Credit card default prediction

Table of Contents

1.Introduction

The primary objective of this project is to leverage machine learning to predict whether a credit card user is likely to default on their payments. By evaluating past financial behaviors and patterns, we aim to provide credit decisions that are both responsible and sustainable

2.Dataset Overview

2.1 Dataset Information

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.

2.2 Variable Info

```
There are 25 features:

ID: ID of each client
LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
SEX: Gender (1=male, 2=female)
EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
MARRIAGE: Marital status (1=married, 2=single, 3=others)
AGE: Age in years
PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, … 8=payment delay for eight months, 9=payment delay for nine months and above)
PAY_2: Repayment status in August, 2005 (scale same as above)
PAY_3: Repayment status in July, 2005 (scale same as above)
PAY_4: Repayment status in June, 2005 (scale same as above)
PAY_5: Repayment status in May, 2005 (scale same as above)
PAY_6: Repayment status in April, 2005 (scale same as above)
BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
default.payment.next.month: Default payment (1=yes, 0=no)

```

3.Import libraries

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries ins
# It is defined by the kaggle/python docker image: https://github.com/kagg
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization purposes
import seaborn as sns # for data visualization
%matplotlib inline
```

In [2]:

```
import warnings

warnings.filterwarnings('ignore')
```

In [3]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,confusion_matrix,classification_
```

In [4]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
```

In [5]:

```
# 4.import Dataset
```

In [6]:

```
df = pd.read_csv(r"D:\Capstone real time project\Internship\default-of-cre
```

In [7]:

```
df.head()
```

Out[7]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_
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	...	

5 rows × 25 columns

In [8]:

```
df.drop(['ID'],axis=1,inplace=True) # removing unnessory attribute

```

In [9]:

```
# Rename the column name
df = df.rename(columns={'default payment next month':'default'})
```

In [10]:
```
df.tail()
```

Out[10]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	..
29995	220000	1	3	1	39	0	0	0	0	0	..
29996	150000	1	3	2	43	-1	-1	-1	-1	0	..
29997	30000	1	2	2	37	4	3	2	-1	0	..
29998	80000	1	3	1	41	1	-1	0	0	0	..
29999	50000	1	2	1	46	0	0	0	0	0	..

5 rows × 24 columns

In [11]:
```
df.shape
```

Out[11]: (30000, 24)

In [12]:
```
df.isnull().sum() # there is no missing values
```

Out[12]:
```
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      0
dtype: int64
```

In [13]:
```
df.info() # summary of dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 24 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   LIMIT_BAL  30000 non-null  int64
 1   SEX        30000 non-null  int64
 2   EDUCATION  30000 non-null  int64
 3   MARRIAGE   30000 non-null  int64
 4   AGE        30000 non-null  int64
 5   PAY_0      30000 non-null  int64
 6   PAY_2      30000 non-null  int64
 7   PAY_3      30000 non-null  int64
 8   PAY_4      30000 non-null  int64
 9   PAY_5      30000 non-null  int64
 10  PAY_6      30000 non-null  int64
 11  BILL_AMT1  30000 non-null  int64
 12  BILL_AMT2  30000 non-null  int64
 13  BILL_AMT3  30000 non-null  int64
 14  BILL_AMT4  30000 non-null  int64
 15  BILL_AMT5  30000 non-null  int64
 16  BILL_AMT6  30000 non-null  int64
 17  PAY_AMT1   30000 non-null  int64
 18  PAY_AMT2   30000 non-null  int64
 19  PAY_AMT3   30000 non-null  int64
 20  PAY_AMT4   30000 non-null  int64
 21  PAY_AMT5   30000 non-null  int64
 22  PAY_AMT6   30000 non-null  int64
 23  default    30000 non-null  int64
dtypes: int64(24)
memory usage: 5.5 MB
```

In [14]:
```
# Above  shows all variables are numerical but some variable categorical we
```

In [15]:
```
df['SEX'].value_counts()
```

Out[15]:
```
2    18112
1    11888
Name: SEX, dtype: int64
```

In [16]:
```
df['EDUCATION'].value_counts()
```

Out[16]:
```
2    14030
1    10585
3     4917
5      280
4      123
6       51
0       14
Name: EDUCATION, dtype: int64
```

In [17]:
```
# Put 0 , 5, 6 into category 4(others)
df['EDUCATION'].replace({0:4,5:4,6:4}, inplace=True)
# 1=graduation school ,2= university,3=high school,4=others
```

In [18]:
```
df['EDUCATION'].value_counts()
```

Out[18]:
```
2    14030
1    10585
3     4917
4      468
Name: EDUCATION, dtype: int64
```

In [19]:
```
df['MARRIAGE'].value_counts()
```

Out[19]:
```
2    15964
1    13659
3      323
0       54
Name: MARRIAGE, dtype: int64
```

In [20]:
```
df['MARRIAGE'].replace({0 : 3},inplace = True)
df['MARRIAGE'].value_counts()

```

Out[20]:
```
2    15964
1    13659
3      377
Name: MARRIAGE, dtype: int64
```

In [21]:
```
df_colums = df[['PAY_0','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6']]
df_colums.nunique()

```

Out[21]:
```
PAY_0    11
PAY_2    11
PAY_3    11
PAY_4    11
PAY_5    10
PAY_6    10
dtype: int64
```

In [22]:
```
#Out of 23 attribute we have 9 categorical value
```

In [23]:
```
df1 = df.copy()
```

In [24]:

```
df.describe()
```

Out[24]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	167484.322667	1.603733	1.842267	1.557267	35.485500	-0.016700
std	129747.661567	0.489129	0.744494	0.521405	9.217904	1.123802
min	10000.000000	1.000000	1.000000	1.000000	21.000000	-2.000000
25%	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000
50%	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000
75%	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000
max	1000000.000000	2.000000	4.000000	3.000000	79.000000	8.000000

8 rows × 24 columns

In [25]:

```
categorical = []
numerical = []
for col in df.columns:
    if df[col].nunique()>12:
        numerical.append(col)

    else:
        categorical.append(col)
print('categorical_columns:',categorical)
print('numerical_columns:',numerical)
```

```
categorical_columns: ['SEX', 'EDUCATION', 'MARRIAGE', 'PAY_0', 'PAY_2', 'PAY_
3', 'PAY_4', 'PAY_5', 'PAY_6', 'default']
numerical_columns: ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT
3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT
3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']
```

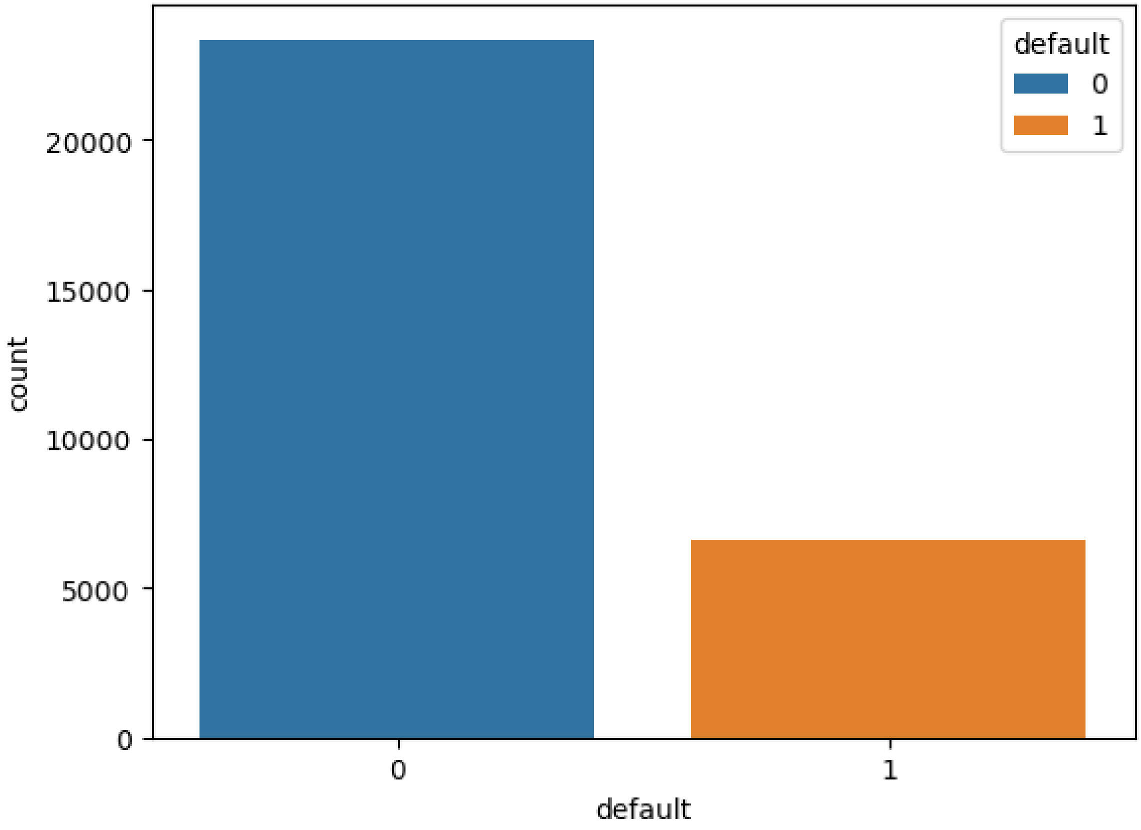
In [26]:

