| |
|---|
| Experiment No. 6 |
| Apply Boosting Algorithm on Adult Census Income Dataset and analyze the performance of the model |
| Date of Performance: 05/10/23 |
| Date of Submission: 12/10/23 |

**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.

In [4]:
```python
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import io
from sklearn.metrics import accuracy_score, precision_score, f1_score, confusi
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('adult.csv')
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
```

In [5]:
```
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
```

In [6]:

```
#Count the occuring of the '?' in all the columns
for i in df.columns:
    t = df[i].value_counts()
    index = list(t.index)
    print ("Count of ? in", i)
    for i in index:
        temp = 0
        if i == '?':
            print (t['?'])
            temp = 1
            break
    if temp == 0:
        print ("0")
```

```
Count of ? in age
0
Count of ? in workclass
1836
Count of ? in fnlwgt
0
Count of ? in education
0
Count of ? in education.num
0
Count of ? in marital.status
0
Count of ? in occupation
1843
Count of ? in relationship
0
Count of ? in race
0
Count of ? in sex
0
Count of ? in capital.gain
0
Count of ? in capital.loss
0
Count of ? in hours.per.week
0
Count of ? in native.country
583
Count of ? in income
0
```

In [7]:

```python
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
```

In [8]:

```python
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
```

In [9]:

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

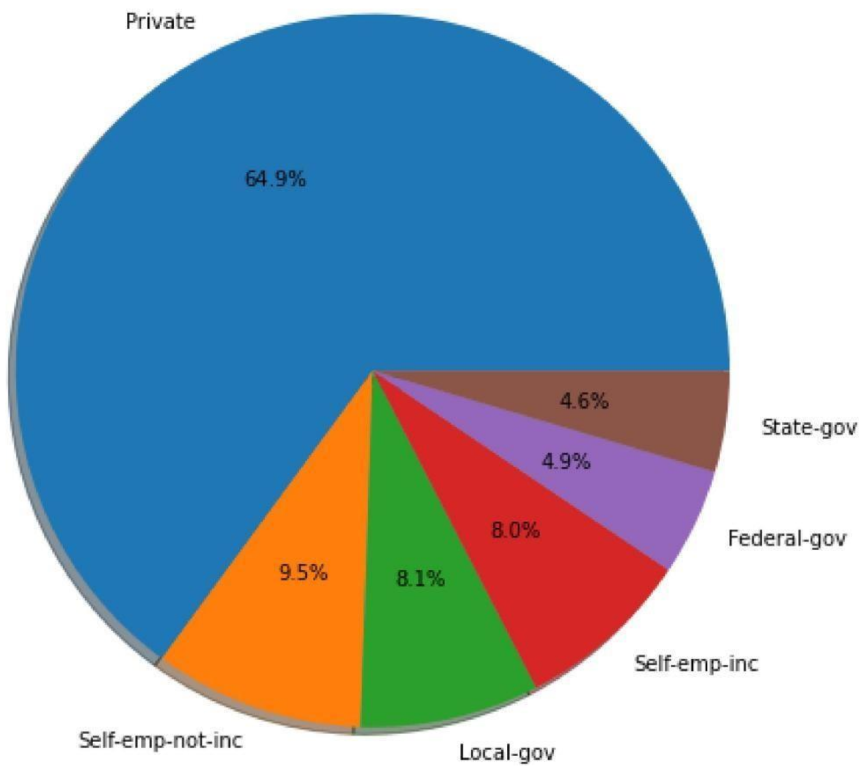
```
    age         workclass  fnlwgt    education  education.num marital.status  \
7    74         State-gov   88638    Doctorate             16  Never-married
10   45           Private  172274    Doctorate             16       Divorced
11   38  Self-emp-not-inc  164526  Prof-school             15  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
```

In [10]:

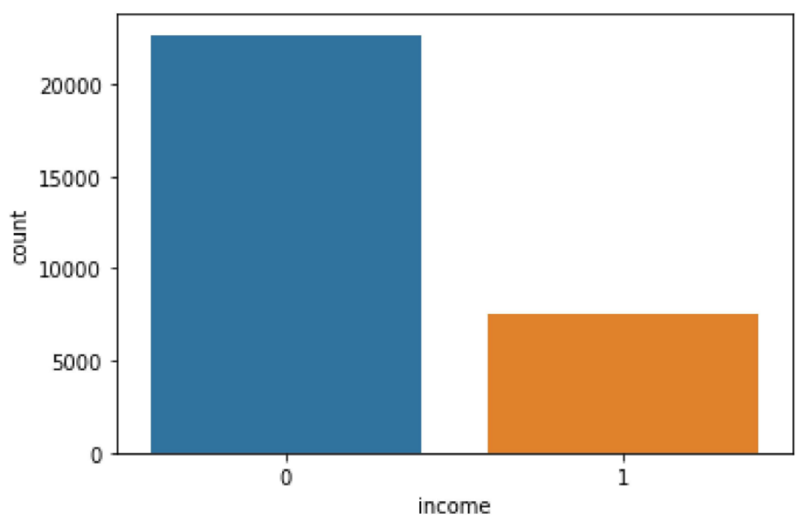
```
workclass_types = df_more['workclass'].value_counts()
labels = list(workclass_types.index)
aggregate = list(workclass_types)
print(workclass_types)
print(aggregate)
print(labels)
```

```
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']
```

In [11]:
```python
plt.figure(figsize=(7,7))
plt.pie(aggregate, labels=labels, autopct='%1.1f%%', shadow = True)
plt.axis('equal')
plt.show()
```

