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:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split,cross_val_score,KFold,GridSearchCV
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
import scikitplot as skplt


dataset=pd.read_csv("/content/adult.csv")
print(dataset.isnull().sum())
```

```
age               0
workclass         0
fnlwgt            0
education         0
education.num     0
marital.status    0
occupation        0
relationship      0
race              0
sex               0
capital.gain      0
capital.loss      0
hours.per.week    0
native.country    0
income            0
dtype: int64
```

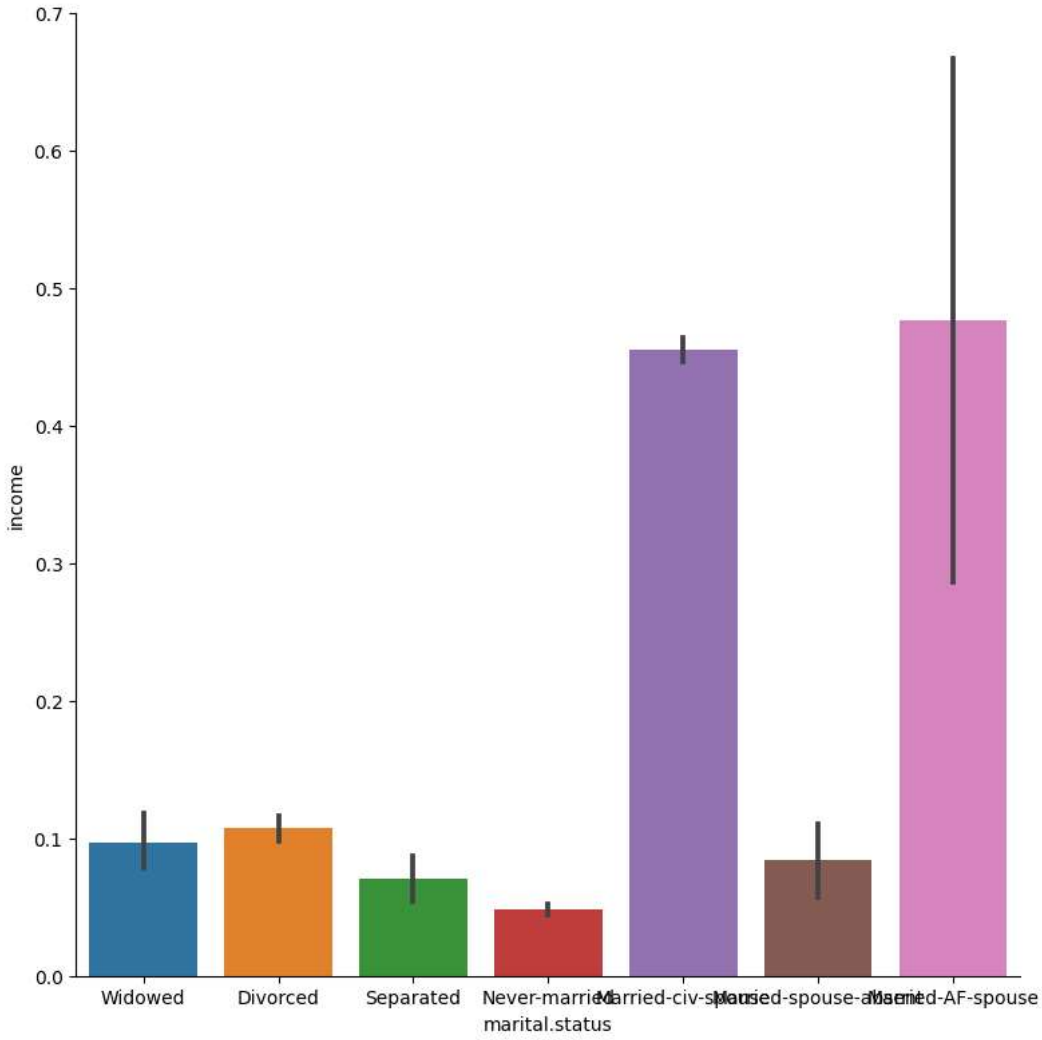
```
dataset.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900