```
categorical.pop()
```

Out[26]: 'default'

5.Visualize the data

In [27]:
```
vis = sns.countplot(x='default',data= df,hue='default')
```

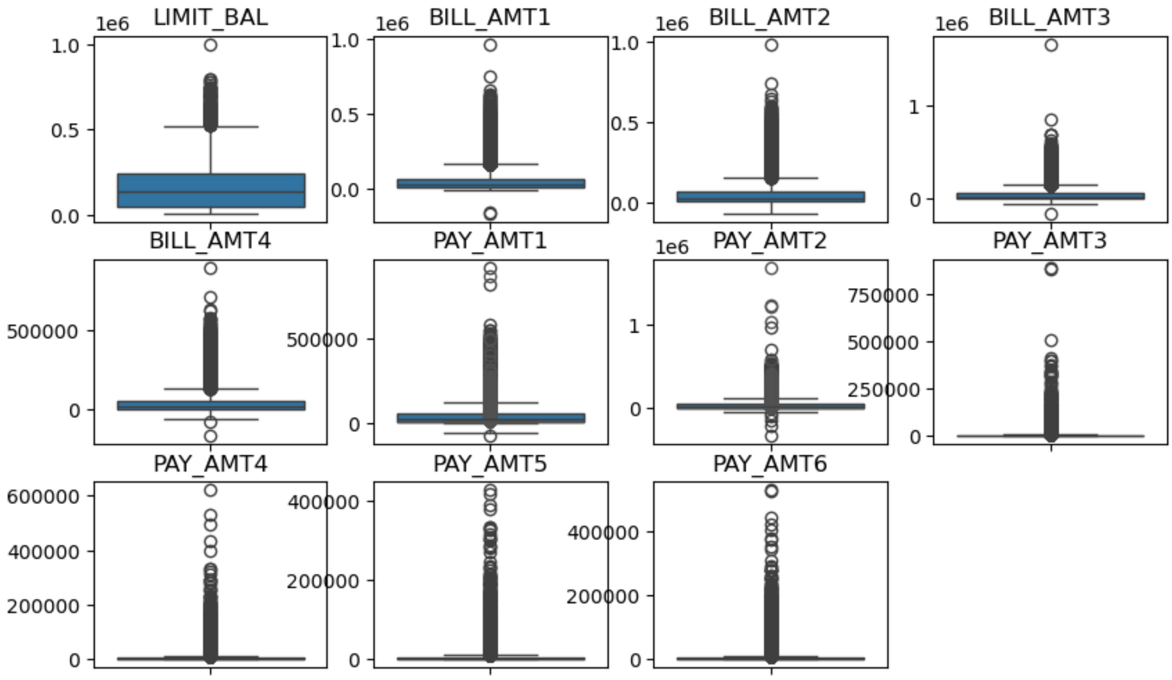


In [28]:
```
# its seems like imbalance data
```

5.1 finding outliers

In [29]:

```
plt.figure(figsize=(10,8))
outlier_col_1 = ['LIMIT_BAL','BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_
                 'BILL_AMT5', 'BILL_AMT6']
outlier_col_2 = ['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5
for i in enumerate(outlier_col_1):
    plt.subplot(4,4,i[0]+1)
    sns.boxplot(y=df[i[1]])
    plt.title(i[1])
    plt.ylabel("")

for i in enumerate(outlier_col_2):
    plt.subplot(4,4,i[0]+6)
    sns.boxplot(y=df[i[1]])
    plt.title(i[1])
    plt.ylabel("")
```

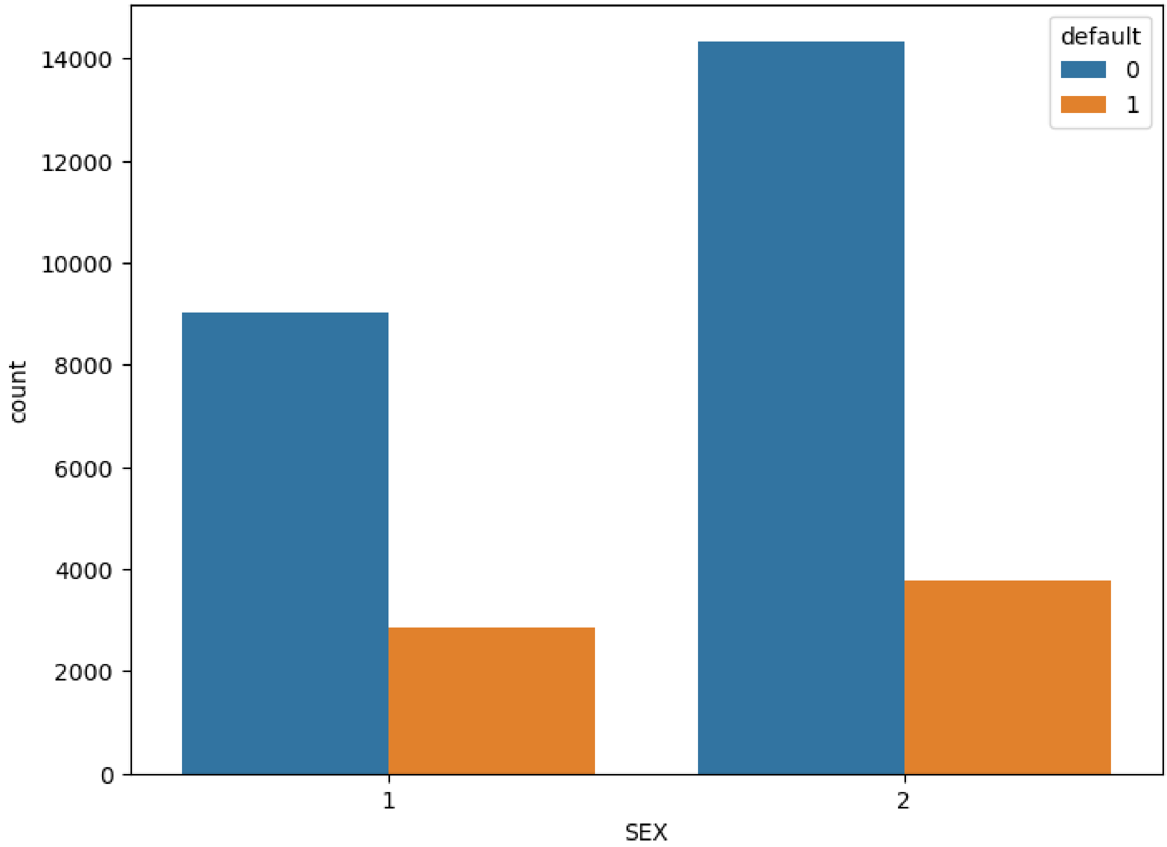


its seems dataset have Many outliers are present in the dataset, and these outliers may hold valuable information for our model.

5.2 univarate analysis

In [30]:

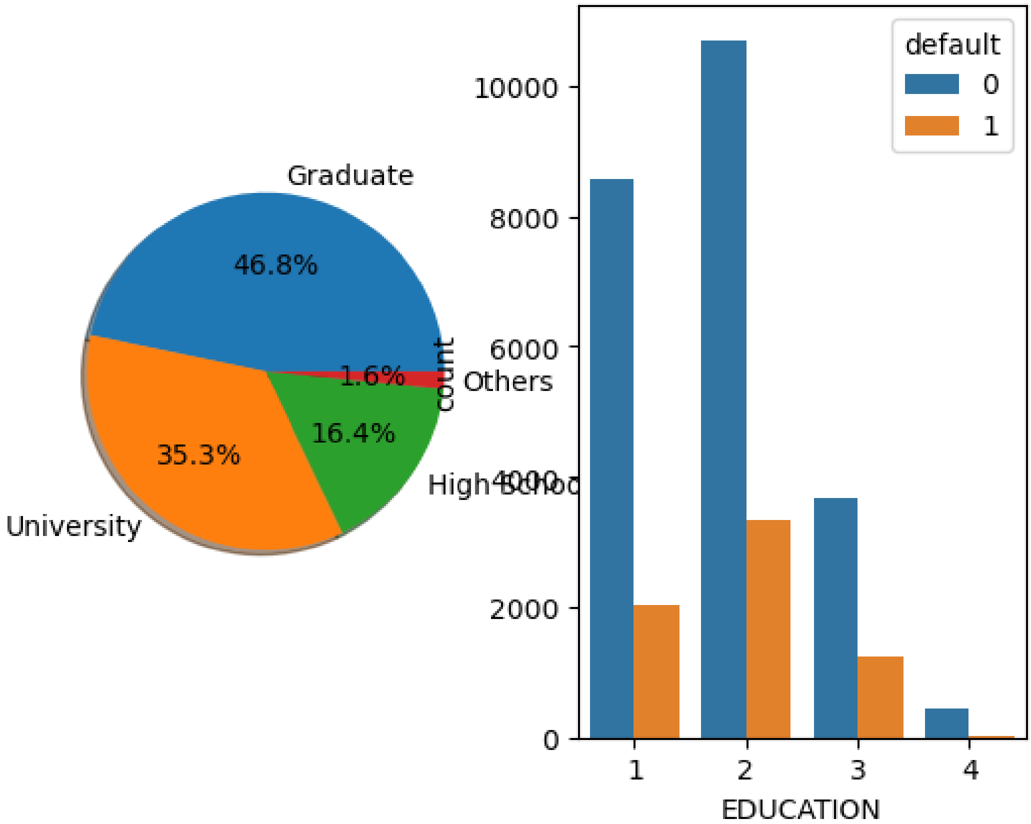
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(x="SEX", data=df, hue='default')
plt.show()
```



