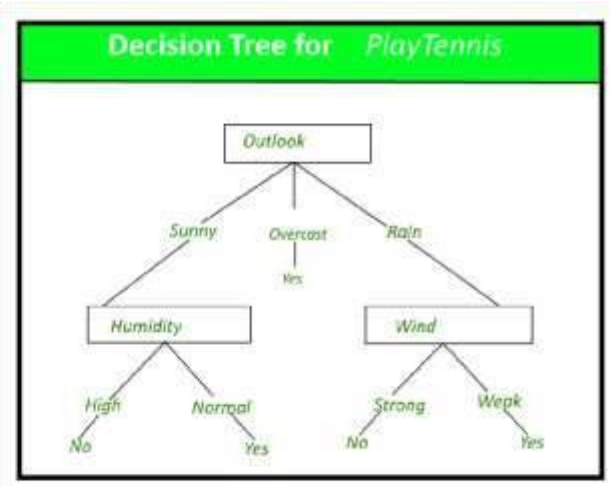
Experiment No. 3
Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:07–08–23
Date of Submission:17–08–23

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, DominicanRepublic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

Code:

Conclusion:

On the Adult Census Income Dataset, Decision Tree model performed well. To improve the model's efficacy, it handled categorical attributes using one-hot encoding and carried out the required data preprocessing, including handling missing values, removing pointless tables, and splitting columns.

The method of hyperparameter tuning is crucial for improving the performance of the Decision Tree model since it gives users some control over the model's complexity by imposing some parameters' upper bounds. We must enhance the model by modifying hyperparameters like max depth, min samples split, etc. utilising techniques like Grid Search or Random Search in order to increase performance.

Accuracy: reached a 0.85 accuracy level, which means that about 85% of forecasts were accurate.

Confusion Matrix: False positive = 481, False negative = 1823, True positive = 9860Precision: Precision of 0.84 suggests that among the instances predicted as 0, about 0.77 are predicted for

Recall: Recall of 0.95 indicates that the model captured the instances for 0 and 0.47 model captured the instances for 1.

F1 Score: The F1 score of 0.90 is the mean of precision and recall for 0 and 0.59 is the mean of precision and recall for 1 in the model's performance.

```
# 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') # Adult
dataset path adult_dataset_path =  "/content/adult.csv"


# Function for loading adult dataset def load_adult_data(adult_path=adult_dataset_path):
csv_path = os.path.join(adult_path)
return pd.read_csv(csv_path)


# Calling load adult function and assigning to a new variable df df = load_adult_data()  # load top 3 rows values
from adult dataset df.head(3)
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship
0	25	Private	226802	11th	7	Nevermarried	Machineop-inspct	Own-child
1	38	Private	89814	HS-grad	9	Marriedcivspouse	Farmingfishing	Husband
2	28	Local-gov	336951	Assocacdm	12	Marriedcivspouse	Protectiveserv	Husband

```
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 :  48842
Columns :  15

Features :
 ['age', 'workclass', 'fnlwgt', 'education', 'educational-num', 'marital-status', 'occupation',

'relationship', 'r Missing values :  0

Unique values :
 age                  74
workclass             9
fnlwgt            28523
education            16
educational-num      16
marital-status        7
occupation           15
relationship          6
race                  5
gender                2
capital-gain        123
capital-loss         99
hours-per-week       96
```

```
native-country        42
income                 2
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              48842 non-null  int64
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64
 11  capital-loss     48842 non-null  int64
 12  hours-per-week   48842 non-null  int64
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

```
df.describe()
```

	age	fnlwgt	educational-num	capital-gain	capital-loss	hours-per-week
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.422382
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.391444
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.000000

```
df.head()
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child
1	38	Private	89814	HS-grad		Married-	Farming	

