Assignment-15-Random Forest (Fraud Data)

Use Random Forest to prepare a model on fraud data treating those who have taxable_income <= 30000 as "Risky" and others are "Good"

Import Libararies

In [1]:

```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score,roc_curve,roc_auc_score
```

Import Data

In [2]:

```
data_fraud=pd.read_csv('Fraud_check.csv')
data_fraud
```

Out[2]:

	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

Data Understanding

Undergrad : person is under graduated or not

Taxable.Income : Taxable income is the amount of how much tax an individual owes to the government

Work Experience : Work experience of an individual person

Urban : Whether that person belongs to urban area or not

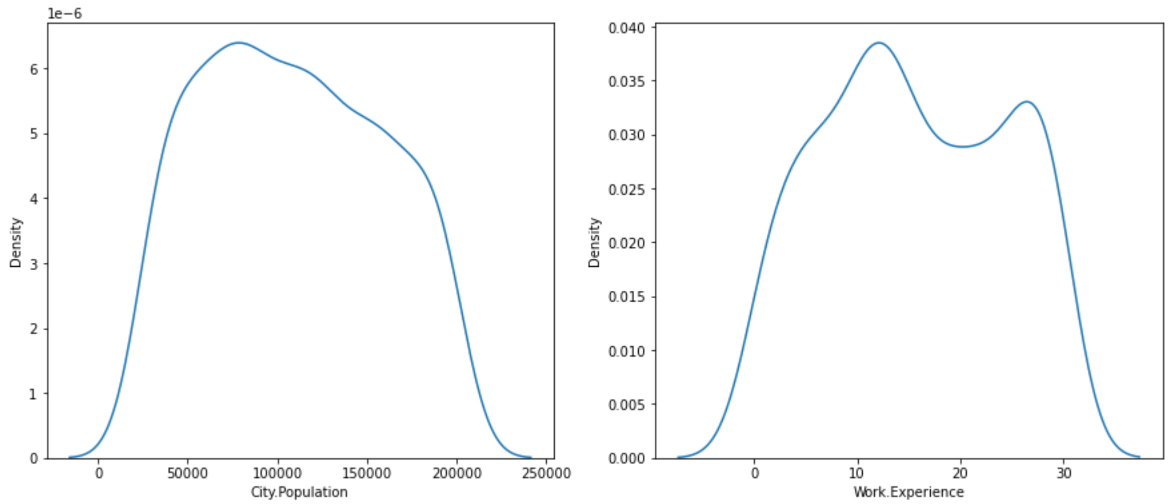
In [3]:

```
data_fraud.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Undergrad        600 non-null    object
 1   Marital.Status   600 non-null    object
 2   Taxable.Income   600 non-null    int64 
 3   City.Population  600 non-null    int64 
 4   Work.Experience  600 non-null    int64 
 5   Urban            600 non-null    object
dtypes: int64(3), object(3)
memory usage: 28.2+ KB
```

In [4]:
```
fig,axes=plt.subplots(1,2)
plt.figure(figsize=[15,15])

fig.set_figheight(6)
fig.set_figwidth(15)

sns.kdeplot(x="City.Population", data=data_fraud,ax=axes[0])

sns.kdeplot(x="Work.Experience", data=data_fraud,ax=axes[1])
```

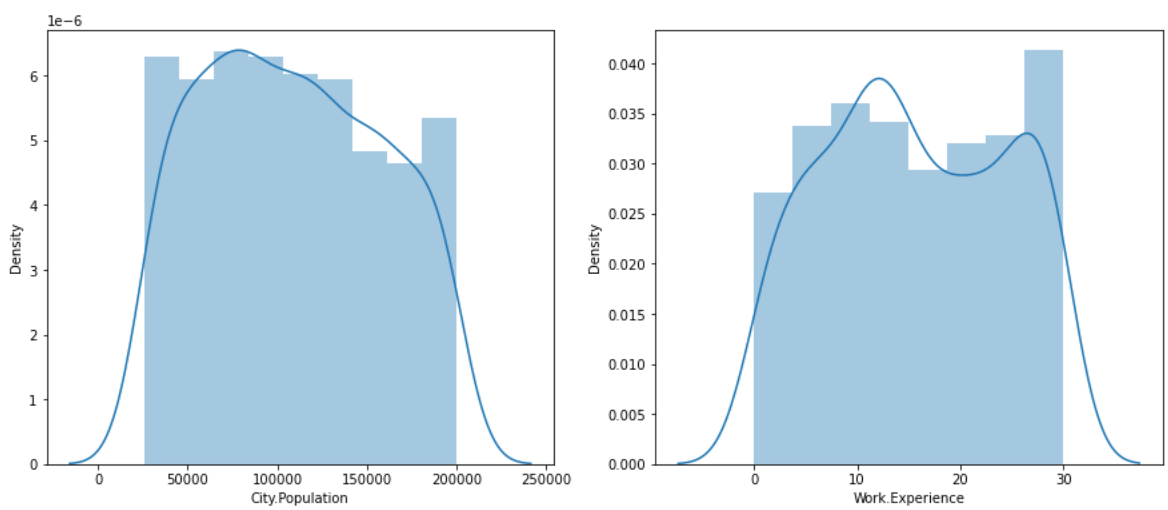
Out[4]: <AxesSubplot:xlabel='Work.Experience', ylabel='Density'>




```
<Figure size 1080x1080 with 0 Axes>
```

