307, -0.90549825, -1.23432296, ..., -0.24422965,
        -0.24868274, -0.23713013]])
```

```
# 5.    Visualize the Data
```

```
# Pairplot to visualize relationships
sns.pairplot(dataset, hue="default payment next month")
plt.show()
```

Show hidden output

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

```
# Separate the features and target variable
training = dataset.values[:, :-1]  # All feature columns
testing = dataset.values[29999, :-1]  # Using the last row as the test sample
training = dataset.values[:29999, :-1]  # Training set (excluding the last row)
```

```
# 7.    Select the number K of the neighbors
```

```
k = 25
```

```
# 8.  Calculate the Euclidean distance of K number of neighbors
```

```
def Euclidean_Distance(row_i, row_j):
    distance = 0.0
    for i in range(len(row_i)):
        distance += (row_i[i] - row_j[i])**2
    return np.sqrt(distance)

# Assuming 'y' contains the target variable values for the training set
trainingclass = y[:29999] # Extract target variable values for training set

# Calculate distances between the test sample and each training sample
distance = []
for i in range(len(training)):
    dist = Euclidean_Distance(training[i], testing)
    distance.append([dist, trainingclass[i]]) # Use 'trainingclass' instead of 'traningclass'
```

```
# 9.  Take the K nearest neighbors as per the calculated Euclidean distance.
```

```
# Sort the distances and select the first k
distance.sort()
k_nearest_neighbors = distance[:k]
```

```
# 10. Among these k neighbors, count the number of the data points in each category.
```

```
# Count occurrences of each class in the K nearest neighbors
result = {}
for dist, label in k_nearest_neighbors:
    result[label] = result.get(label, 0) + 1
```

```
# 12. Determine the predicted class based on the majority class among the K nearest neighbors.
# Assuming 'unique_list' is supposed to hold unique class labels
# Replace this with the actual list of your unique class labels if different
unique_list = list(set(y))  # Create unique_list from target variable 'y'

# Determine the class with the highest count
max_key = max(result, key=result.get)
class_name = unique_list[max_key]
print("Predicted Class:", class_name) # Predicted Class: 0
```