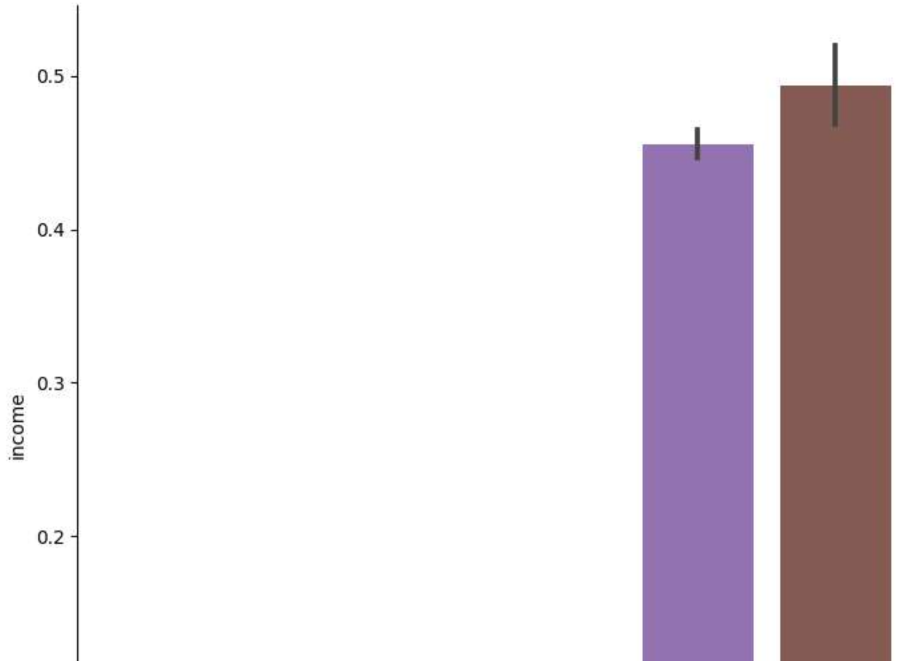
```
#removing '?' containing rows
dataset = dataset[(dataset != '?').all(axis=1)]
#label the income objects as 0 and 1
dataset['income']=dataset['income'].map({'<=50K': 0, '>50K': 1})


sns.catplot(x='education.num',y='income',data=dataset,kind='bar',height=6)
plt.show()
```


```
sns.catplot(x='marital.status',y='income',data=dataset,kind='bar',height=8)
plt.show()
```