In [5]:
```
fig2,axes2=plt.subplots(1,2)
plt.figure(figsize=[5,15])

fig2.set_figheight(6)
fig2.set_figwidth(15)

sns.distplot(data_fraud["City.Population"],ax=axes2[0])

sns.distplot(data_fraud["Work.Experience"],ax=axes2[1])
```

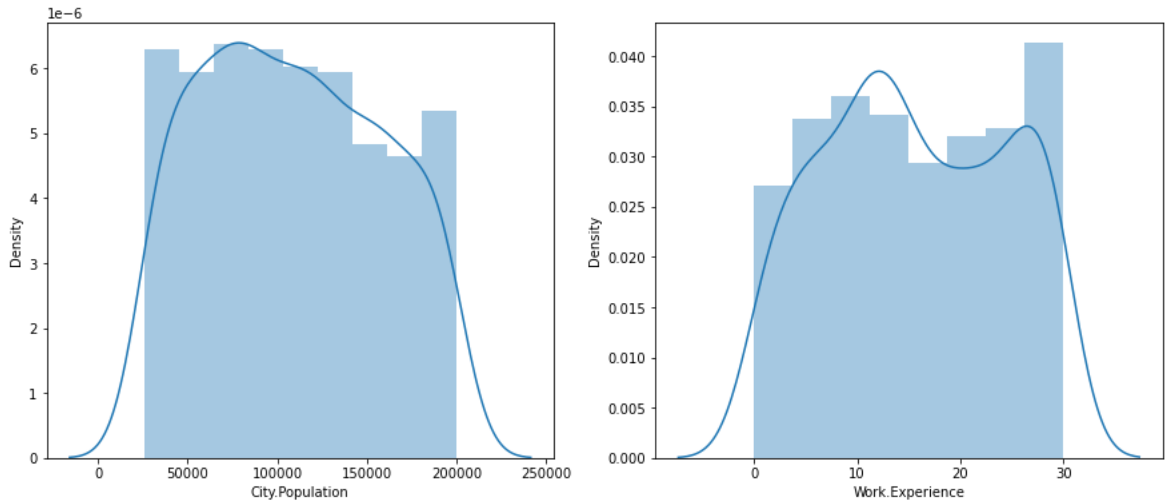
Out[5]: <AxesSubplot:xlabel='Work.Experience', ylabel='Density'>




```
<Figure size 360x1080 with 0 Axes>
```

In [6]:
```
fig2,axes2=plt.subplots(1,2)
plt.figure(figsize=[5,15])

fig2.set_figheight(6)
fig2.set_figwidth(15)

sns.distplot(data_fraud["City.Population"],ax=axes2[0])

sns.distplot(data_fraud["Work.Experience"],ax=axes2[1])
```

Out[6]: <AxesSubplot:xlabel='Work.Experience', ylabel='Density'>



<Figure size 360x1080 with 0 Axes>

Data Preparation

In [7]:
```
Income_List=[]
for i in data_fraud["Taxable.Income"]:
    if i >30000:
        Income_List.append(1)
    else:
        Income_List.append(0)
```

In [8]:
```
Undergrad_List=[]
for i in data_fraud["Undergrad"]:
    if i=="YES":
        Undergrad_List.append(1)
    else:
        Undergrad_List.append(0)
```

In [9]:
```
Urban_List=[]
for i in data_fraud["Urban"]:
    if i=="YES":
        Urban_List.append(1)
    else:
        Urban_List.append(0)
```

In [10]:
```
data_fraud["Taxable.Income"]=Income_List
data_fraud.Undergrad=Undergrad_List
data_fraud.Urban=Urban_List
```

In [11]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [12]:
```
data_fraud_copy=pd.read_csv('Fraud_check.csv')
```

In [13]:
```
data_fraud_copy["Marital.Status"]=le.fit_transform(data_fraud_copy["Marital.Status"])
```

