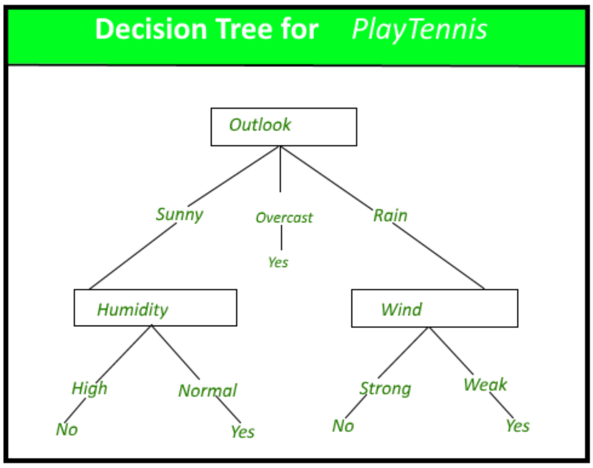
Experiment No. 3
Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:07/08/2023
Date of Submission:20/08/2023

Aim: Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model.

Objective: To perform various feature engineering tasks, apply Decision Tree Algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score. Improve the performance by performing different data engineering and feature engineering tasks.

Theory:

Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.



Dataset:

Predict whether income exceeds $50K/yr based on census data. Also known as "Adult" dataset.

Attribute Information:

Listing of attributes:

>50K, <=50K.

age: continuous.

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.

education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.

capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

Code:

Conclusion:

1. Dealing with Categorical Attributes during Data Pre-processing: 1. Managing Categorical Attributes During Data Pre-processing:

The values of the following categories columns are converted into unique number labels using

label encoding. This numerical representation is required by many machine learning methods

that need numerical input data. Encoding these categorical characteristics prepares the data for

machine learning model training.

1. The term "workclass" refers to their job position.

2. The word "education" indicates their greatest degree of schooling.

3. "marital-status" indicates their marital status.

4. "occupation" displays their work functions.

5. "relationship" describes their familial situation.

6. "race" usually refers to their racial heritage.

7. The word "sex" denotes gender.

8. "native-country" frequently refers to their country of origin or citizenship.

Certain columns are eliminated during data pre-processing in the code you gave.

2. The following columns are specifically removed:

1. Channel: This column is removed with the data.drop(labels=(['Channel', Region')),axis=1,inplace=True) function. The Channel column appears to have been deleted from the dataset.

2. Region: The Region column, like the Channel column, is discarded using the same line of code. This column is also deleted from the dataset. Hyperparameter Tuning:

The Decision Tree classifier is hyperparameter tuned in this code:

The Decision Tree classifier is built with a maximum depth of 5 specified: DecisionTree Classifier(max_depth-5). Because it influences the depth of the tree, this is a type of hyperparameter adjustment. This code sample, however, does not show a comprehensive hyperparameter tweaking procedure. In reality, more extensive approaches such as grid search or random search can be used to find the optimum hyperparameters. Only the max depth is changed here.

3. Evaluation Metrics for Classification Models Confusion Matrix: It accurately identified 4310 cases as negative (0) and 767 instances as

positive (1), but it also projected 243 positives and 713 negatives incorrectly. Performance Metrics: The accuracy for positive predictions (1) is 0.76 lower than in Model I. while the recall is 0.52 higher. This model has an F1-score of 0.62. This model has an overall accuracy of 0.84.

	precision	recall	f1-score	support
0	0.85	0.95	0.90	7475
1	0.76	0.47	0.58	2294
Accuracy			0.84	9769
macro avg	0.81	0.71	0.74	9769
weighted avg	0.83	0.84	0.83	9769

```
# Import libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# To ignore warning messages
import warnings
warnings.filterwarnings('ignore')
```

```
df=pd.read_csv("/content/adult.csv")
df.head(10)
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	ca
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	
5	34	Private	216864	HS-grad	9	Divorced	Other-service	Unmarried	White	Female	0	
6	38	Private	150601	10th	6	Separated	Adm-clerical	Unmarried	White	Male	0	
7	74	State-gov	88638	Doctorate	16	Never-married	Prof-specialty	Other-relative	White	Female	0	
8	68	Federal-gov	422013	HS-grad	9	Divorced	Prof-specialty	Not-in-family	White	Female	0	
9	41	Private	70037	Some-college	10	Never-married	Craft-repair	Unmarried	White	Male	0	