Compare to male female default is high

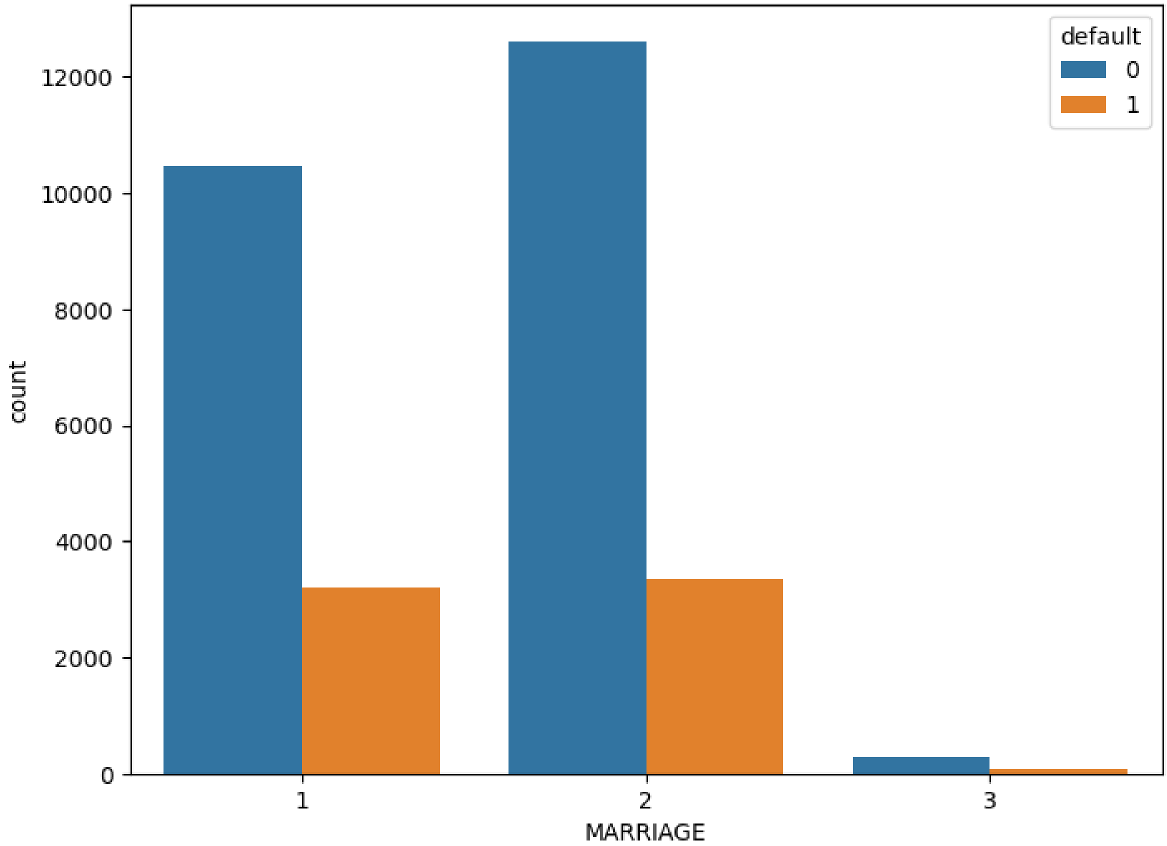
In [31]:

```python
df['EDUCATION'].value_counts()
labels = ['Graduate','University','High School','Others']
values = df['EDUCATION'].value_counts().values

f, ax = plt.subplots(1,2)

sns.countplot(x="EDUCATION", data=df, hue='default',ax=ax[1])
ax[0].pie(values, labels = labels, autopct='%1.1f%%', shadow = True)
plt.show()

```



default of credit card holder university degree holder is high compare to others. Graduate users categorized as 'Other' demonstrate a substantial 50% chance of defaulting on their credit card payments.

