Experiment No. 6
Apply Boosting Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:
Date of Submission:

Aim: Apply Boosting algorithm on Adult Census Income Dataset and analyze the performance of the model.

Objective: Apply Boosting algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score.

Theory:

Suppose that as a patient, you have certain symptoms. Instead of consulting one doctor, you choose to consult several. Suppose you assign weights to the value or worth of each doctor's diagnosis, based on the accuracies of previous diagnosis they have made. The final diagnosis is then a combination of the weighted diagnosis. This is the essence behind boosting.

Algorithm: Adaboost- A boosting algorithm—create an ensemble of classifiers. Each one

gives a weighted vote.

Input:
- D , a set of d class labelled training tuples
- k, the number of rounds (one classifier is generated per round)
- a classification learning scheme

Output: A composite model

Method
1. Initialize the weight of each tuple in D is 1/d
2. For i=1 to k do // for each round
3. Sample D with replacement according to the tuple weights to obtain D_i
4. Use training set D_i to derive a model M_i
5. Computer error(M_i), the error rate of M_i
6. Error(M_i)=$\sum w_j$*err(X_j)
7. If Error(M_i)>0.5 then
8. Go back to step 3 and try again
9. endif
10. for each tuple in D_i that was correctly classified do
11. Multiply the weight of the tuple by error(Mi)/(1-error(M_i)
12. Normalize the weight of each tuple
13. end for

To use the ensemble to classify tuple X

1. Initialize the weight of each class to 0
2. for i=1 to k do // for each classifier
3. w_i=log((1-error(M_i))/error(M_i))//weight of the classifiers vote
4. C=M_i(X) // get class prediction for X from M_i
5. Add w_i to weight for class C
6. end for
7. Return the class with the largest weight.

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. Comment on the accuracy, confusion matrix, precision, recall and F1 score obtained.

We used two different types of booster and results are as follows:

AdaBoostClassifier:

Accuracy: 0.8637719588995691

F1 score : 0.7008007765105557

Precision : 0.7921009325287987

The accuracy obtained is pretty decent. Additionally, the F1 score of 0.70 and precision of 0.79 show a good balance between correctly identifying positive cases and minimizing false positives.

XGBClassifier:

Accuracy : 0.8683018450999989

F1 score : 0.7149689143950263

Precision : 0.7935244161358811

The model demonstrates a commendable accuracy of 86.83%, indicating strong overall predictive performance. Additionally, the F1 score of 0.71 and precision of 0.79 show a good balance between correctly identifying positive cases and minimizing false positives, highlighting its effectiveness in classification tasks.

Both the booster gives almost similar results.

2. Compare the results obtained by applying boosting and random forest algorithm on the Adult Census Income Dataset

Both models show high accuracy, with Boosting slightly outperforming Random Forest.

In terms of F1 score, both models have similar performance, but Boosting has a slightly better balance between precision and recall.

Random Forest has higher precision for class 0 (0.89) but lower precision for class 1 (0.75) compared to Boosting's precision of 0.79 for both classes.

Random Forest exhibits better recall for class 0 but lower recall for class 1.

Boosting tends to provide a more balanced classification performance across both classes, whereas Random Forest excels in classifying the majority class (class 0) but falls short in class 1.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import io
from sklearn.metrics import accuracy_score, precision_score, f1_score,confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
```

```
import os
for dirname, _, filenames in os.walk('/content/adult.csv'):
    for filename in filenames:
       print(os.path.join(dirname, filename))
```

```
file = ('/content/adult.csv')
df = pd.read_csv(file)
```

```
print(df.head())
```

```
   age workclass  fnlwgt     education  education.num marital.status  \
0   90         ?   77053       HS-grad              9        Widowed
1   82   Private  132870       HS-grad              9        Widowed
2   66         ?  186061  Some-college             10        Widowed
3   54   Private  140359       7th-8th              4       Divorced
4   41   Private  264663  Some-college             10      Separated

          occupation   relationship   race     sex  capital.gain  \
0                  ?  Not-in-family  White  Female             0
1    Exec-managerial  Not-in-family  White  Female             0
2                  ?      Unmarried  Black  Female             0
3  Machine-op-inspct      Unmarried  White  Female             0
4     Prof-specialty      Own-child  White  Female             0

   capital.loss  hours.per.week native.country income
0          4356              40  United-States  <=50K
1          4356              18  United-States  <=50K
2          4356              40  United-States  <=50K
3          3900              40  United-States  <=50K
4          3900              40  United-States  <=50K
```