```
# checking "?" total values present in particular 'workclass' feature
df_check_missing_workclass = (df['workclass'
```

]=='?').sum9() civ- fishing Husband spouse df_check_missing_workclass

Married-

27992 28 Local-gov 336951 Assoc-acdm 12 civ- Protective-serv Husband

spouse

checking "?" total values present in particular 'occupation' featureMarried-

df_check_missing_occupation = (df[3 44 Private 160323'occupation'Some-]=='?').10sum() civ- Machine-Husband

college op-inspct df_check_missing_occupation spouse

28094 18 ? 103497 Some-married 10 Never- ? Own-child college

```
# checking "?" values, how many are there in the w hole dataset
df_missing = (df=='?').sum()
df_missing
```

```
age                  0
workclass         2799
fnlwgt               0
education            0
educational-num      0
marital-status       0
occupation        2809
relationship         0
race                 0
gender               0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     857
income               0
dtype: int64
```

```
percent_missing = (df=='?').sum() * 100/len(df)
```

percent_missing

```
age                0.000000
workclass          5.730724
fnlwgt             0.000000
education          0.000000
educational-num    0.000000
marital-status     0.000000
occupation         5.751198
relationship       0.000000
race               0.000000
gender             0.000000
capital-gain       0.000000
capital-loss       0.000000
hours-per-week     0.000000
native-country     1.754637
income             0.000000
dtype: float64
```

```
# find total number of rows which doesn't contain any missing value as '?' df.apply(lambda
x: x !='?',axis=1).sum()
```

```
age                48842
workclass          46043
fnlwgt             48842
education          48842
educational-num    48842
```

```
     marital-status     48842
     occupation         46033
     relationship       48842 race
     48842 gender
     48842 capital-gain
     48842 capital-loss
     48842 hours-per-week
     48842 native-country     47985
     income
     48842 dtype: int64
```

```
# dropping the rows having missing values in workclass
df = df[df['workclass'] !='?'] df.head()
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband
5	34	Private	198693	10th	6	Never-married	Other-service	Not-in-family

```
# select all categorical variables df_categorical =
df.select_dtypes(include=['object']) # checking whether any
other column contains '?' value
df_categorical.apply(lambda x: x=='?',axis=1).sum()
```

```
     workclass           0
     education           0
     marital-status      0
     occupation         10
     relationship        0
     race                0
     gender              0
     native-country    811
     income              0
     dtype: int64
```

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

	workclass	education	marital-status	occupation	relationship	race	gender	native-country	inc
0	Private	11th	Never-married	Machine-op-inspct	Own-child	Black	Male	United-States	<=

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

	workclass	education	marital-status	occupation	relationship	race	gender	native-country	inco
0	2	1	4	6	3	2	1	39	
1	2	11	2	4	0	4	1	39	
2	1	7	2	10	0	4	1	39	
3	2	15	2	6	0	2	1	39	

```
# Next, Concatenate df_categorical dataframe with original df (dataframe)
# first, 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	educationalnum	capitalgain	capitalloss	hoursperweek	workclass	education	maritalstatus
0	25	226802	7	0	0	40	2	1	4
1	38	89814	9	0	0	50	2	11	2
2	28	336951	12	0	0	40	1	7	2
3	44	160323	10	7688	0	40	2	15	2
5	34	198693	6	0	0	30	2	0	4

```
# look at column type df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 46033 entries, 0 to 48841 Data
columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
0   age              46033 non-null  int64
1   fnlwgt           46033 non-null  int64
2   educational-num  46033 non-null  int64
3   capital-gain     46033 non-null  int64
4   capital-loss     46033 non-null  int64
5   hours-per-week   46033 non-null  int64
6   workclass        46033 non-null  int64
7   education        46033 non-null  int64 8 marital-status   46033 non-null  int64
9 occupation       46033 non-null  int64  10
relationship     46033 non-null  int64
11  race             46033 non-null  int64
12  gender           46033 non-null  int64
13  native-country   46033 non-null  int64  14  income
```