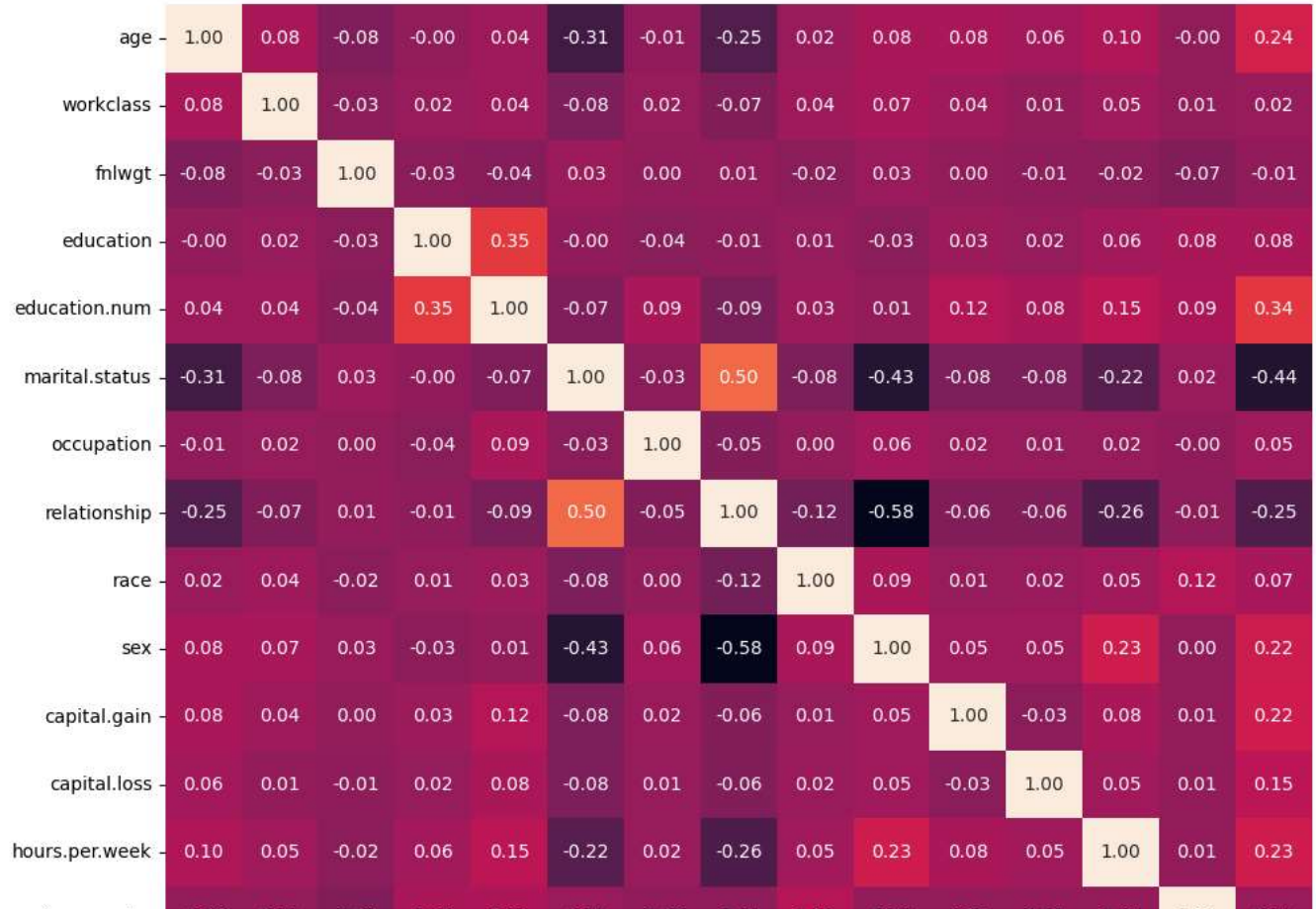

```
sns.catplot(x='relationship',y='income',data=dataset,kind='bar',height=7)
plt.show()
```




```
dataset['marital.status']=dataset['marital.status'].map({'Married-civ-spouse':'Married', 'Divorced':'Single', 'Never-married':'Single', 'Sepa
'Widowed':'Single', 'Married-spouse-absent':'Married', 'Married-AF-spouse':'Married'})
for column in dataset:
    enc=LabelEncoder()
    if dataset.dtypes[column]==np.object:
        dataset[column]=enc.fit_transform(dataset[column])
```

```
<ipython-input-43-08500846f906>:5: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warn
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-43-08500846f906>:5: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warn
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-43-08500846f906>:5: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warn
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-43-08500846f906>:5: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warn
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-43-08500846f906>:5: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warn
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-43-08500846f906>:5: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warn
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-43-08500846f906>:5: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warn
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-43-08500846f906>:5: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warn
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-43-08500846f906>:5: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warn
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
```

```
plt.figure(figsize=(14,10))
sns.heatmap(dataset.corr(),annot=True,fmt='.2f')
plt.show()
```




```
dataset=dataset.drop(['relationship','education'],axis=1)
dataset=dataset.drop(['occupation','fnlwgt','native.country'],axis=1)
print(dataset.head())
```

```
    age  workclass  education.num  marital.status  race  sex  capital.gain  \
1    82          2              9               1     4    0             0
3    54          2              4               1     4    0             0
4    41          2             10               1     4    0             0
5    34          2              9               1     4    0             0
6    38          2              6               1     4    1             0

    capital.loss  hours.per.week  income
1           4356              18       0
3           3900              40       0
4           3900              40       0
5           3770              45       0
6           3770              40       0
```

```
X=dataset.iloc[:,0:-1]
y=dataset.iloc[:,-1]
print(X.head())
print(y.head())
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.33,shuffle=False)
```

```
    age  workclass  education.num  marital.status  race  sex  capital.gain  \
1    82          2              9               1     4    0             0
3    54          2              4               1     4    0             0
4    41          2             10               1     4    0             0
5    34          2              9               1     4    0             0
6    38          2              6               1     4    1             0

    capital.loss  hours.per.week
1           4356              18
3           3900              40
4           3900              40
5           3770              45
6           3770              40
1    0
3    0
4    0
5    0
```

```
6    0
Name: income, dtype: int64
```

```
import xgboost as xgb
xgb.__version__
dmat=xgb.DMatrix(x_train,y_train)
test_dmat=xgb.DMatrix(x_test)
```

```
from skopt import BayesSearchCV
import warnings
warnings.filterwarnings('ignore', message='The objective has been evaluated at this point before.')

params={'min_child_weight': (0, 10),
        'max_depth': (0, 30),
        'subsample': (0.5, 1.0, 'uniform'),
        'colsample_bytree': (0.5, 1.0, 'uniform'),
        'n_estimators':(50,100),
        'reg_lambda':(1,100,'log-uniform'),
        }

bayes=BayesSearchCV(estimator=xgb.XGBClassifier(objective='binary:logistic',eval_metric='error',eta=0.1),search_spaces=params,n_iter=50,scori
res=bayes.fit(x_train,y_train)
print(res.best_params_)
print(res.best_score_)
```

```
OrderedDict([('colsample_bytree', 0.5), ('max_depth', 6), ('min_child_weight', 10), ('n_estimators', 50), ('reg_lambda', 100), ('subsam
0.7892426478178091
```

```
final_p={'colsample_bytree': 1.0, 'max_depth': 3, 'min_child_weight': 0,'subsample': 0.5,'reg_lambda': 100.0,'objective':'binary:logistic','e
cv_res=xgb.cv(params=final_p,dtrain=dmat,num_boost_round=1000,early_stopping_rounds=100,metrics=['error'],nfold=5)
cv_res.tail()
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [06:50:35] WARNING: /workspace/src/learner.cc:742:
Parameters: { "n_estimators", "silent" } are not used.

  warnings.warn(smsg, UserWarning)
```