```
print ("Rows     : " ,df.shape[0])
print ("Columns  : " ,df.shape[1])
print ("\nFeatures : \n" ,df.columns.tolist())
print ("\nMissing values : ", df.isnull().sum().values.sum())
print ("\nUnique values : \n",df.nunique())
```

```
Rows     :  32561
Columns  :  15

Features :
 ['age', 'workclass', 'fnlwgt', 'education', 'education.num', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'capit

Missing values :   0

Unique values :
 age                  73
workclass             9
fnlwgt            21648
education            16
education.num        16
marital.status        7
occupation           15
relationship          6
race                  5
sex                   2
capital.gain        119
capital.loss         92
hours.per.week       94
native.country       42
income                2
dtype: int64
```

Preprocessing

```
# encode categorical variables using label Encoder
from sklearn import preprocessing
df_categorical = df.select_dtypes(include=['object'])
df_categorical.head()
```

	workclass	education	marital.status	occupation	relationship	race	sex	native.country	income
0	?	HS-grad	Widowed	?	Not-in-family	White	Female	United-States	<=50K
1	Private	HS-grad	Widowed	Exec-managerial	Not-in-family	White	Female	United-States	<=50K

```
# apply label encoder to df_categorical
le = preprocessing.LabelEncoder()
df_categorical = df_categorical.apply(le.fit_transform)
df_categorical.head()
```

	workclass	education	marital.status	occupation	relationship	race	sex	native.country	income
0	0	11	6	0	1	4	0	39	0
1	4	11	6	4	1	4	0	39	0
2	0	15	6	0	4	2	0	39	0
3	4	5	0	7	4	4	0	39	0
4	4	15	5	10	3	4	0	39	0

```
# Next, Concatenate df_categorical dataframe with original df (dataframe)
# Drop earlier duplicate columns which had categorical values
df = df.drop(df_categorical.columns,axis=1)
df = pd.concat([df,df_categorical],axis=1)
df.head()
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass	education	marital.status	occupation	relation
0	90	77053	9	0	4356	40	0	11	6	0	
1	82	132870	9	0	4356	18	4	11	6	4	
2	66	186061	10	0	4356	40	0	15	6	0	
3	54	140359	4	0	3900	40	4	5	0	7	
4	41	264663	10	0	3900	40	4	15	5	10	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   fnlwgt          32561 non-null  int64
 2   education.num   32561 non-null  int64
 3   capital.gain    32561 non-null  int64
 4   capital.loss    32561 non-null  int64
 5   hours.per.week  32561 non-null  int64
 6   workclass       32561 non-null  int64
 7   education       32561 non-null  int64
 8   marital.status  32561 non-null  int64
 9   occupation      32561 non-null  int64
 10  relationship    32561 non-null  int64
 11  race            32561 non-null  int64
 12  sex             32561 non-null  int64
 13  native.country  32561 non-null  int64
 14  income          32561 non-null  int64
dtypes: int64(15)
memory usage: 3.7 MB
```

```
# convert target variable income to categorical
df['income'] = df['income'].astype('category')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   fnlwgt          32561 non-null  int64
 2   education.num   32561 non-null  int64
 3   capital.gain    32561 non-null  int64
 4   capital.loss    32561 non-null  int64
 5   hours.per.week  32561 non-null  int64
 6   workclass       32561 non-null  int64
 7   education       32561 non-null  int64
 8   marital.status  32561 non-null  int64
 9   occupation      32561 non-null  int64
```