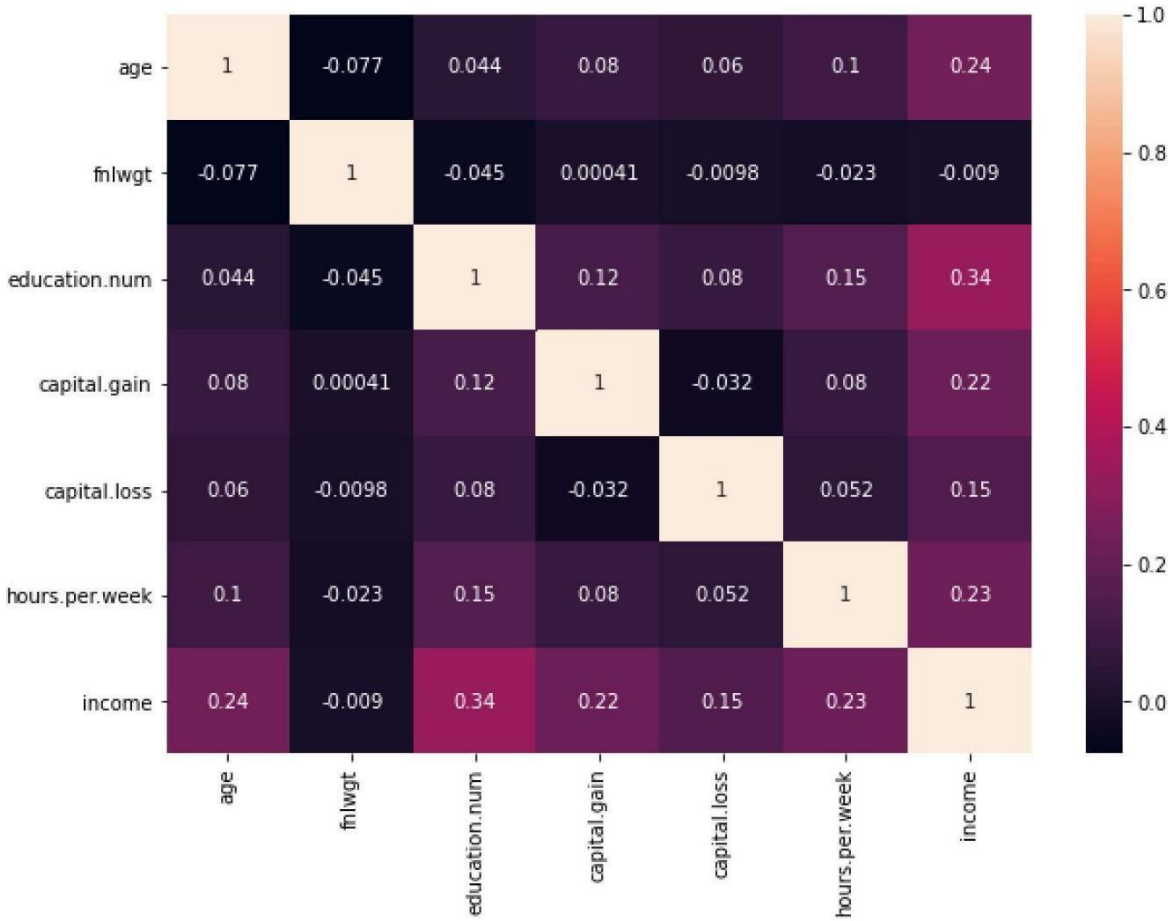



In [12]:
```python
#Count plot on single categorical variable
sns.countplot(x ='income', data = df)
plt.show()
df['income'].value_counts()
```



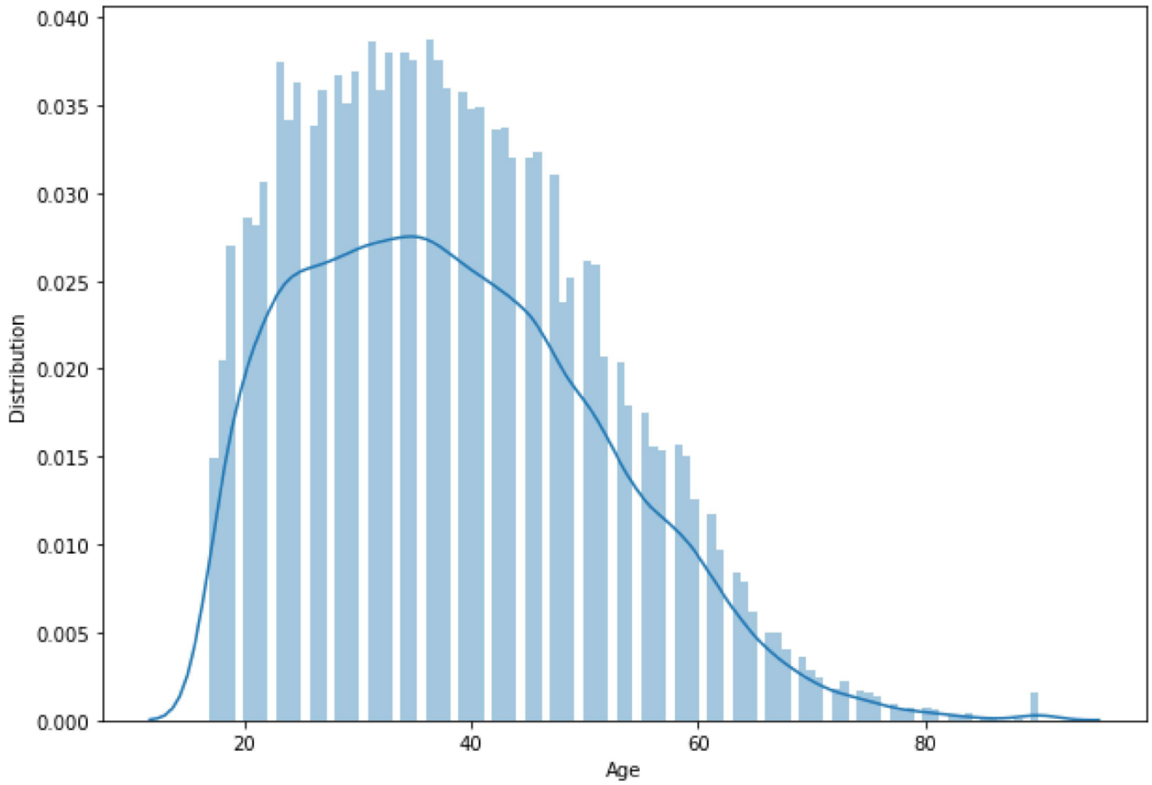

Out[12]:
```
0    22661
1     7508
Name: income, dtype: int64
```

In [32]:
```
#Plot figsize
plt.figure(figsize=(10,7))
sns.heatmap(df.corr(), annot=True)
print(plt.show())
```