```
print(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   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64
 5   marital.status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64
 11  capital.loss    32561 non-null  int64
 12  hours.per.week  32561 non-null  int64
 13  native.country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
None
```

```
df=df.loc[(df['workclass'] != '?') & (df['native.country'] != '?')]
print(df.head())
```

```
   age workclass  fnlwgt     education  education.num marital.status  \
1   82   Private  132870       HS-grad              9        Widowed
3   54   Private  140359       7th-8th              4       Divorced
4   41   Private  264663  Some-college             10      Separated
5   34   Private  216864       HS-grad              9       Divorced
6   38   Private  150601          10th              6      Separated

          occupation   relationship   race     sex  capital.gain  \
1    Exec-managerial  Not-in-family  White  Female             0
3  Machine-op-inspct      Unmarried  White  Female             0
4     Prof-specialty      Own-child  White  Female             0
5      Other-service      Unmarried  White  Female             0
6       Adm-clerical      Unmarried  White    Male             0
```

```
   capital.loss  hours.per.week native.country income
1          4356              18  United-States  <=50K
3          3900              40  United-States  <=50K
4          3900              40  United-States  <=50K
5          3770              45  United-States  <=50K
6          3770              40  United-States  <=50K
```

```
df["income"] = [1 if i=='>50K' else 0 for i in df["income"]]
print(df.head())
```

```
   age workclass  fnlwgt     education  education.num marital.status  \
1   82   Private  132870       HS-grad              9        Widowed
3   54   Private  140359       7th-8th              4       Divorced
4   41   Private  264663  Some-college             10      Separated
5   34   Private  216864       HS-grad              9       Divorced
6   38   Private  150601          10th              6      Separated

          occupation   relationship   race     sex  capital.gain  \
1    Exec-managerial  Not-in-family  White  Female             0
3  Machine-op-inspct      Unmarried  White  Female             0
4     Prof-specialty      Own-child  White  Female             0
5      Other-service      Unmarried  White  Female             0
6       Adm-clerical      Unmarried  White    Male             0

   capital.loss  hours.per.week native.country  income
1          4356              18  United-States       0
3          3900              40  United-States       0
4          3900              40  United-States       0
5          3770              45  United-States       0
6          3770              40  United-States       0
<ipython-input-7-595c69654189>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df["income"] = [1 if i=='>50K' else 0 for i in df["income"]]
```

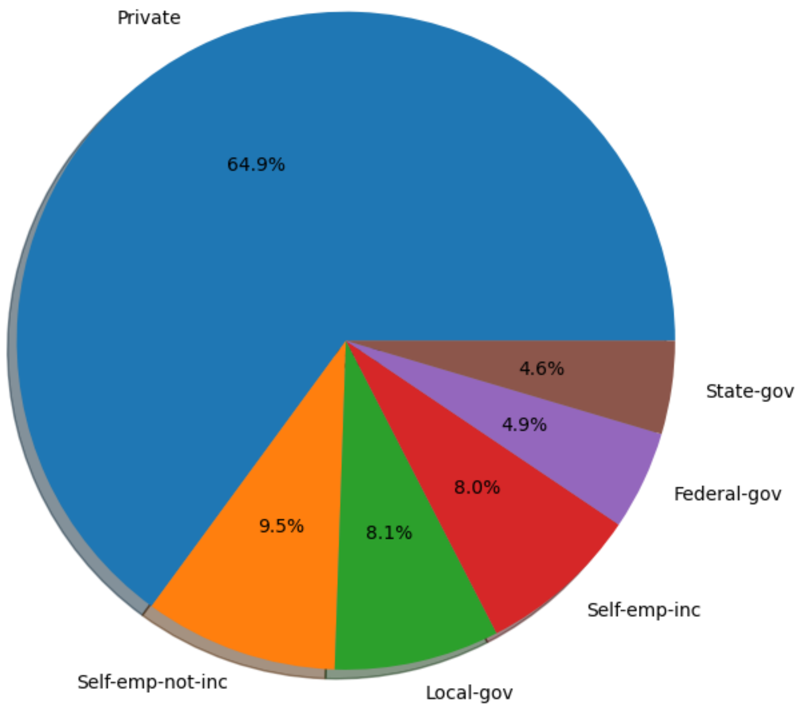
```
df_more=df.loc[df['income'] == 1]
print(df_more.head())
```