```
 10  relationship    32561 non-null  int64
 11  race            32561 non-null  int64
 12  sex             32561 non-null  int64
 13  native.country  32561 non-null  int64
 14  income          32561 non-null category
dtypes: category(1), int64(14)
memory usage: 3.5 MB
```

Model Building

```
from sklearn.model_selection import train_test_split
X = df.drop('income',axis=1)
y = df['income']
```

```
X.head(3)
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass	education	marital.status	occupation	relation
0	90	77053	9	0	4356	40	0	11	6	0	
1	82	132870	9	0	4356	18	4	11	6	4	
2	66	186061	10	0	4356	40	0	15	6	0	

```
y.head(3)
```

```
0    0
1    0
2    0
Name: income, dtype: category
Categories (2, int64): [0, 1]
```

```
#train test split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,random_state=99)
X_train.head()
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass	education	marital.status	occupation	rela
5728	30	117963	13	0	0	40	4	9	2	1	
10700	18	80564	9	0	0	60	0	11	4	0	
29425	31	242984	10	0	0	40	4	15	5	6	
2088	37	588003	13	15024	0	40	4	9	2	4	
16292	40	170730	9	0	0	50	4	11	2	3	

```
from sklearn.tree import DecisionTreeClassifier
dt_default = DecisionTreeClassifier(max_depth=5)
dt_default.fit(X_train,y_train)
```

```
▾      DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
y_pred_default = dt_default.predict(X_test)
print(classification_report(y_test,y_pred_default))
```

```
              precision    recall  f1-score   support

           0       0.86      0.95      0.90      7475
           1       0.76      0.50      0.60      2294

    accuracy                           0.84      9769
   macro avg       0.81      0.72      0.75      9769
weighted avg       0.84      0.84      0.83      9769
```

```
print(confusion_matrix(y_test,y_pred_default))
print(accuracy_score(y_test,y_pred_default))
```

```
[[7111  364]
 [1154 1140]]
0.8446105026102979
```

```
!pip install pydotplus
```

```
Requirement already satisfied: pydotplus in /usr/local/lib/python3.10/dist-packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.10/dist-packages (from pydotplus) (3.1.1)
```

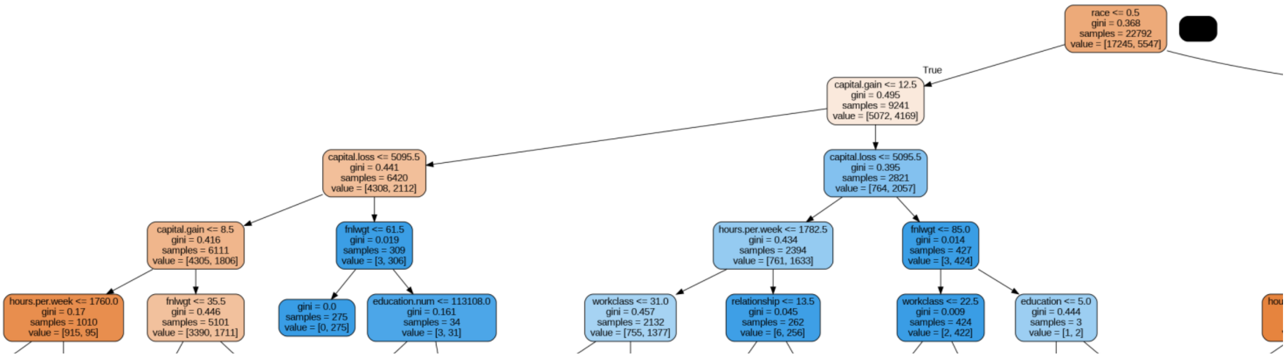
```
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydotplus,graphviz

# Putting features
features = list(df.columns[1:])
features
```

```
['fnlwgt',
 'education.num',
 'capital.gain',
 'capital.loss',
 'hours.per.week',
 'workclass',
 'education',
 'marital.status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native.country',
 'income']
```

```
dot_data = StringIO()
export_graphviz(dt_default, out_file=dot_data,
                feature_names=features, filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