	train-error-mean	train-error-std	test-error-mean	test-error-std
734	0.130109	0.000372	0.137520	0.003300
735	0.129949	0.000501	0.137817	0.003137
736	0.130085	0.000657	0.137619	0.003216
737	0.130072	0.000475	0.137371	0.003124
738	0.130085	0.000658	0.137173	0.003238

```
final_clf=xgb.train(params=final_p,dtrain=dmat,num_boost_round=837)
pred=final_clf.predict(test_dmat)
print(pred)
pred[pred > 0.5 ] = 1
pred[pred <= 0.5] = 0
print(pred)
print(accuracy_score(y_test,pred)*100)
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [06:50:46] WARNING: /workspace/src/learner.cc:742:
Parameters: { "n_estimators", "silent" } are not used.

  warnings.warn(smsg, UserWarning)
[8.8464266e-01 8.0791152e-01 5.4679954e-01 ... 2.8672358e-01 3.3943113e-02
 5.4113433e-04]
[1. 1. 1. ... 0. 0. 0.]
85.0713281093028
```

```
classification_rep = classification_report(y_test, pred)
print("Classification Report:")
print(classification_rep)
```

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

           0       0.88      0.95      0.91      7942
           1       0.69      0.47      0.56      2012
```

```
            accuracy                           0.85      9954
           macro avg       0.78      0.71      0.74      9954
        weighted avg       0.84      0.85      0.84      9954
```

```
confusion_mat = confusion_matrix(y_test, pred)
print("Confusion Matrix:")
print(confusion_mat)
```

```
Confusion Matrix:
[[7521  421]
 [1065  947]]
```

Conclusion:

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

   Through this experiment by using XGBoost we have achieved an accuracy of 85.07%.
   We also achieved precision of 0.88 on 0 class and 0.69 on 1 class.
   We achieved a recall of 0.95 on 0 class and 0.44 on 1 class.
   We have achieved an F1 score of 0.91 on 0 class and 0.56 on 1 class.
   True Positive (TP): 947 instances were correctly predicted as class 1.
   True Negative (TN): 7521 instances were correctly predicted as class 0.
   False Positive (FP): 421 instances were wrongly predicted as class 1.
   False Negative (FN): 1065 instances were wrongly predicted as class 0.

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

   On the adult dataset we have applied two machine learning algorithms. In the previous experiment we had applied Random Forest and in this experiment we have used the XGBoost algorithm. In the previous experiment we had achieved accuracy of about 84.55 and by using XGBoost we achieved accuracy of 85.07.As you can see we got a difference of more than 0.5% Also the precision, recall and F1 score were comparatively higher.