```
    age         workclass  fnlwgt    education  education.num marital.status  \
7    74         State-gov   88638    Doctorate             16  Never-married
10   45           Private  172274    Doctorate             16       Divorced
11   38  Self-emp-not-inc  164526  Prof-school             15  Never-married
12   52           Private  129177    Bachelors             13        Widowed
13   32           Private  136204      Masters             14      Separated

         occupation    relationship   race     sex  capital.gain  \
7    Prof-specialty  Other-relative  White  Female             0
10   Prof-specialty       Unmarried  Black  Female             0
11   Prof-specialty   Not-in-family  White    Male             0
12    Other-service   Not-in-family  White  Female             0
13  Exec-managerial   Not-in-family  White    Male             0

    capital.loss  hours.per.week native.country  income
7           3683              20  United-States       1
10          3004              35  United-States       1
11          2824              45  United-States       1
12          2824              20  United-States       1
13          2824              55  United-States       1
```

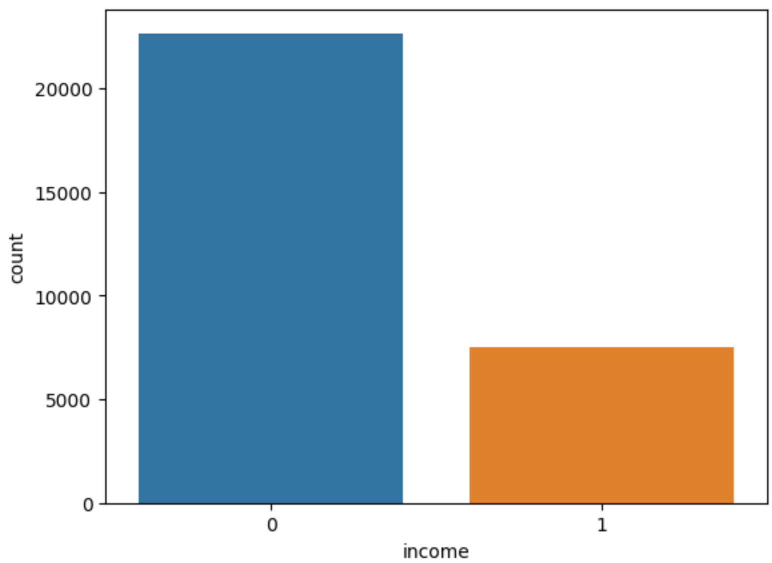
```
workclass_types = df_more['workclass'].value_counts()
labels = list(workclass_types.index)
aggregate = list(workclass_types)
print(workclass_types)
print(aggregate)
print(labels)
```

```
Private             4876
Self-emp-not-inc     714
Local-gov            609
Self-emp-inc         600
Federal-gov          365
State-gov            344
Name: workclass, dtype: int64
[4876, 714, 609, 600, 365, 344]
['Private', 'Self-emp-not-inc', 'Local-gov', 'Self-emp-inc', 'Federal-gov', 'State-gov']
```

```
plt.figure(figsize=(7,7))
plt.pie(aggregate, labels=labels, autopct='%1.1f%%', shadow = True)
plt.axis('equal')
plt.show()
```