The hyperparameter min_samples_leaf indicates the minimum number of samples required to be at a leaf. So if the values of min_samples_leafis less, say 5, then the will be constructed even if a leaf has 5, 6 etc. observations (and is likely to overfit). Let's see what will be the optimum value for min_samples_leaf.

```
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
n_folds = 5
parameters = {'min_samples_leaf': range(5, 200, 20)}

dtree = DecisionTreeClassifier(criterion = "gini",random_state = 100)
# fit tree on training data
tree = GridSearchCV(dtree, parameters,cv=n_folds,scoring="accuracy")
tree.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	params	split0_test_score	split1
0	0.228439	0.065863	0.013353	0.006299	5	{'min_samples_leaf': 5}	0.828032	
1	0.127333	0.016337	0.006742	0.000326	25	{'min_samples_leaf': 25}	0.843606	
						{'min_samples_leaf':		

The hyperparameter min_samples_split is the minimum no. of samples required to split an internal node. Its default value is 2, which means that even if a node is having 2 samples it can be furthur divided into leaf nodes.

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-58-030718213b9e> in <cell line: 3>()
      2 plt.figure()
      3 plt.plot(scores["param_min_samples_leaf"],
----> 4          scores["mean_train_score"],
      5          label="training accuracy")
      6 plt.plot(scores["param_min_samples_leaf"],

KeyError: 'mean_train_score'
```

SEARCH STACK OVERFLOW

```
<Figure size 640x480 with 0 Axes>
```

```
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	params	split0_test_score	split1
0	0.228439	0.065863	0.013353	0.006299	5	{'min_samples_leaf': 5}	0.828032	
1	0.127333	0.016337	0.006742	0.000326	25	{'min_samples_leaf': 25}	0.843606	
2	0.110355	0.018286	0.007531	0.002102	45	{'min_samples_leaf': 45}	0.850625	
3	0.098082	0.006524	0.007646	0.002988	65	{'min_samples_leaf': 65}	0.850186	
4	0.114428	0.034493	0.010196	0.005237	85	{'min_samples_leaf': 85}	0.849309	

Finding The Optimal Hyperparameters

```
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
    'criterion': ["entropy", "gini"]
}

n_folds = 5
dtree = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator = dtree, param_grid = param_grid,cv = n_folds, verbose = 1)
grid_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
     GridSearchCV
 estimator: DecisionTreeClassifier
   DecisionTreeClassifier
```

```
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

3	0.070686	0.007395	0.006872	0.001339	entropy	5	100
4	0.105225	0.007187	0.006467	0.000176	entropy	10	50
5	0.103518	0.003829	0.006739	0.001023	entropy	10	50
6	0.102757	0.009600	0.006512	0.000272	entropy	10	100
7	0.070052	0.006765	0.004625	0.000395	entropy	10	100
8	0.040169	0.001968	0.004211	0.000686	gini	5	50
9	0.039028	0.003692	0.003358	0.000090	gini	5	50
10	0.037827	0.001184	0.003831	0.000681	gini	5	100
11	0.037002	0.000282	0.003332	0.000090	gini	5	100
12	0.061435	0.001203	0.003849	0.000430	gini	10	50
13	0.065688	0.003816	0.004903	0.001368	gini	10	50
14	0.076610	0.008686	0.005366	0.000817	gini	10	100

```
print("best accuracy", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
best accuracy 0.8530186783184268
DecisionTreeClassifier(max_depth=10, min_samples_leaf=100, min_samples_split=50)
```

```
clf_gini = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=10,
                                  min_samples_leaf=50,
                                  min_samples_split=50)
clf_gini.fit(X_train, y_train)
```

