In [26]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb

import warnings
warnings.filterwarnings('ignore')
```

In [27]:
```
fr_data = pd.read_csv('Fraud_check.csv')
fr_data
```

Out[27]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
0	NO	Single	68833	50047	10	YES
1	YES	Divorced	33700	134075	18	YES
2	NO	Married	36925	160205	30	YES
3	YES	Single	50190	193264	15	YES
4	NO	Married	81002	27533	28	NO
...	...	...	...	...	...	...
595	YES	Divorced	76340	39492	7	YES
596	YES	Divorced	69967	55369	2	YES
597	NO	Divorced	47334	154058	0	YES
598	YES	Married	98592	180083	17	NO
599	NO	Divorced	96519	158137	16	NO

600 rows × 6 columns

In [28]:
```
fr_data.loc[fr_data['Taxable.Income'] <= 30000 ,'Taxable_Income'] = 'Risky'
fr_data.loc[fr_data['Taxable.Income'] > 30000 , 'Taxable_Income'] = 'Good'
```

In [29]:
```
fr_data
```

Out[29]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban	Taxable_Ir
0	NO	Single	68833	50047	10	YES	
1	YES	Divorced	33700	134075	18	YES	
2	NO	Married	36925	160205	30	YES	
3	YES	Single	50190	193264	15	YES	
4	NO	Married	81002	27533	28	NO	
...	...	...	...	...	...	...	
595	YES	Divorced	76340	39492	7	YES	
596	YES	Divorced	69967	55369	2	YES	
597	NO	Divorced	47334	154058	0	YES	
598	YES	Married	98592	180083	17	NO	
599	NO	Divorced	96519	158137	16	NO	

600 rows × 7 columns

In [30]:
```
fr_data.drop('Taxable.Income',axis = 1 , inplace = True)
fr_data.head()
```

Out[30]:

	Undergrad	Marital.Status	City.Population	Work.Experience	Urban	Taxable_Income
0	NO	Single	50047	10	YES	Good
1	YES	Divorced	134075	18	YES	Good
2	NO	Married	160205	30	YES	Good
3	YES	Single	193264	15	YES	Good
4	NO	Married	27533	28	NO	Good

In [31]:
```
fr_data.shape
```

Out[31]:
```
(600, 6)
```

In [32]:
```
fr_data.isna().sum()
```

Out[32]:
```
Undergrad          0
Marital.Status     0
City.Population    0
Work.Experience    0
Urban              0
Taxable_Income     0
dtype: int64
```

In [33]:
```
fr_data.dtypes
```

Out[33]:
```
Undergrad          object
Marital.Status     object
City.Population     int64
Work.Experience     int64
Urban              object
Taxable_Income     object
dtype: object
```

In [34]:
```
fr_data.describe()
```

Out[34]:

	City.Population	Work.Experience
count	600.000000	600.000000
mean	108747.368333	15.558333
std	49850.075134	8.842147
min	25779.000000	0.000000
25%	66966.750000	8.000000
50%	106493.500000	15.000000
75%	150114.250000	24.000000
max	199778.000000	30.000000

In [35]:
```
le = LabelEncoder()
fr_data['Undergrad'] = le.fit_transform(fr_data['Undergrad'])
fr_data['Undergrad'].unique()
```

Out[35]:
```
array([0, 1])
```

In [36]:
```
fr_data['Marital.Status'] = le.fit_transform(fr_data['Marital.Status'])
fr_data['Marital.Status'].unique()
```

Out[36]:
```
array([2, 0, 1])
```

In [37]:
```
fr_data['Urban'] = le.fit_transform(fr_data['Urban'])
```

In [44]:
```
fr_data['Taxable_Income'] = le.fit_transform(fr_data['Taxable_Income'])
```

In [45]:
```
fr_data.head()
```

Out[45]:

	Undergrad	Marital.Status	City.Population	Work.Experience	Urban	Taxable_Income
0	0	2	50047	10	1	0
1	1	0	134075	18	1	0
2	0	1	160205	30	1	0
3	1	2	193264	15	1	0
4	0	1	27533	28	0	0

In [46]:
```
x = fr_data.drop('Taxable_Income',axis = 1)
y = fr_data[['Taxable_Income']]
```

In [49]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=12,test_size=0.
```

In [50]:
```
x_train.shape,y_train.shape
```

Out[50]:
```
((480, 5), (480, 1))
```

In [17]:
```
x_test.shape,y_test.shape
```

Out[17]:
```
((120, 5), (120, 1))
```

In [18]:
```
rt = RandomForestClassifier(n_estimators= 1000,criterion='entropy')
```

In [19]:
```
rt.fit(x_train,y_train)
```

Out[19]:
```
RandomForestClassifier(criterion='entropy', n_estimators=1000)
```

In [20]:
```
y_pred_train = rt.predict(x_train)
```

In [21]:
```
print('Accuracy Score :',accuracy_score(y_train,y_pred_train))
```

```
Accuracy Score : 1.0
```

In [22]:
```
print('Confusion metrix :\n',confusion_matrix(y_train,y_pred_train))
```

```
Confusion metrix :
 [[369   0]
 [  0 111]]
```

In [23]:
```
y_pred_test = rt.predict(x_test)
```

In [24]:
```
print('Accuracy Score :',accuracy_score(y_test,y_pred_test))
```

```
Accuracy Score : 0.7916666666666666
```

In [25]:
```
print('Confusion metrix :\n',confusion_matrix(y_test,y_pred_test))
```

```
Confusion metrix :
 [[95 12]
 [13  0]]
```

In [52]:
```
train_x_y = xgb.DMatrix(data = x_train,label = y_train)
train_x_y
```

Out[52]:
```
<xgboost.core.DMatrix at 0x28c3a2a7e80>
```

In [53]:
```
parameters = {'max_depth':10,
              'objective':'binary:logistic',
              'eval_metric':'auc',
              'learning_rate':.05,}
```

In [54]:
```
xgb_classifier = xgb.train(dtrain = train_x_y , params = parameters)
```

In [55]:
```
x_train_Dm = xgb.DMatrix(x_train)
```

In [57]:
```
y_pred_tr = xgb_classifier.predict(x_train_Dm)
```

In [62]:
```
for i in range(0,y_pred_tr.shape[0]):
    if y_pred_tr[i]>=0.5:
        y_pred_tr[i] = 1
    else:
        y_pred_tr[i] = 0
```

In [63]:
```
print('Accuracy Score :',accuracy_score(y_train,y_pred_tr))
```

```
Accuracy Score : 0.8229166666666666
```

In [67]:
```
print('Confusion metrix :\n',confusion_matrix(y_train,y_pred_tr))
```

```
Confusion metrix :
 [[360   9]
 [ 76  35]]
```

In [ ]:
```

```

In [64]:
```
x_test_Dm = xgb.DMatrix(x_test)
```

In [70]:
```
y_pred_ts = xgb_classifier.predict(x_test_Dm)
```

In [71]:
```
for i in range(0,y_pred_ts.shape[0]):
    if y_pred_ts[i]>=0.5:
        y_pred_ts[i] = 1
    else:
        y_pred_ts[i] = 0
```

In [72]:
```
print('Accuracy Score :',accuracy_score(y_test,y_pred_ts))
```

```
Accuracy Score : 0.825
```

In [73]:
```
print('Confusion metrix :\n',confusion_matrix(y_test,y_pred_ts))
```

```
Confusion metrix :
 [[97 10]
 [11  2]]
```

In [ ]:
```

```