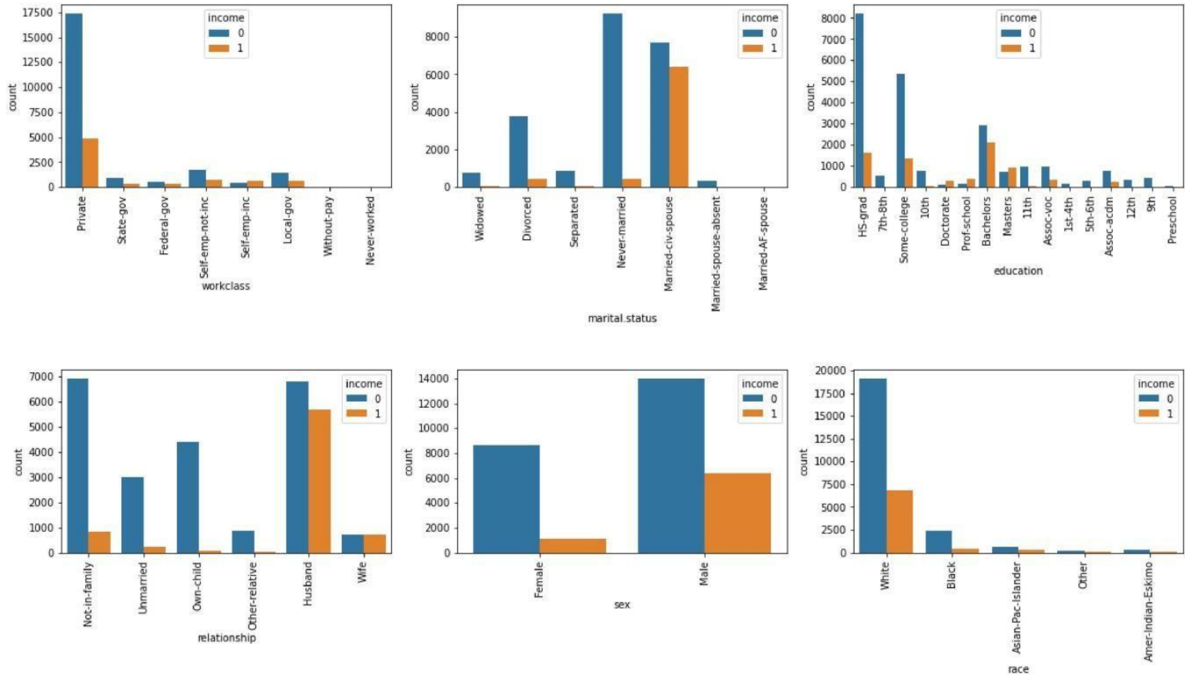



None

In [31]:
```
plt.figure(figsize=(10,7))
sns.distplot(df['age'], bins=100)
plt.ylabel("Distribution", fontsize = 10)
plt.xlabel("Age", fontsize = 10)
plt.show()
```



In [30]:
```
#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,
              )
plt.xticks(rotation=90)

plt.subplot(232)
sns.countplot(x ='marital.status',
              hue='income',
              data = df,
              )
plt.xticks(rotation=90)

plt.subplot(233)
sns.countplot(x ='education',
              hue='income',
              data = df,
              )
plt.xticks(rotation=90)

plt.subplot(234)
sns.countplot(x ='relationship',
              hue='income',
              data = df,
              )
plt.xticks(rotation=90)

plt.subplot(235)
sns.countplot(x ='sex',
              hue='income',
              data = df,
             )
plt.xticks(rotation=90)

plt.subplot(236)
sns.countplot(x ='race',
              hue='income',
              data = df,
              )
plt.xticks(rotation=90)
plt.subplots_adjust(hspace=1)
plt.show()
```




In [16]:
```
df1 = df.copy()
```

In [17]:
```
categorical_features = list(df1.select_dtypes(include=['object']).columns)
print(categorical_features)
df1
```

