Student Name: Tushar Panchal

En. No. : 21162101014

Branch: CBA

Batch: 71

Subject: ML (Machine Learning)

EXPERIMENT 5

Credit Card Default Prediction

Instructions:

Financial threats are displaying a trend about the credit risk of commercial banks as the incredible improvement in the financial industry has arisen. In this way, one of the biggest threats faces by commercial banks is the risk prediction of credit clients. The goal is to predict the probability of credit default based on credit card owner's characteristics and payment history.

Approach: The classical machine learning tasks like Data Exploration, Data Cleaning, Feature Engineering, Model Building and Model Testing. Try out different machine learning algorithms that's best fit for the above case.

Results: You have to build a solution that should able to predict the probability of credit default based on credit card owner's characteristics and payment history.

Dataset: https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset

Try different classification models and justify which one is best using any accuracy measures.

IMPORT DATASETS

In [7]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [8]:
```
import pandas as pd
df=pd.read_csv('/content/drive/MyDrive/ML_DATASETS/ML_5/UCI_Credit_Card.csv')
print(df.head())
```

```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0   1    20000.0    2          2         1   24      2      2     -1     -1
1   2   120000.0    2          2         2   26     -1      2      0      0
2   3    90000.0    2          2         2   34      0      0      0      0
3   4    50000.0    2          2         1   37      0      0      0      0
4   5    50000.0    1          2         1   57     -1      0     -1      0

   ...  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  \
0  ...        0.0        0.0        0.0       0.0     689.0       0.0
1  ...     3272.0     3455.0     3261.0       0.0    1000.0    1000.0
2  ...    14331.0    14948.0    15549.0    1518.0    1500.0    1000.0
3  ...    28314.0    28959.0    29547.0    2000.0    2019.0    1200.0
4  ...    20940.0    19146.0    19131.0    2000.0   36681.0   10000.0

   PAY_AMT4  PAY_AMT5  PAY_AMT6  default.payment.next.month
0       0.0       0.0       0.0                           1
1    1000.0       0.0    2000.0                           1
2    1000.0    1000.0    5000.0                           0
3    1100.0    1069.0    1000.0                           0
4    9000.0     689.0     679.0                           0

[5 rows x 25 columns]
```

In [9]:
```
# Check for missing values
print("Missing values in each column:")
print(df.isnull().sum())

# Drop rows with missing values if any (if necessary)
df.dropna(inplace=True)

# Convert categorical variables to numeric using one-hot encoding if needed
df = pd.get_dummies(df, columns=['SEX', 'EDUCATION', 'MARRIAGE'], drop_first=True)

# Verify changes
print("Data after processing:")
print(df.head())
```

```
Missing values in each column:
ID                            0
LIMIT_BAL                     0
SEX                           0
EDUCATION                     0
MARRIAGE                      0
AGE                           0
PAY_0                         0
PAY_2                         0
PAY_3                         0
PAY_4                         0
PAY_5                         0
PAY_6                         0
BILL_AMT1                     0
BILL_AMT2                     0
BILL_AMT3                     0
BILL_AMT4                     0
BILL_AMT5                     0
BILL_AMT6                     0
PAY_AMT1                      0
PAY_AMT2                      0
PAY_AMT3                      0
PAY_AMT4                      0
PAY_AMT5                      0
PAY_AMT6                      0
default.payment.next.month    0
dtype: int64
Data after processing:
   ID  LIMIT_BAL  AGE  PAY_0  PAY_2  PAY_3  PAY_4  PAY_5  PAY_6  BILL_AMT1  \
0   1    20000.0   24      2      2     -1     -1     -2     -2     3913.0
1   2   120000.0   26     -1      2      0      0      0      2     2682.0
2   3    90000.0   34      0      0      0      0      0      0    29239.0
3   4    50000.0   37      0      0      0      0      0      0    46990.0
4   5    50000.0   57     -1      0     -1      0      0      0     8617.0

   ...  SEX_2  EDUCATION_1  EDUCATION_2  EDUCATION_3  EDUCATION_4  \
0  ...   True        False         True        False        False
1  ...   True        False         True        False        False
2  ...   True        False         True        False        False
3  ...   True        False         True        False        False
4  ...  False        False         True        False        False

   EDUCATION_5  EDUCATION_6  MARRIAGE_1  MARRIAGE_2  MARRIAGE_3
0        False        False        True       False       False
1        False        False       False        True       False
2        False        False       False        True       False
3        False        False        True       False       False
4        False        False        True       False       False

[5 rows x 32 columns]
```