```
#Count plot on single categorical variable
sns.countplot(x ='income', data = df)
plt.show()
df['income'].value_counts()
```

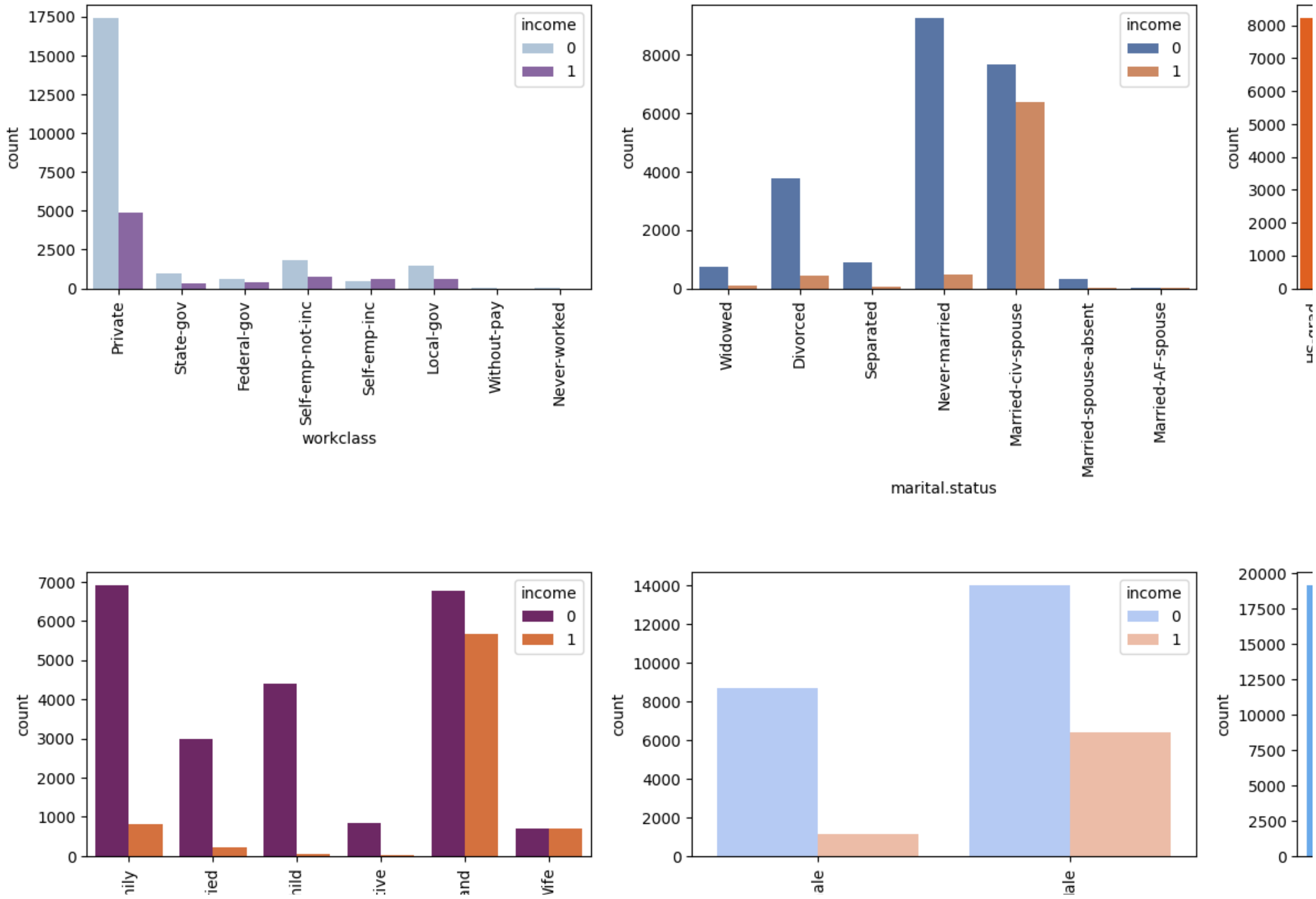



```
0    22661
1     7508
Name: income, dtype: int64
```

```
#To find distribution of categorical columns w.r.t income
fig, axes = plt.subplots(figsize=(20, 10))
plt.subplot(231)
sns.countplot(x ='workclass',
hue='income',
data = df,
palette="BuPu")
plt.xticks(rotation=90)
plt.subplot(232)
sns.countplot(x ='marital.status',
hue='income',
data = df,
palette="deep")
plt.xticks(rotation=90)
plt.subplot(233)
sns.countplot(x ='education',
hue='income',
data = df,
```

```
palette = "autumn")
plt.xticks(rotation=90)
plt.subplot(234)
sns.countplot(x ='relationship',
hue='income',
data = df,
palette = "inferno")
plt.xticks(rotation=90)
plt.subplot(235)
sns.countplot(x ='sex',
hue='income',
data = df,
palette = "coolwarm")
plt.xticks(rotation=90)
plt.subplot(236)
sns.countplot(x ='race',
hue='income',
data = df,
palette = "cool")
plt.xticks(rotation=90)
plt.subplots_adjust(hspace=1)
plt.show()
```

```
<ipython-input-13-e1b6d1f0108f>:3: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will b
  plt.subplot(231)
```




```
df1 = df.copy()
categorical_features = list(df1.select_dtypes(include=['object']).columns)
print(categorical_features)
df1
```

```
['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'native.country']
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capit
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	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	
...	...	...	...	...	...	...	...	...	...	...	...	
32556	22	Private	310152	Some-	10	Never-married	Protective-	Not-in-family	White	Male	0	