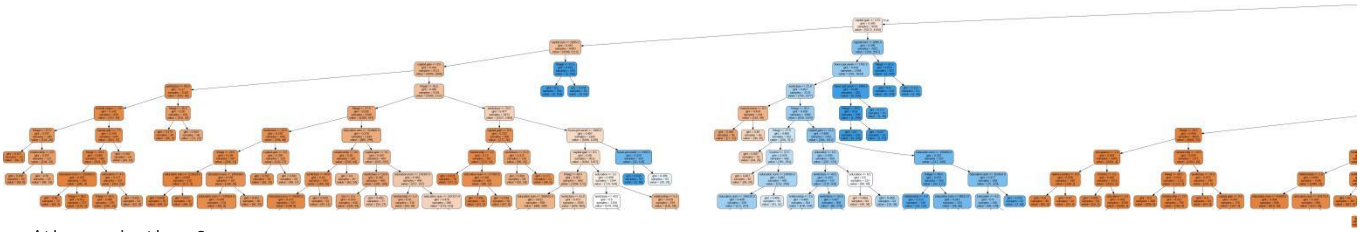
```
▾                              DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, min_samples_split=50,
                       random_state=100)
```

```
clf_gini.score(X_test,y_test)
```

```
0.8520831200737026
```

```
dot_data = StringIO()
export_graphviz(clf_gini, out_file=dot_data,feature_names=features,filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



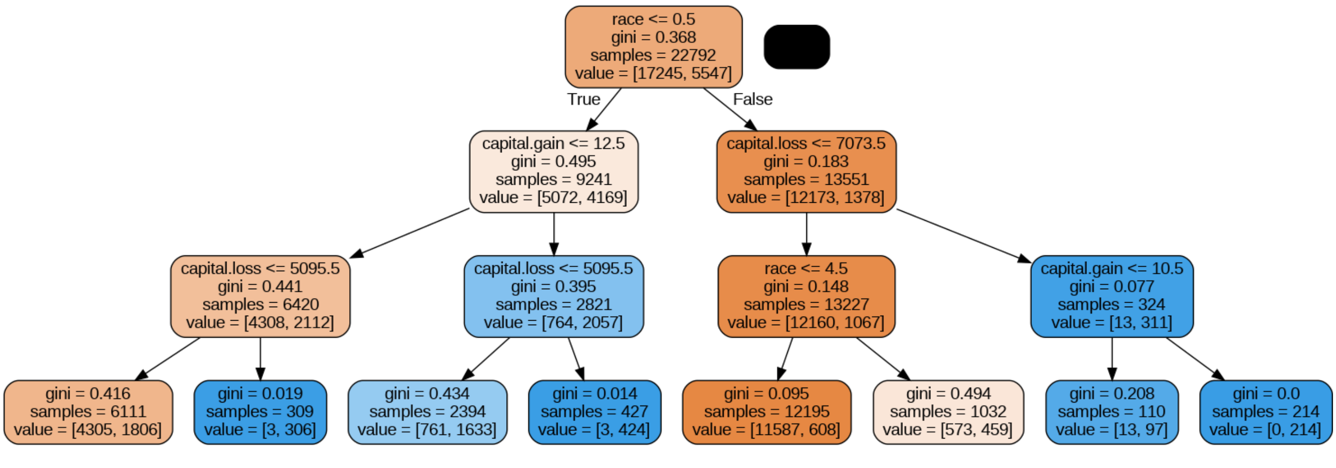
```
# tree with max_depth = 3
clf_gini = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=3,
                                  min_samples_leaf=50,
                                  min_samples_split=50)
clf_gini.fit(X_train, y_train)

# score
print(clf_gini.score(X_test,y_test))
```

```
0.8400040945849114
```

```
dot_data = StringIO()
export_graphviz(clf_gini, out_file=dot_data,feature_names=features,filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```




```
from sklearn.metrics import classification_report,confusion_matrix
y_pred = clf_gini.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.95      0.90      7475
           1       0.76      0.47      0.58      2294

    accuracy                           0.84      9769
   macro avg       0.81      0.71      0.74      9769
weighted avg       0.83      0.84      0.83      9769
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[7136  339]
 [1224 1070]]
```

```
PRINT
```