```
        46033 non-null  int64 dtypes: int64(15) memory usage:
        5.6 MB
# convert target variable income to categorical df['income']
= df['income'].astype('category')
# check df info again whether everything is in right format or not df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 46033 entries, 0 to 48841 Data
columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
0   age             46033 non-null  int64
1   fnlwgt          46033 non-null  int64    2 educational-num  46033 nonnull  int64
3   capital-gain    46033 non-null  int64
4   capital-loss    46033 non-null  int64
5   hours-per-week  46033 non-null  int64
6   workclass       46033 non-null  int64
7   education       46033 non-null  int64
8   marital-status  46033 non-null  int64
9   occupation      46033 non-null  int64     10 relationship     46033 non-null  int64
11  race            46033 non-null  int64
12  gender          46033 non-null  int64
13  native-country  46033 non-null  int64    14  income          46033 non-null  category dtypes:
    category(1), int64(14) memory usage: 5.3 MB
```

```
# Importing train_test_split
from sklearn.model_selection import train_test_split
# Putting independent variables/features to X
X = df.drop('income',axis=1)
# Putting response/dependent variable/feature to y y
= df['income']
```

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

	age	fnlwgt	educational-num	capital-gain	capital-loss	hours-per-week	workclass	education	marital-status	occupation	relationship
0	25	226802	7	0	0	40	2	1	4	6	3
1	38	89814	9	0	0	50	2	11	2	4	0
2	28	336951	12	0	0	40	1	7	2	10	0

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

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

```
# Splitting the data into train and test
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,random_state=99) X_train.head()
```

hours-

	age	fnlwgt	educational-num	capital-gain	capital-loss	per-week	workclass	education	marist
29293	39	203070	11	0	0	49	2	8	
3452	50	243115	9	0	0	40	2	11	
9788	31	154227	10	0	0	40	2	15	

```
# Importing decision tree classifier from sklearn library
from sklearn.tree import DecisionTreeClassifier
# Fitting the decision tree with default hyperparameters, apart from
# max_depth which is 5 so that we can plot and read the tree.
dt_default = DecisionTreeClassifier(max_depth=5)
dt_default.fit(X_train,y_train)
```

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

```
# check the evaluation metrics of our default model
# Importing classification report and confusion matrix from sklearn metrics from sklearn.metrics
import classification_report,confusion_matrix,accuracy_score
# making predictions
y_pred_default = dt_default.predict(X_test) # Printing  classifier
report after prediction
print(classification_report(y_test,y_pred_default))               precision
recall  f1-score   support

    0       0.86      0.95      0.90     10341
    1       0.79      0.53      0.64      3469

    accuracy                           0.85     13810
   macro avg       0.83      0.74      0.77     13810
weighted avg       0.84      0.85      0.84     13810
```

```
# Printing confusion matrix and accuracy print(confusion_matrix(y_test,y_pred_default))
print(accuracy_score(y_test,y_pred_default))

    [[9861  480]
     [1616 1853]]
    0.848225923244026
```

```
!pip install my-package

    Collecting my-package
      Downloading my_package-0.0.0-py3-none-any.whl (2.0 kB)
    Installing collected packages: my-package
    Successfully installed my-package-0.0.0
```

```
!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
```

```
# Importing required packages for visualization
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',
 'educational-num',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'workclass',
 'education',
 'marital-status',
 'occupation',
 'relationship',
 'race',
 'gender',
 'native-country',
 'income']
```

```
!pip install graphviz
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (0.20.1)
```

```
# plotting tree with max_depth=3 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())
```

```
# GridSearchCV to find optimal max_depth from
sklearn.model_selection import KFold from
sklearn.model_selection import GridSearchCV #
specify number of folds for k-fold CV n_folds =
5
# parameters to build the model on parameters
= {'max_depth': range(1, 40)}
# instantiate the model 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 of GridSearch CV scores
= tree.cv_results_
pd.DataFrame(scores).head()
```

mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	p

0	0.017508	0.001094	0.003653	0.000255	1	{'max_
1	0.026208	0.000638	0.003497	0.000050	2	{'max_
2	0.035516	0.000600	0.003571	0.000109	3	{'max_
3	0.046339	0.002691	0.003698	0.000265	4	{'max_
4	0.054046	0.000881	0.003738	0.000126	5	{'max_