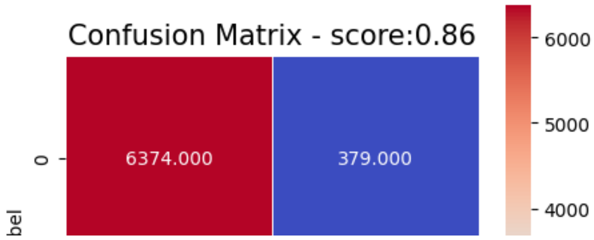
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for feat in categorical_features:
        df1[feat] = le.fit_transform(df1[feat].astype(str))
        df1


X = df1.drop(columns = ['income'])
y = df1['income'].values
# Splitting the data set into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,random_state = 0)
print ("Train set size: ", X_train.shape)
print ("Test set size: ", X_test.shape)

from sklearn.ensemble import AdaBoostClassifier
# Train Adaboost Classifer
abc = AdaBoostClassifier(n_estimators = 300, learning_rate=1)
abc_model = abc.fit(X_train, y_train)
#Prediction
y_pred_abc = abc_model.predict(X_test)
```

```
Train set size:  (21118, 14)
Test set size:  (9051, 14)
```

```
print("Accuracy: ", accuracy_score(y_test, y_pred_abc))
print("F1 score :",f1_score(y_test, y_pred_abc, average='binary'))
print("Precision : ", precision_score(y_test, y_pred_abc))
```

```
Accuracy:  0.8637719588995691
F1 score : 0.7008007765105557
Precision :  0.7921009325287987
```

```
cm = confusion_matrix(y_test, y_pred_abc)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap ="coolwarm");
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
plt.title('Confusion Matrix - score:' + str(round(accuracy_score(y_test,y_pred_abc), 2)), size = 15);
plt.show()
print(classification_report(y_test, y_pred_abc))
```

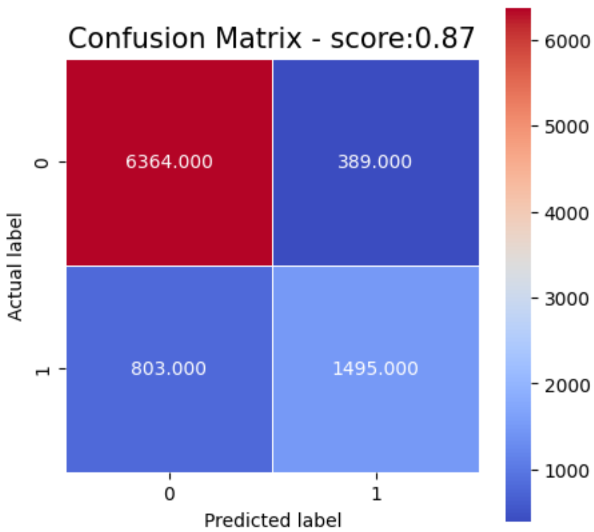



```
import xgboost as xgb
from xgboost import XGBClassifier
#Training the model with gradient boosting
xgboost = XGBClassifier(learning_rate=0.01,
colsample_bytree = 0.4,
n_estimators=1000,
max_depth=20,
gamma=1)
xgboost_model = xgboost.fit(X_train, y_train)
```

Predicted label

```
# Predictions
y_pred_xgboost = xgboost_model.predict(X_test)
print("Accuracy : ",accuracy_score(y_test, y_pred_xgboost))
print("F1 score : ", f1_score(y_test, y_pred_xgboost, average = 'binary'))
print("Precision : ", precision_score(y_test, y_pred_xgboost))

rms = np.sqrt(mean_squared_error(y_test, y_pred_xgboost))
print("RMSE for xgboost: ", rms)

cm = confusion_matrix(y_test, y_pred_xgboost)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap ="coolwarm");
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
plt.title('Confusion Matrix - score:'+str(round(accuracy_score(y_test,y_pred_xgboost),2)), size = 15);
plt.show()
print(classification_report(y_test,y_pred_xgboost))
```

```
Accuracy :  0.868301845099989
F1 score :  0.7149689143950263
Precision :  0.7935244161358811
RMSE for xgboost:  0.3629024040978663
```




```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      6753
           1       0.79      0.65      0.71      2298

    accuracy                           0.87      9051
   macro avg       0.84      0.80      0.81      9051
weighted avg       0.86      0.87      0.86      9051
```