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

Out[17]:

| | age | workclass | fnlwgt | education | education.num | marital.status | occupation | relationshi |
|---|---|---|---|---|---|---|---|---|
| **1** | 82 | Private | 132870 | HS-grad | 9 | Widowed | Exec-managerial | Not-in-fami |
| **3** | 54 | Private | 140359 | 7th-8th | 4 | Divorced | Machine-op-inspct | Unmarrie |
| **4** | 41 | Private | 264663 | Some-college | 10 | Separated | Prof-specialty | Own-chi |
| **5** | 34 | Private | 216864 | HS-grad | 9 | Divorced | Other-service | Unmarrie |
| **6** | 38 | Private | 150601 | 10th | 6 | Separated | Adm-clerical | Unmarrie |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **32556** | 22 | Private | 310152 | Some-college | 10 | Never-married | Protective-serv | Not-in-fami |
| **32557** | 27 | Private | 257302 | Assoc-acdm | 12 | Married-civ-spouse | Tech-support | Wi |
| **32558** | 40 | Private | 154374 | HS-grad | 9 | Married-civ-spouse | Machine-op-inspct | Husbar |
| **32559** | 58 | Private | 151910 | HS-grad | 9 | Widowed | Adm-clerical | Unmarrie |
| **32560** | 22 | Private | 201490 | HS-grad | 9 | Never-married | Adm-clerical | Own-chi |

30169 rows × 15 columns

In [18]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for feat in categorical_features:
    df1[feat] = le.fit_transform(df1[feat].astype(str))
df1
```

Out[18]:

| | age | workclass | fnlwgt | education | education.num | marital.status | occupation | relationshi |
|---|---|---|---|---|---|---|---|---|
| 1 | 82 | 3 | 132870 | 11 | 9 | 6 | 4 | |
| 3 | 54 | 3 | 140359 | 5 | 4 | 0 | 7 | |
| 4 | 41 | 3 | 264663 | 15 | 10 | 5 | 10 | |
| 5 | 34 | 3 | 216864 | 11 | 9 | 0 | 8 | |
| 6 | 38 | 3 | 150601 | 0 | 6 | 5 | 1 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 32556 | 22 | 3 | 310152 | 15 | 10 | 4 | 11 | |
| 32557 | 27 | 3 | 257302 | 7 | 12 | 2 | 13 | |
| 32558 | 40 | 3 | 154374 | 11 | 9 | 2 | 7 | |
| 32559 | 58 | 3 | 151910 | 11 | 9 | 6 | 1 | |
| 32560 | 22 | 3 | 201490 | 11 | 9 | 4 | 1 | |

30169 rows × 15 columns

In [19]:
```
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,ranc

print ("Train set size: ", X_train.shape)
print ("Test set size: ", X_test.shape)
```

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

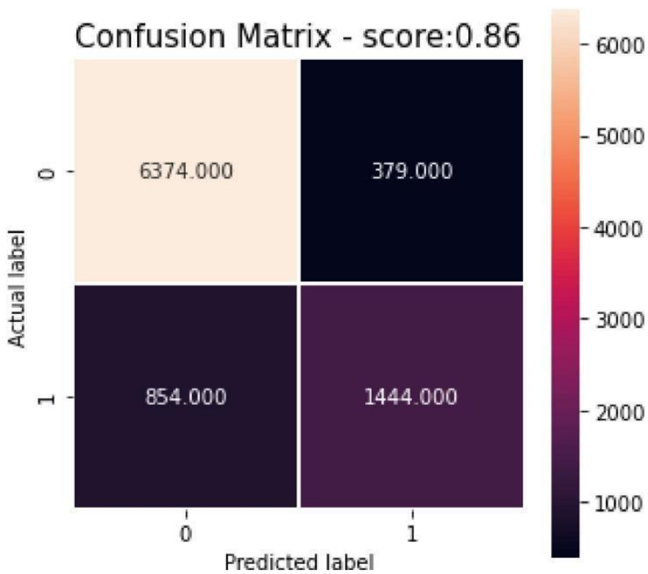
In [20]:

```
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)

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
```

In [23]:

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




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.88 | 0.94 | 0.91 | 6753 |
| 1 | 0.79 | 0.63 | 0.70 | 2298 |
| accuracy | | | 0.86 | 9051 |
| macro avg | 0.84 | 0.79 | 0.81 | 9051 |
| weighted avg | 0.86 | 0.86 | 0.86 | 9051 |

In [22]:

```
from sklearn.ensemble import GradientBoostingClassifier

#Training the model with gradient boosting
gbc = GradientBoostingClassifier(
    learning_rate = 0.1,
    n_estimators = 500,
    max_depth = 5,
    subsample = 0.9,
    min_samples_split = 100,
    max_features='sqrt',
    random_state=10)
gbc.fit(X_train,y_train)

# Predictions
y_pred_gbc = gbc.predict(X_test)

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

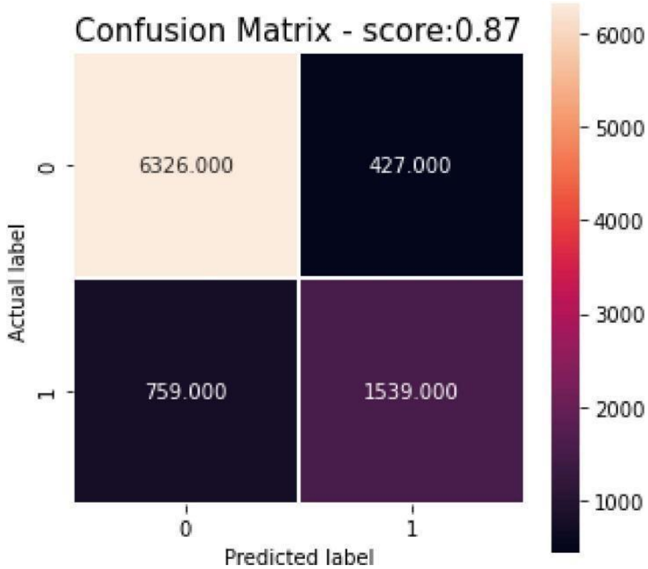
```
Accuracy :  0.8689647552756602
F1 score :  0.7218574108818011
Precision :  0.7828077314343845
```

In [24]:

```
rms = np.sqrt(mean_squared_error(y_test, y_pred_gbc))
print("RMSE for gradient boost: ", rms)
```

```
RMSE for gradient boost:  0.3619879068758235
```

In [25]:
```
cm = confusion_matrix(y_test, y_pred_gbc)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot = True, fmt=".3f", linewidths = 0.5, square = True);
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
plt.title('Confusion Matrix - score:' + str(round(accuracy_score(y_test, y_pre
plt.show()
print(classification_report(y_test, y_pred_gbc))
```




```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      6753
           1       0.78      0.67      0.72      2298

    accuracy                           0.87      9051
   macro avg       0.84      0.80      0.82      9051
weighted avg       0.86      0.87      0.87      9051
```

In [26]:

```python
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)

# 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))
```

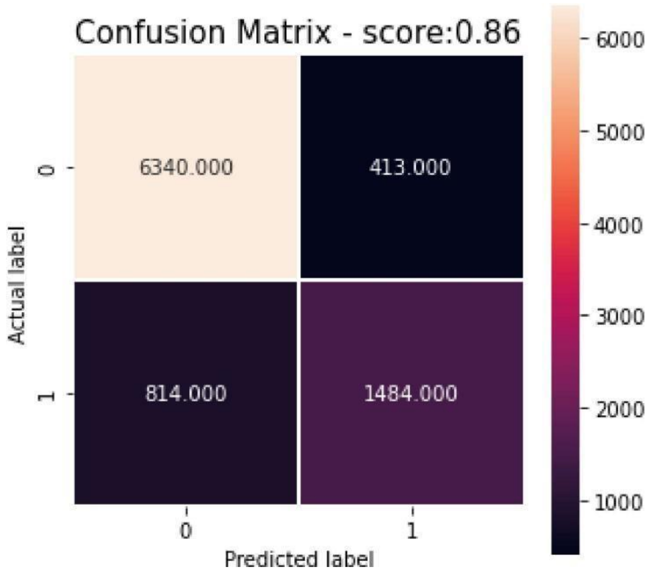
```
Accuracy :  0.8644348690752403
F1 score :  0.7075089392133492
Precision :  0.7822878228782287
```

In [27]:

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

```
RMSE for xgboost:  0.36819170404119606
```