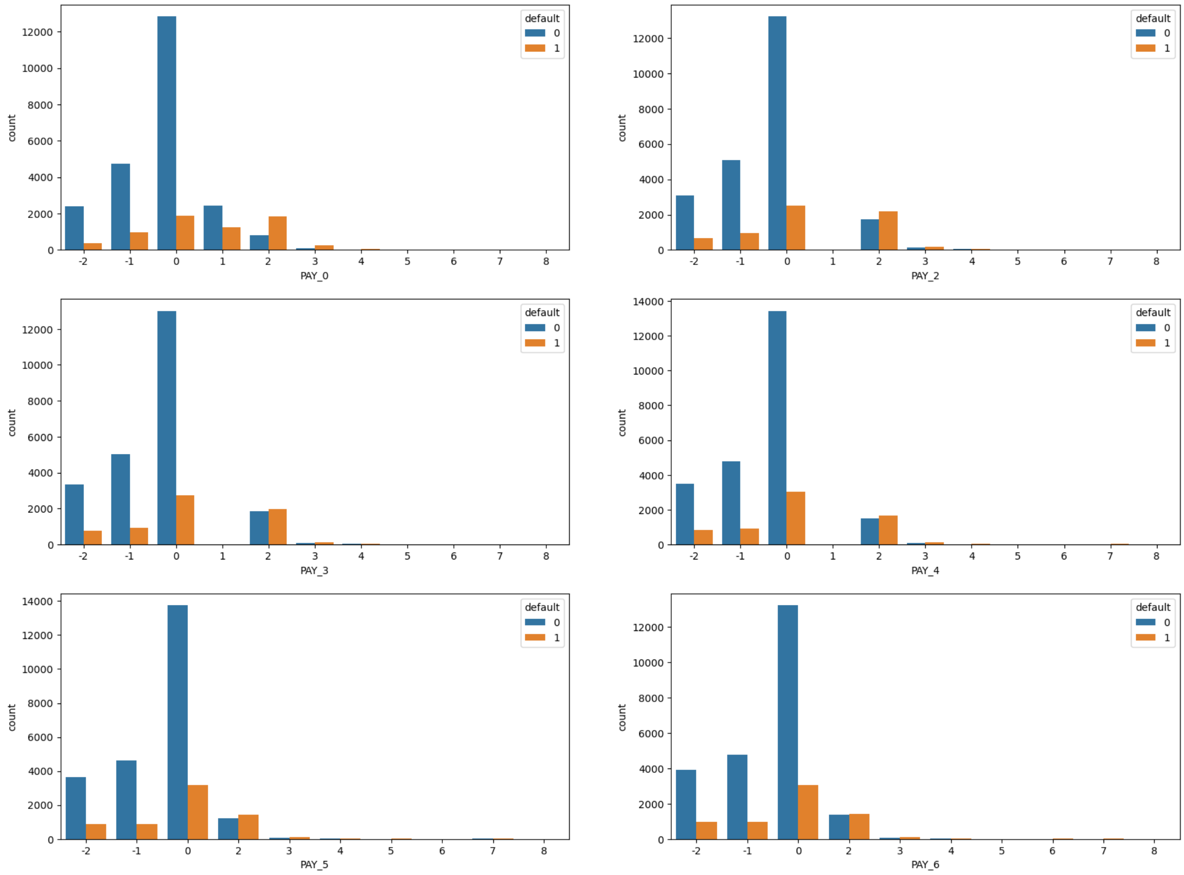
In [32]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(x="MARRIAGE", data=df, hue='default')
plt.show()
```



In [33]:
```
# almost same both married and unmarried as a defaulters
```

In [34]:

```python
figure, axes = plt.subplots(3,2,figsize=(20,15))
for i,column in enumerate(df_colums):
    row , col = divmod(i,2)
    ax=axes[row,col]
    sns.countplot(data =df, x=column,hue='default',ax=ax)
```

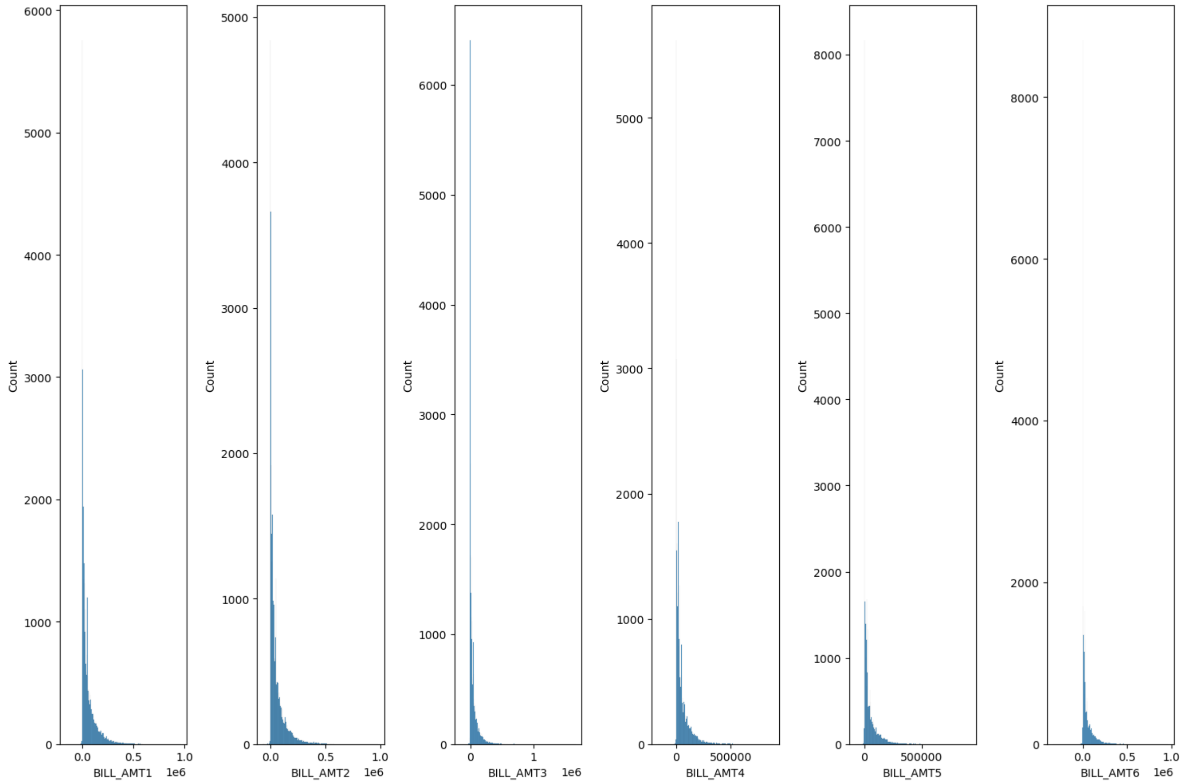


0= revolving credit ,defaulter or non default nut most of users engage with revolving credit

5.3 univarate with numerical data

In [36]:

```
Previous_bills = ['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BIL
fig, ax = plt.subplots(1,6, figsize=(15,10))
for i,bill_columns in enumerate(Previous_bills):
    sns.histplot(
        data = df, x = bill_columns,ax=ax[i]
    )
    plt.tight_layout()
```

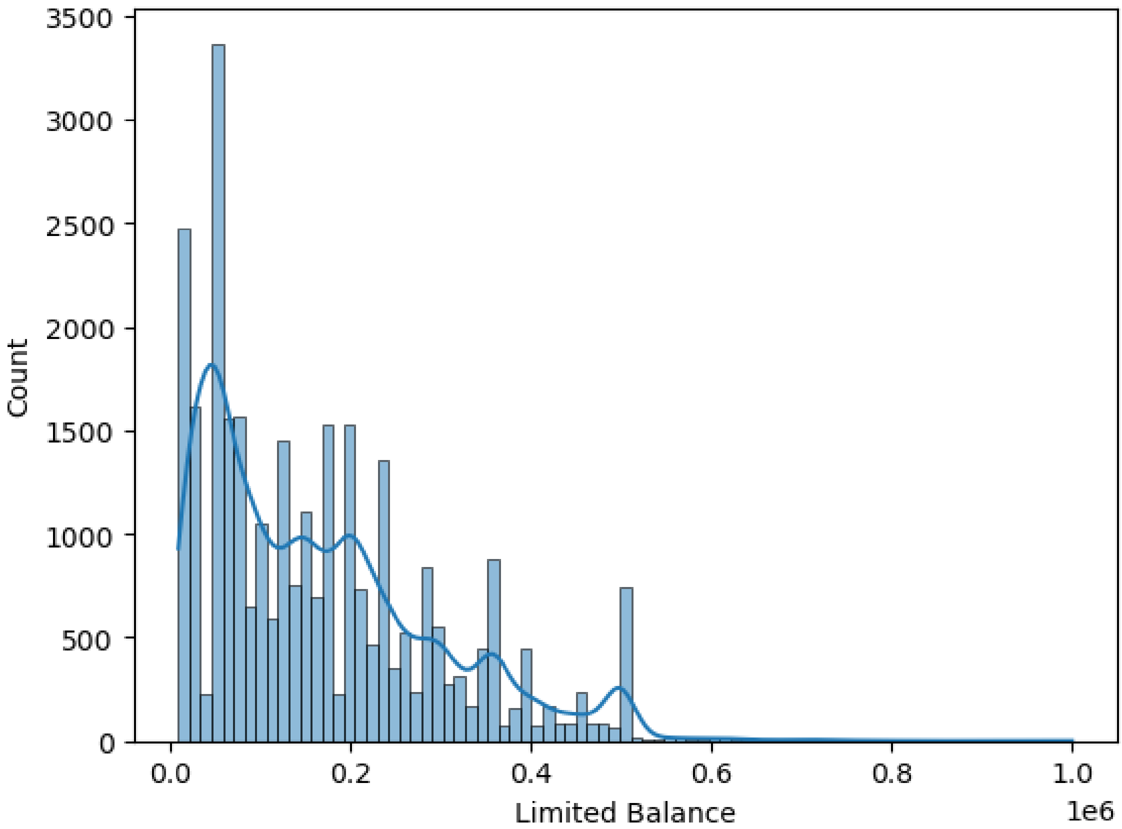


Across all months postively skewness

In [37]:

```
sns.histplot(df['LIMIT_BAL'], kde=True)
plt.xlabel('Limited Balance')
```

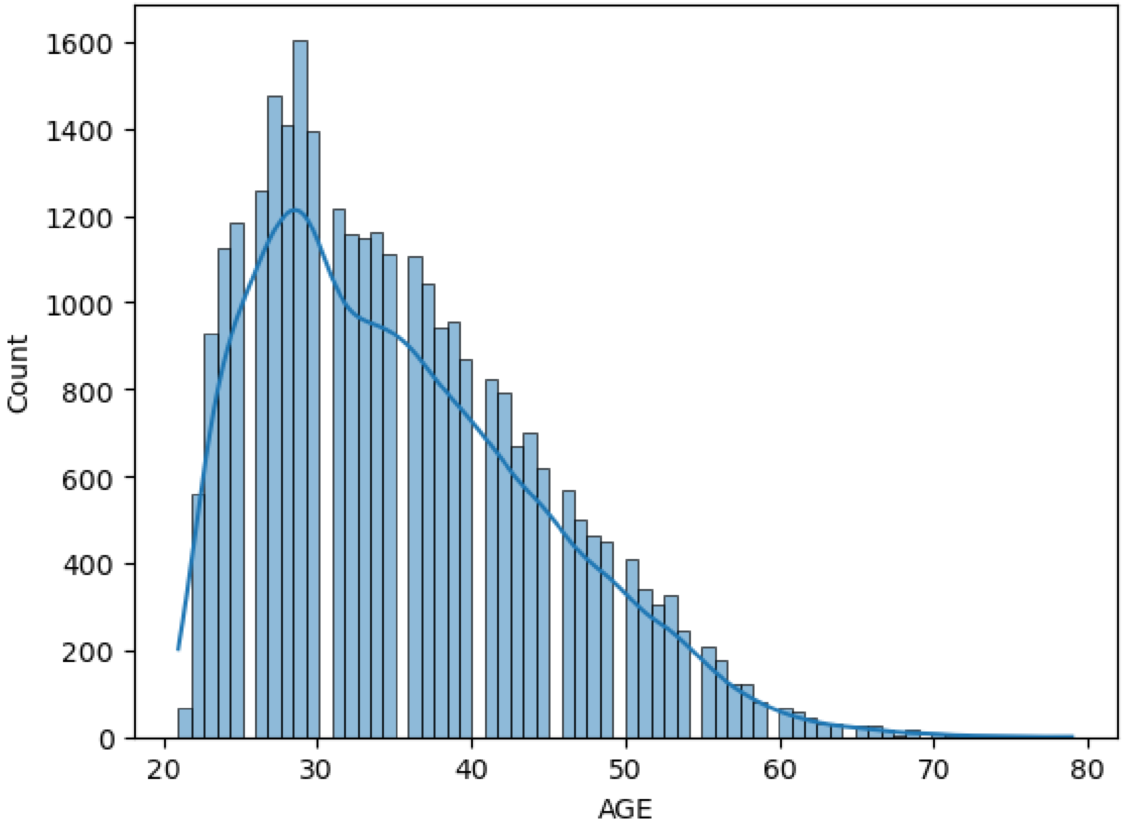
Out[37]: Text(0.5, 0, 'Limited Balance')



In [38]:

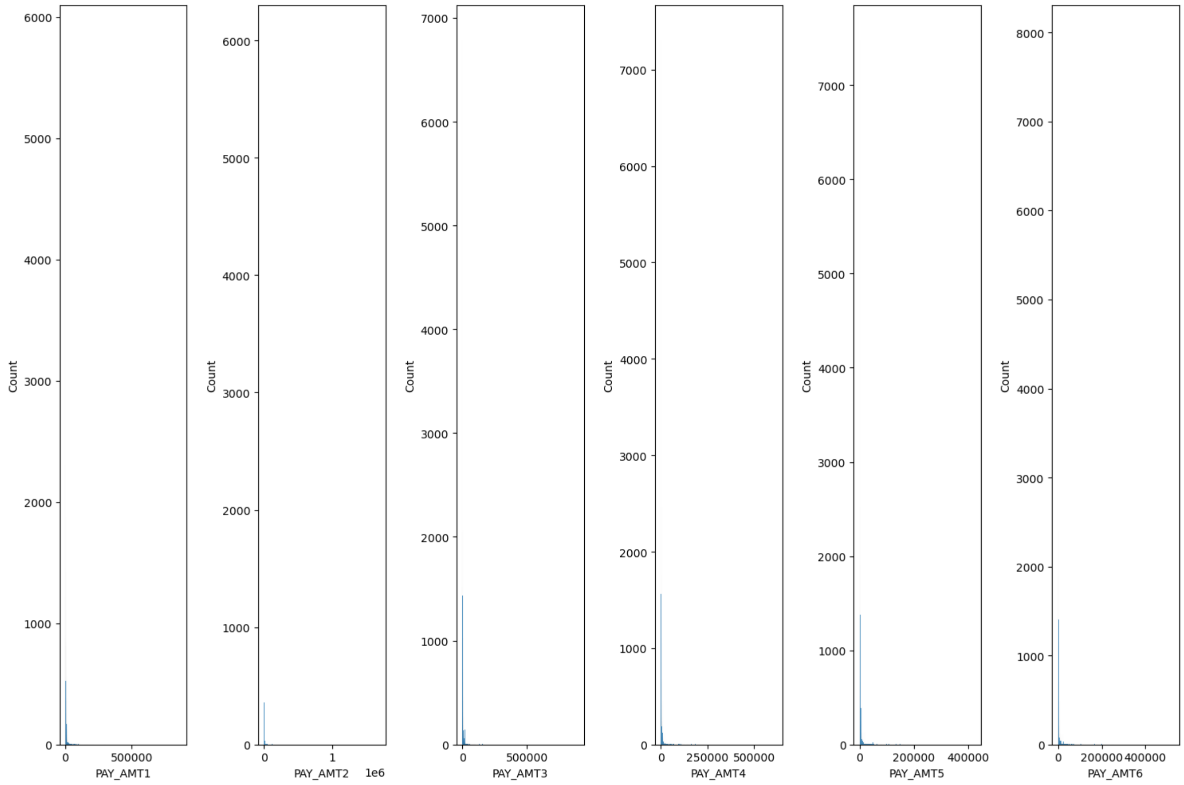
```
sns.histplot(df['AGE'], kde=True)
plt.xlabel('AGE')
```

Out[38]: Text(0.5, 0, 'AGE')



In [39]:

```
Previous_payment = ['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_A
fig, ax = plt.subplots(1,6, figsize=(15,10))
for i,payment in enumerate(Previous_payment):
    sns.histplot(
        data = df, x = payment,ax=ax[i]
    )
    plt.tight_layout()
```



6.Feature vector and Target variable

In [40]:

```

X = df.drop(['default'],axis=1)
y=df['default']
```

6.1 train and test the model

In [42]:

```
Xtrain,Xtest,ytrain,ytest = train_test_split(X,y,test_size=0.2,random_stat
```

7.Feature Scaling

In [44]:

```
scaler = StandardScaler()
Xtrain = scaler.fit_transform(Xtrain)
Xtest= scaler.transform(Xtest)
```

8.Model development

In [46]:
```
classifiers = [
    ("Decision Tree", DecisionTreeClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("SVM", SVC()),
    ("KNN", KNeighborsClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ('XGBoost' , XGBClassifier()),
    ('LGBM' , LGBMClassifier())
]

# Create an empty DataFrame to store results
results_df = pd.DataFrame(columns=["Classifier", "Accuracy"])
for method,classifier in classifiers:
    model =classifier.fit(Xtrain,ytrain)
    ypred = model.predict(Xtest)
    accuracy = accuracy_score(ytest, ypred)
    # Append results to DataFrame
    results_df = results_df.append({"Classifier": method, "Accuracy": accu
print(results_df)
```

```
[LightGBM] [Info] Number of positive: 5339, number of negative: 18661
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.011733 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3266
[LightGBM] [Info] Number of data points in the train set: 24000, number of us
ed features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.222458 -> initscore=-1.2513
97
[LightGBM] [Info] Start training from score -1.251397
            Classifier  Accuracy
0        Decision Tree  0.731333
1        Random Forest  0.822333
2                  SVM  0.825667
3                  KNN  0.789667
4  Logistic Regression  0.819167
5              XGBoost  0.818333
6                 LGBM  0.828000
```

#LGBM algorthim model is good accuracy 82.8% with imbalanced data #svm algorthrim model is 82.5% #randomforest model 82.2%

9.Dealing with imbalanced data

In [47]:
```
df_smote = df.copy()
```

In [48]:
```
from imblearn.over_sampling import SMOTE
x = df_smote.drop(['default'],axis=1)
y=df_smote['default']
```

In [49]:
```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.2,random_stat
```

In [50]:
```
scaler = StandardScaler()
xtrain = scaler.fit_transform(xtrain)
xtest= scaler.transform(xtest)
```

In [57]:
```
smote = SMOTE(random_state=0)
x_sample,y_sample = smote.fit_resample(xtrain,ytrain)

```

In [58]:
```
ytest.shape,xtest.shape
```

Out[58]:
```
((6000,), (6000, 23))
```

In [59]:
```
x_sample.shape,y_sample.shape   # after sampling the data shape changed
```

Out[59]:
```
((37322, 23), (37322,))
```

In [60]:
```
y_sample.value_counts() # convert imbalace data into balanced data using SM
```

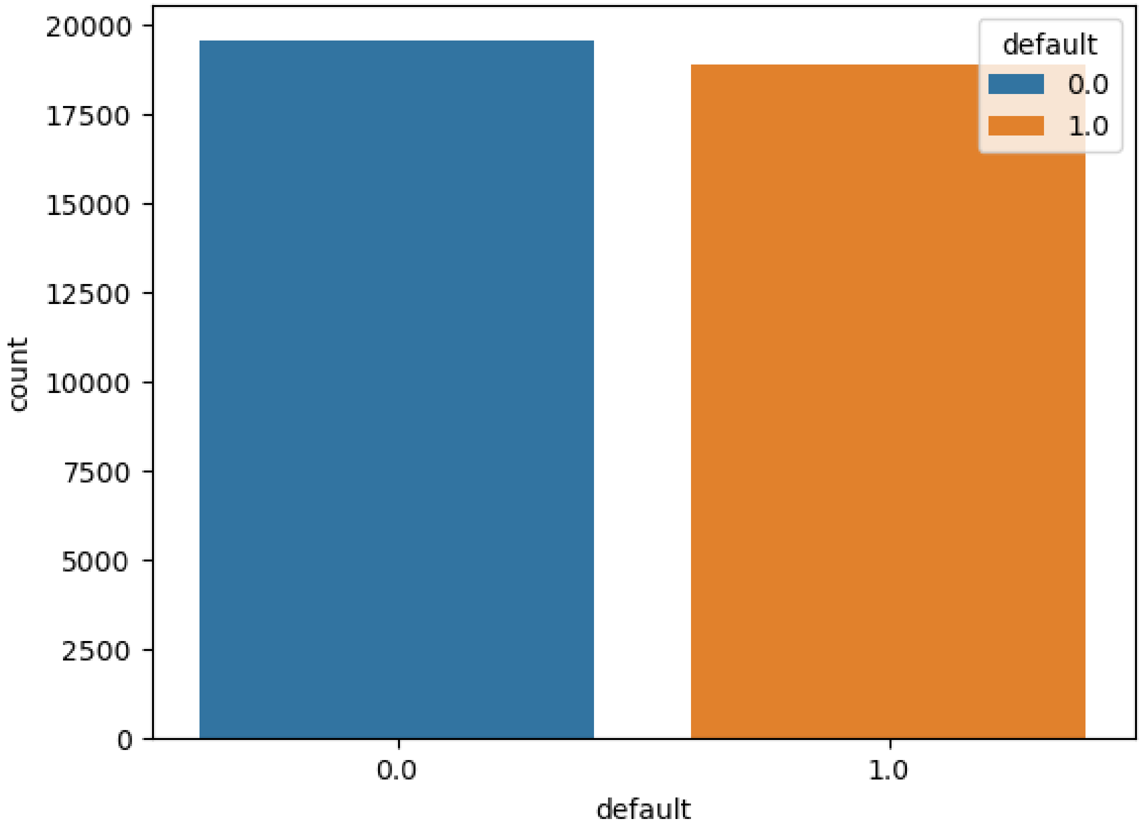
Out[60]:
```
0    18661
1    18661
Name: default, dtype: int64
```

In [70]:
```
# Create new DataFrames with the resampled training set and the original te
train_resampled_df = pd.DataFrame(x_sample, columns=x.columns)
train_resampled_df['default'] = y_sample
test_df = pd.DataFrame(xtest, columns=x.columns)
test_df['default'] = ytest
```

In [71]:
```

# Combine the resampled training DataFrame with the original test DataFrame
combined_df = pd.concat([train_resampled_df, test_df], ignore_index=True)

# Save the combined DataFrame as a CSV file
combined_df.to_csv('combined_data.csv', index=False)

```

In [72]:
```
df_smote1 = combined_df.copy()
```

In [73]:
```
sns.countplot(x='default',data=df_smote1,hue='default')  #see data balanced
```
Out[73]: <Axes: xlabel='default', ylabel='count'>



10.Model development with balance data

In [74]:

```
classifiers = [
    ("Decision Tree", DecisionTreeClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("SVM", SVC()),
    ("KNN", KNeighborsClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ('XGBoost' , XGBClassifier()),
    ('LGBM' , LGBMClassifier())
]

# Create an empty DataFrame to store results
results_df = pd.DataFrame(columns=["Classifier", "Accuracy"])
for method,classifier in classifiers:
    model =classifier.fit(x_sample,y_sample)
    ypred_smote = model.predict(xtest)
    accuracy = accuracy_score(ytest, ypred_smote)
    # Append results to DataFrame
    results_df = results_df.append({"Classifier": method, "Accuracy": accu
print(results_df)
```

```
[LightGBM] [Info] Number of positive: 18661, number of negative: 18661
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of tes
ting was 0.017742 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 5735
[LightGBM] [Info] Number of data points in the train set: 37322, number of us
ed features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.00000
0
            Classifier  Accuracy
0        Decision Tree  0.703167
1        Random Forest  0.802000
2                  SVM  0.780167
3                  KNN  0.652000
4  Logistic Regression  0.674667
5              XGBoost  0.811667
6                 LGBM  0.824333
```

In [66]:

```
# here LGBM is Good accuracy  82.4%,XGBoost is 81.1% and RandomForest 80.2
```

10.1 XGBOOST

In [76]:

```
# import packages for hyperparameters tuning
from hyperopt import STATUS_OK, Trials, fmin, hp, tpe
space={'max_depth': hp.quniform("max_depth", 3, 18, 1),
        'gamma': hp.uniform ('gamma', 1,9),
        'reg_alpha' : hp.quniform('reg_alpha', 40,180,1),
        'reg_lambda' : hp.uniform('reg_lambda', 0,1),
        'colsample_bytree' : hp.uniform('colsample_bytree', 0.5,1),
        'min_child_weight' : hp.quniform('min_child_weight', 0, 10, 1),
        'n_estimators': 180,
        'seed': 0
    }
```

In [77]:

```python
import xgboost as xgb


def objective(space):
    clf=xgb.XGBClassifier(
                    n_estimators =space['n_estimators'], max_depth = int(s
                    reg_alpha = int(space['reg_alpha']),min_child_weight=i
                    colsample_bytree=int(space['colsample_bytree']))

    evaluation = [( x_sample, y_sample), ( xtest, ytest)]

    clf.fit(x_sample, y_sample,
            eval_set=evaluation, eval_metric="auc",
            early_stopping_rounds=10,verbose=False)


    pred = clf.predict(xtest)
    accuracy = accuracy_score(ytest, pred)
    print ("SCORE:", accuracy)
    return {'loss': -accuracy, 'status': STATUS_OK }
```

In [80]:

```python
trials = Trials()

best_hyperparams = fmin(fn = objective,
                        space = space,
                        algo = tpe.suggest,
                        max_evals = 100,
                        trials = trials)

```

```
0.8018333333333333
SCORE:
0.8
SCORE:
0.7976666666666666
SCORE:
0.8016666666666666
SCORE:
0.7941666666666667
SCORE:
0.791
SCORE:
0.7891666666666667
SCORE:
0.796
SCORE:
0.8
SCORE:
0.7975
100%|██████████████████████████████| 100/100 [01:46<00:00,
```

In [81]:

```
print("The best hyperparameters are : ","\n")
print(best_hyperparams)
best_hyperparams['max_depth'] = int(best_hyperparams['max_depth'])
```

```
The best hyperparameters are :

{'colsample_bytree': 0.7125085163110019, 'gamma': 2.916107216242168, 'max_dep
th': 13.0, 'min_child_weight': 4.0, 'reg_alpha': 51.0, 'reg_lambda': 0.238452
0699705056}
```

In [82]:

```
xgb = XGBClassifier(**best_hyperparams)
xgb.fit(x_sample,y_sample)
ypredtion=xgb.predict(xtest)
print('Final accuracy of the model is {}%'.format(accuracy_score(ytest,ypr
print('Classification Report \n{}'.format(classification_report(ytest,ypre
print('Confusion Matrix \n{}'.format(confusion_matrix(ytest,ypredtion)))
```

```
Final accuracy of the model is 79.4%
Classification Report
              precision    recall  f1-score   support

           0       0.87      0.87      0.87      4703
           1       0.52      0.53      0.53      1297

    accuracy                           0.79      6000
   macro avg       0.70      0.70      0.70      6000
weighted avg       0.80      0.79      0.79      6000

Confusion Matrix
[[4076  627]
 [ 609  688]]
```

10.2 RandomForest

In [85]:

```
from sklearn.model_selection import RandomizedSearchCV
rf_classifier = RandomForestClassifier()

# Define the hyperparameter search space
param_dist = {
    'n_estimators': [int(x) for x in range(10, 200)],
    'max_features': ['auto', 'sqrt', 'log2', None],
    'max_depth': [int(x) for x in range(1, 20)],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],

}

# Perform RandomizedSearchCV
random_search = RandomizedSearchCV(
    rf_classifier,
    param_distributions=param_dist,
    n_iter=20,  # Number of parameter settings that are sampled
    cv=5,       # Number of cross-validation folds
    n_jobs=-1,  # Use all available CPU cores
    verbose=1,  # Print progress
    random_state=0
)

# Fit the model to the training data
random_search.fit(x_sample, y_sample)
best_parameters = random_search.best_params_

```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

In [86]:

```
rf = RandomForestClassifier(**best_parameters)
rf.fit(x_sample,y_sample)
y_predrf = rf.predict(xtest)
print('Final accuracy of the model is {}%'.format(accuracy_score(ytest,y_p
print('Classification Report \n{}'.format(classification_report(ytest,y_pr
print('Confusion Matrix \n{}'.format(confusion_matrix(ytest,y_predrf)))
```

```
Final accuracy of the model is 79.7%
Classification Report
              precision    recall  f1-score   support

           0       0.86      0.88      0.87      4703
           1       0.53      0.48      0.51      1297

    accuracy                           0.80      6000
   macro avg       0.70      0.68      0.69      6000
weighted avg       0.79      0.80      0.79      6000

Confusion Matrix
[[4159  544]
 [ 674  623]]
```

10.3 LGBM

In [ ]:

```
# import packages for hyperparameters tuning
from hyperopt import STATUS_OK, Trials, fmin, hp, tpe
space={'max_depth': hp.uniform("max_depth", 4, 18),
        'learning_rate': hp.loguniform ('learning_rate', np.log(0.01), np.
       'num_leaves': hp.uniform('num_leaves', 20, 150),
        'min_child_samples': hp.uniform('min_child_samples', 5, 100),
          'colsample_bytree' : hp.uniform('colsample_bytree', 0.5,0.7),
          'subsample':hp.uniform('subsample', 0.5,0.9),
          'subsample_freq': hp.uniform('subsample_freq', 1,2),
          'reg_alpha' : hp.uniform('reg_alpha', 40,180),
          'reg_lambda' : hp.uniform('reg_lambda', 0,1),
           'max_bin': hp.uniform('max_bin', 75,100),
           'feature_fraction': hp.uniform('feature_fraction', 0.5, 1.0),
         'bagging_fraction': hp.uniform('bagging_fraction', 0.5, 1.0),
         'bagging_freq': hp.uniform('bagging_freq', 1, 10) ,
          'nthread': 8,
          'verbose': 0

    }
```

In [94]:

```
import lightgbm as lgb


def objective(space):

    clf=lgb.LGBMClassifier(
                    num_leaves =int(space['num_leaves']), max_depth = int(
                   objective ='binary',min_child_samples=int(space['min_ch
                    colsample_bytree=space['colsample_bytree'],subsample =
                    subsample_freq = space['subsample_freq'],bagging_freq=
                    n_estimators=150)
    evaluation = [( x_sample, y_sample), ( xtest, ytest)]

    clf.fit(x_sample, y_sample,
            eval_set=evaluation, eval_metric="auc")
    pred = clf.predict(xtest)
    accuracy = accuracy_score(ytest, pred)
    print ("SCORE:", accuracy)
    return {'loss': -accuracy, 'status': STATUS_OK }
```

In [95]:

```python
trials = Trials()

best_hyperparams = fmin(fn = objective,
                        space = space,
                        algo = tpe.suggest,
                        max_evals = 100,
                        trials = trials)
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] bagging_freq is set=2, subsample_freq=1.52384056321214
57 will be ignored. Current value: bagging_freq=2
SCORE:
0.814
[LightGBM] [Warning] bagging_freq is set=1, subsample_freq=1.17992371880536
97 will be ignored. Current value: bagging_freq=1
[LightGBM] [Warning] bagging_freq is set=1, subsample_freq=1.17992371880536
97 will be ignored. Current value: bagging_freq=1
[LightGBM] [Info] Number of positive: 18661, number of negative: 18661
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of t
esting was 0.013212 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 5735
```

In [103]:

```python
print("The best hyperparameters are : ","\n")
print(best_hyperparams)
best_hyperparams['num_leaves'] = int(best_hyperparams['num_leaves'])
best_hyperparams['max_depth'] = int(best_hyperparams['max_depth'])
best_hyperparams['bagging_freq'] = int(best_hyperparams['bagging_freq'])
best_hyperparams['max_bin'] = int(best_hyperparams['max_bin'])
```

```
The best hyperparameters are :

{'bagging_fraction': 0.6828580064307468, 'bagging_freq': 2, 'colsample_bytre
e': 0.5130684539621747, 'feature_fraction': 0.9684276671090114, 'learning_rat
e': 0.015223939585351605, 'max_bin': 78.63726078924715, 'max_depth': 14, 'min
_child_samples': 24.75726287535893, 'num_leaves': 24, 'reg_alpha': 129.651697
35764212, 'reg_lambda': 0.5362968135570422, 'subsample': 0.8134663323186887,
'subsample_freq': 1.3343624216548728}
```

In [104]:

```
lgb = LGBMClassifier(**best_hyperparams,min_data_in_leaf=int(24.7))
lgb.fit(x_sample,y_sample)
ypredtion=lgb.predict(xtest)
print('Final accuracy of the model is {}%'.format(accuracy_score(ytest,ypr
print('Classification Report \n{}'.format(classification_report(ytest,ypre
print('Confusion Matrix \n{}'.format(confusion_matrix(ytest,ypredtion)))
```

```
[LightGBM] [Warning] min_data_in_leaf is set=24, min_child_samples=24.7572628
7535893 will be ignored. Current value: min_data_in_leaf=24
[LightGBM] [Warning] feature_fraction is set=0.9684276671090114, colsample_by
tree=0.5130684539621747 will be ignored. Current value: feature_fraction=0.96
84276671090114
[LightGBM] [Warning] bagging_fraction is set=0.6828580064307468, subsample=0.
8134663323186887 will be ignored. Current value: bagging_fraction=0.682858006
4307468
[LightGBM] [Warning] bagging_freq is set=2, subsample_freq=1.3343624216548728
will be ignored. Current value: bagging_freq=2
[LightGBM] [Warning] min_data_in_leaf is set=24, min_child_samples=24.7572628
7535893 will be ignored. Current value: min_data_in_leaf=24
[LightGBM] [Warning] feature_fraction is set=0.9684276671090114, colsample_by
tree=0.5130684539621747 will be ignored. Current value: feature_fraction=0.96
84276671090114
[LightGBM] [Warning] bagging_fraction is set=0.6828580064307468, subsample=0.
8134663323186887 will be ignored. Current value: bagging_fraction=0.682858006
4307468
[LightGBM] [Warning] bagging_freq is set=2, subsample_freq=1.3343624216548728
will be ignored. Current value: bagging_freq=2
[LightGBM] [Info] Number of positive: 18661, number of negative: 18661
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of tes
ting was 0.008318 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1730
[LightGBM] [Info] Number of data points in the train set: 37322, number of us
ed features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.00000
0
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] min_data_in_leaf is set=24, min_child_samples=24.7572628
7535893 will be ignored. Current value: min_data_in_leaf=24
[LightGBM] [Warning] feature_fraction is set=0.9684276671090114, colsample_by
tree=0.5130684539621747 will be ignored. Current value: feature_fraction=0.96
84276671090114
[LightGBM] [Warning] bagging_fraction is set=0.6828580064307468, subsample=0.
8134663323186887 will be ignored. Current value: bagging_fraction=0.682858006
4307468
[LightGBM] [Warning] bagging_freq is set=2, subsample_freq=1.3343624216548728
will be ignored. Current value: bagging_freq=2
Final accuracy of the model is 77.21666666666667%
Classification Report
              precision    recall  f1-score   support

           0       0.88      0.83      0.85      4703
           1       0.48      0.58      0.52      1297

    accuracy                           0.77      6000
   macro avg       0.68      0.70      0.69      6000
weighted avg       0.79      0.77      0.78      6000
```

```
Confusion Matrix
[[3880  823]
 [ 544  753]]
```

11.checking overfitting and underfitting

In [91]:
```python
# print the scores on training and test set

print('Training set score: {:.4f}'.format(rf.score(x_sample, y_sample)))

print('Test set score: {:.4f}'.format(rf.score(xtest, ytest)))
```

```
Training set score: 0.9473
Test set score: 0.7970
```

#The training set score is significantly higher than the test set score. This suggests that the model is likely overfitting.

12.K-Fold cross validation

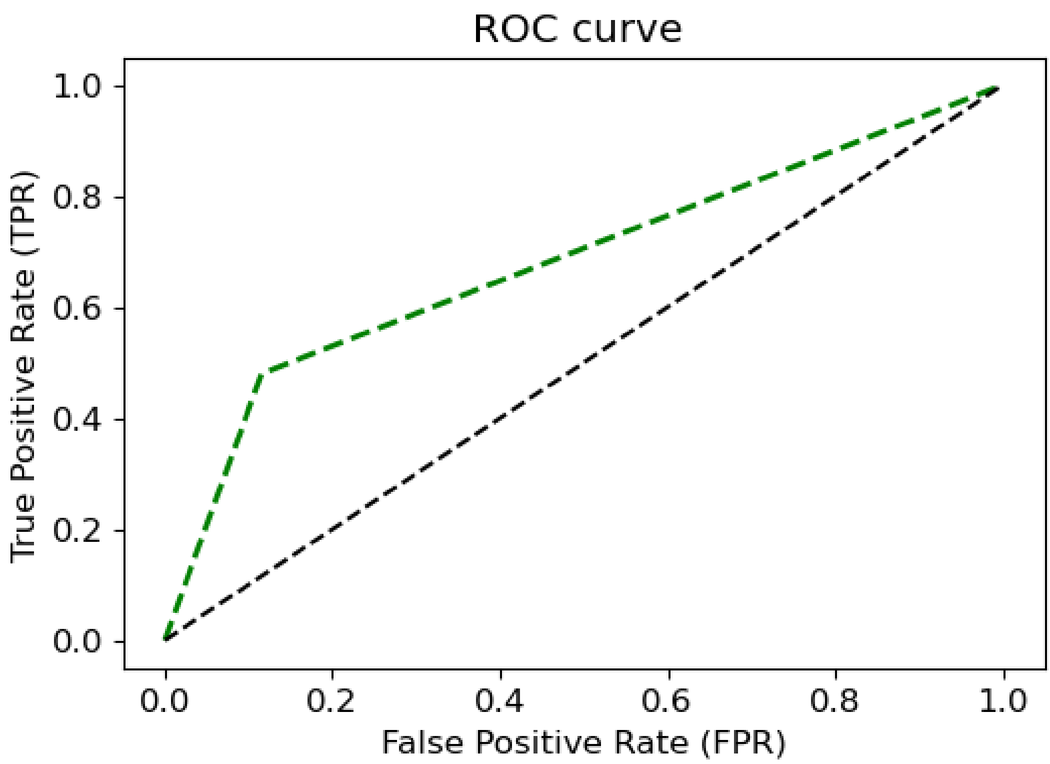
In [93]:
```python
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = rf, X = x_sample,y = y_sample, cv
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 83.68 %
Standard Deviation: 6.81 %
```

13.ROC-AUC

In [101]:

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(ytest, y_predrf)

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linestyle='--', color = 'green',linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.rcParams['font.size'] = 12

plt.title('ROC curve ')

plt.xlabel('False Positive Rate (FPR)')

plt.ylabel('True Positive Rate (TPR)')

plt.show()

```



In [98]:

```
from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(ytest, y_predrf)

print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.6823
```

In [99]:

```

from sklearn.model_selection import cross_val_score

Cross_validated_ROC_AUC = cross_val_score(rf, x_sample, y_sample, cv=5, sc

print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))
```

```
Cross validated ROC AUC : 0.9097
```

14.Conclution

```
In conclusion, In this project, I build a all classifation models to classify thecredit card defaulter.
we get 3 better models RandomForest,XGBoost,LGBM .
In XGBoost hyperparameter tunning using HyperOPT ,the model accuracy is 79.4%
In RandomForest the model accuracy is 79.7% using the model tunning technique RandomizedSearchCV
In LGBM hyperparameter tunning using HyperOPT, the model acuracy is 77.2%
Finally go with RandomForest Classifer
The training-set accuracy score is 0.9473 while the test-set accuracy to be 0.797. So, the model is overfitting.
If we look at all the 10 scores produced by the 10-fold cross-validation, we can also conclude that there is a relatively low variance in the accuracy between folds, ranging from 100% accuracy to 83.68% accuracy.
ROC AUC score is 68.23%
Cross validated ROC AUC score is 90.97%
```

GotoTOP

In [ ]:

```

```