```
# GridSearchCV to find optimal max_depth from
sklearn.model_selection import KFold from
sklearn.model_selection import GridSearchCV #
specify number of folds for k-fold CV n_folds =
5
# parameters to build the model on parameters = {'min_samples_leaf':
range(5, 200, 20)}
# instantiate the model
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 of GridSearch CV scores
= tree.cv_results_
pd.DataFrame(scores).head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf
0	0.137336	0.004291	0.004951	0.000940	5
1	0.115085	0.003973	0.004306	0.000304	25
2	0.107868	0.005090	0.004147	0.000113	45
3	0.101131	0.002170	0.004166	0.000221	65
4	0.100600	0.004742	0.004072	0.000051	85

```
# GridSearchCV to find optimal min_samples_split
from sklearn.model_selection import KFold from
sklearn.model_selection import GridSearchCV #
specify number of folds for k-fold CV n_folds =
5
# parameters to build the model on parameters =
```

```
{'min_samples_split': range(5, 200, 20)}
# instantiate the model 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 of GridSearch CV scores
= tree.cv_results_
pd.DataFrame(scores).head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_spli
0	0.153947	0.002147	0.005929	0.000578	
1	0.143463	0.004214	0.005008	0.000071	2
2	0.138373	0.001752	0.005027	0.000082	4
3	0.133974	0.005993	0.004872	0.000038	6
4	0.133274	0.005639	0.005146	0.000392	8

```
# Create the parameter grid 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
```

```
# Instantiate the grid search model dtree = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator = dtree, param_grid = param_grid,
cv = n_folds, verbose = 1) # Fit the grid search to the data
grid_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

```
cv_results▸ GridSearchCV          # cv results▸estimator:
             DecisionTreeClassifier
      ults = pd.DataFrame(grid_search.cv_re
          ▸DecisionTreeClassifier
```

6	0.137194	0.005227	0.006264	0.001309	entropy
7	0.122113	0.018699	0.004998	0.000869	entropy
8	0.052541	0.002334	0.003601	0.000090	gini
9	0.051650	0.000660	0.003619	0.000087	gini

```
# printing the optimal accuracy score and hyperparameters
print("best accuracy", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
best accuracy 0.8523105983446813
DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, min_samples_split=50)
```

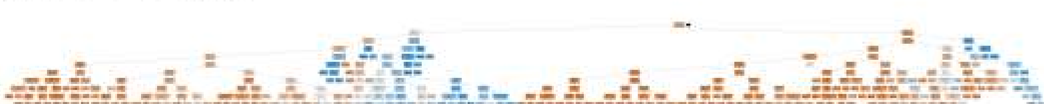
```
# model with optimal hyperparameters
clf_gini = DecisionTreeClassifier(criterion = "gini",
random_state = 100,
max_depth=10,
min_samples_leaf=50,
min_samples_split=50)
clf_gini.fit(X_train, y_train)
```

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

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

```
0.852860246198407
```

```
#plotting the tree
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)
```

```
min_samples_split 50)  clf_gini.fit(X_train, y_train)
# score print(clf_gini.score(X_test,y_test)) # plotting tree with max_depth=3 dot_data = StringIO()
export_graphviz(clf_gini, out_file=dot_data,feature_names=features,filled=True,rounded=True)
```

```
    0.8331643736422882
```

```
# classification metrics
from sklearn.metrics import classification_report,confusion_matrix  y_pred = clf_gini.predict(X_test)
print(classification_report(y_test, y_pred)) # confusion matrix  print(confusion_matrix(y_test,y_pred))
precision    recall  f1-score   support

    0  0.84      0.95      0.90     10341            1      0.77      0.47      0.59      3469

        accuracy                           0.83     13810
    macro avg       0.81      0.71      0.74     13810
    weighted avg       0.83      0.83      0.82     13810

    [[9860  481]
     [1823 1646]]
```