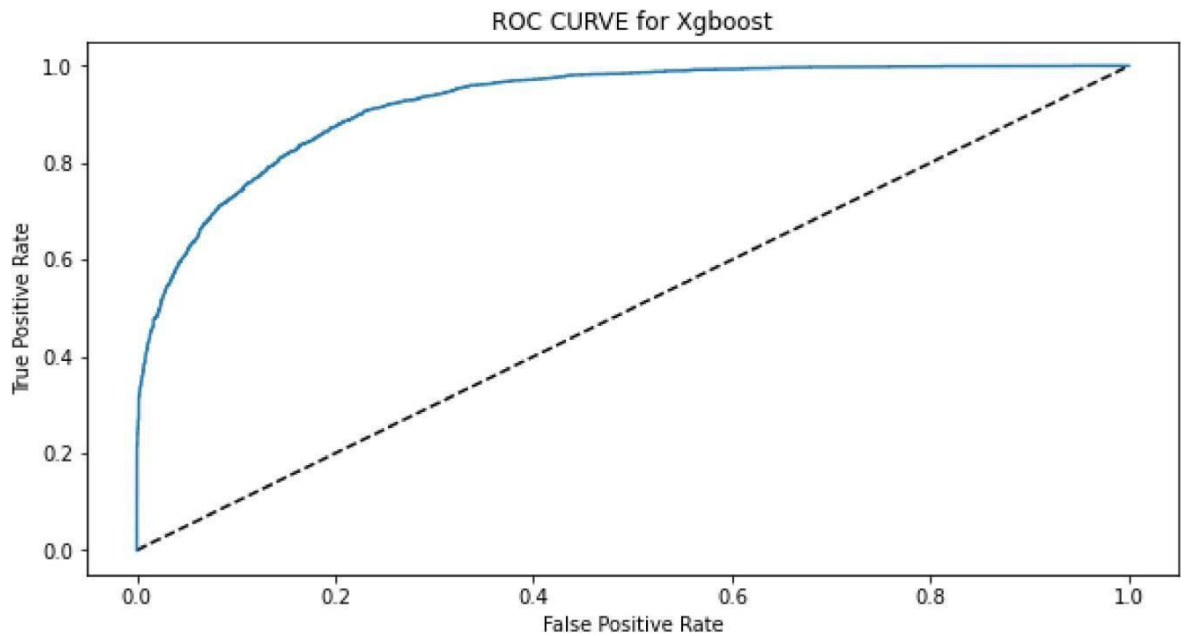
In [28]:
```
cm = confusion_matrix(y_test, y_pred_xgboost)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True);
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
plt.title('Confusion Matrix - score:'+str(round(accuracy_score(y_test, y_pred_
plt.show()
print(classification_report(y_test,y_pred_xgboost))
```




```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      6753
           1       0.78      0.65      0.71      2298

    accuracy                           0.86      9051
   macro avg       0.83      0.79      0.81      9051
weighted avg       0.86      0.86      0.86      9051
```

In [29]:

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, xgboost.predict_proba(X_test)[:,1])
plt.figure(figsize = (10,5))
plt.plot([0,1],[0,1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE for Xgboost')
plt.show()
```




I would like to conclude this notebook by mentioning that here, I have tuned the hyperparameters myself instead of using Grid Search or random search as it didn't seem to increase my accuracies. I would appreciate any suggestions or improvements that I can make to make this better.

**Conclusion:**

1. The Accuracy score obtained by applying boosting algo on the testing data is 0.86 which means our model is 86% accurate on the testing data.
2. Confusion matrix is used to assess the performance of a classification model, in our case the no. of TP is 1444, no. of TN is 6374, no. of FP is 379 and no. of FN are 854 which means our model is better in predicting negative cases than the positive cases.
3. Precision measures the accuracy of the positive predictions and the precision score obtained by our model is 0.88
4. Recall measures the ability of the model to correctly identify all relevant instances and the Recall score obtained by our model is 0.94
5. F1-score is the harmonic mean of precision and recall and provides a balance between the 2 metrics and the F1-score obtained by our model is 0.91
6. In the random forest algorithm, the accuracy, precision, recall and F1-score obtained respectively is 84%, 88%, 95%, 91%. and the accuracy, precision, recall and F1-score obtained by boosting algorithm respectively is 86%, 88%, 94%, 91%. Thus we can conclude that boosting algorithm is slightly better than the random forest algorithm