In [11]:

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Define features and target variable
X = df.drop(columns=['ID', 'default.payment.next.month'])  # Features
y = df['default.payment.next.month']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Initialize and fit the logistic regression model
logreg = LogisticRegression(max_iter=1000)
```

```
logreg.fit(X_train, y_train)

# Predict and evaluate the model
y_pred = logreg.predict(X_test)

# Print the classification report
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.97      0.88      7040
           1       0.65      0.20      0.31      1960

    accuracy                           0.80      9000
   macro avg       0.73      0.59      0.60      9000
weighted avg       0.78      0.80      0.76      9000
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [12]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

# Initialize and fit the KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predict and evaluate the model
y_pred_knn = knn.predict(X_test)

# Print the classification report
print("K-Nearest Neighbors Classification Report:")
print(classification_report(y_test, y_pred_knn))
```

```
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.91      0.85      7040
           1       0.36      0.17      0.23      1960

    accuracy                           0.75      9000
   macro avg       0.58      0.54      0.54      9000
weighted avg       0.70      0.75      0.72      9000
```

In [13]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

# Initialize and fit the decision tree model
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

# Predict and evaluate the model
y_pred_dt = dt.predict(X_test)
```

```python
# Print the classification report
print("Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt))
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.81      0.82      7040
           1       0.38      0.41      0.39      1960

    accuracy                           0.72      9000
   macro avg       0.60      0.61      0.61      9000
weighted avg       0.73      0.72      0.73      9000
```

In [14]:

```python
from sklearn.metrics import jaccard_score, confusion_matrix, precision_recall_fscor

# Compute evaluation metrics for Logistic Regression
print("Logistic Regression Evaluation Metrics:")
print("Jaccard Score:", jaccard_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred, average=
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

print("Log Loss:", log_loss(y_test, logreg.predict_proba(X_test)))
print("Classification Report:\n", classification_report(y_test, y_pred, zero_divisi

# Compute evaluation metrics for K-Nearest Neighbors
print("\nk-Nearest Neighbors Evaluation Metrics:")
print("Jaccard Score:", jaccard_score(y_test, y_pred_knn))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))

precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred_knn, aver
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

print("Log Loss:", log_loss(y_test, knn.predict_proba(X_test)))
print("Classification Report:\n", classification_report(y_test, y_pred_knn, zero_di

# Compute evaluation metrics for Decision Tree
print("\nDecision Tree Evaluation Metrics:")
print("Jaccard Score:", jaccard_score(y_test, y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))

precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred_dt, avera
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

print("Log Loss:", log_loss(y_test, dt.predict_proba(X_test)))
print("Classification Report:\n", classification_report(y_test, y_pred_dt, zero_div
```

```
Logistic Regression Evaluation Metrics:
Jaccard Score: 0.18064516129032257
Confusion Matrix:
 [[6830  210]
 [1568  392]]
Precision: 0.6511627906976745
Recall: 0.2
F1 Score: 0.30601092896174864
Log Loss: 0.48189313395142935
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.97      0.88      7040
           1       0.65      0.20      0.31      1960

    accuracy                           0.80      9000
   macro avg       0.73      0.59      0.60      9000
weighted avg       0.78      0.80      0.76      9000


k-Nearest Neighbors Evaluation Metrics:
Jaccard Score: 0.13237016790316283
Confusion Matrix:
 [[6439  601]
 [1621  339]]
Precision: 0.3606382978723404
Recall: 0.17295918367346938
F1 Score: 0.23379310344827586
Log Loss: 2.3165278164776297
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.91      0.85      7040
           1       0.36      0.17      0.23      1960

    accuracy                           0.75      9000
   macro avg       0.58      0.54      0.54      9000
weighted avg       0.70      0.75      0.72      9000


Decision Tree Evaluation Metrics:
Jaccard Score: 0.24430955993930198
Confusion Matrix:
 [[5705 1335]
 [1155  805]]
Precision: 0.37616822429906543
Recall: 0.4107142857142857
F1 Score: 0.3926829268292683
Log Loss: 9.96052498464641
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.81      0.82      7040
           1       0.38      0.41      0.39      1960

    accuracy                           0.72      9000
   macro avg       0.60      0.61      0.61      9000
weighted avg       0.73      0.72      0.73      9000
```