In [14]:
```
data_fraud["Marital.Status"]=data_fraud_copy["Marital.Status"]
```

In [15]:
```
data_fraud.head()
```

Out[15]:

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

Model Building

In [16]:
```
X=data_fraud.drop("Taxable.Income",axis=1)
y=data_fraud[["Taxable.Income"]]
```

In [17]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.2,random_state=123)
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
```

```
(480, 5) (480, 1)
(120, 5) (120, 1)
```

Model Training

In [18]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model=RandomForestClassifier(max_depth=3,random_state=123)
rf_model.fit(X_train,y_train)
```

Out[18]:
```
RandomForestClassifier(max_depth=3, random_state=123)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [19]:
```
rf_model.score(X_test,y_test)
```

Out[19]:
```
0.7833333333333333
```

Model Optimization

GridSearch CV

In [20]:
```
from sklearn.model_selection import GridSearchCV
grid_search_cv = GridSearchCV(estimator = rf_model,
                              param_grid={'criterion':['gini','entropy'],
                                          'max_depth':[2,3,4,5,6]},
                              cv=5)
grid_search_cv.fit(X,y)
print(grid_search_cv.best_params_)
print(grid_search_cv.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 2}
0.7933333333333332
```

In [21]:
```
rf_model_1 = RandomForestClassifier(criterion='gini',max_depth=2,random_state=123)
rf_model_1.fit(X_train,y_train)
```

Out[21]:
```
RandomForestClassifier(max_depth=2, random_state=123)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluating rf_Model_1

In [22]:
```
rf_model_1.score(X_test,y_test)
```

Out[22]:
```
0.7833333333333333
```

In [23]:
```
y_pred_test = rf_model.predict(X_test)
```

In [24]:
```
print(confusion_matrix(y_test,y_pred_test))
```

```
[[ 0 26]
 [ 0 94]]
```

In [25]:
```
print(classification_report(y_test,y_pred_test))
```

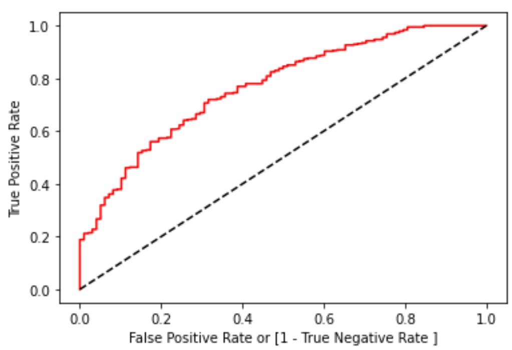
```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        26
           1       0.78      1.00      0.88        94

    accuracy                           0.78       120
   macro avg       0.39      0.50      0.44       120
weighted avg       0.61      0.78      0.69       120
```

In [26]:
```
accuracy_score(y_test,y_pred_test)
```

Out[26]: 0.7833333333333333

In [27]:
```
fpr, tpr, threshholds = roc_curve(y_train,rf_model.predict_proba (X_train)[:,1])

auc = roc_auc_score(y_train,rf_model.predict_proba (X_train)[:,1])
print(auc)

import matplotlib.pyplot as plt
plt.plot(fpr, tpr, color='red', label='Random Forest model ( area = %0.2f)'%auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate or [1 - True Negative Rate ]')
plt.ylabel('True Positive Rate')
plt.show()
```

```
0.7679372796238915
```

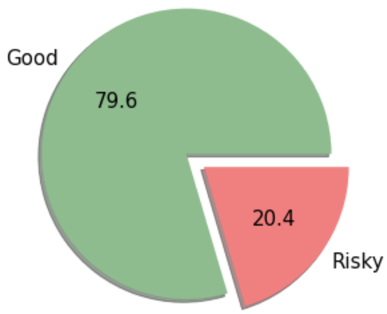


Eventhough the model has accuracy score of 0.78 the stability and specificity of the model is very low(determined from the values of precision and recall)

Data Optimization

In [28]:
```
plt.figure(figsize=(8,5))
plt.pie(x=y_train.value_counts(),labels=['Good','Risky'],explode=[0.1,0.05],
        autopct='%0.1f',colors=['darkseagreen','lightcoral'],shadow=True,textprops = {"fontsize":15})
plt.show
```

Out[28]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



The Data is imbalance

In [29]:
```
from imblearn.over_sampling import SMOTE
balanced = SMOTE()

X_balanced , y_balanced = balanced.fit_resample(X,y)
```

In [30]:
```
Optimized_Data = X_balanced.copy()
Optimized_Data['y']=y_balanced
```

```
Optimized_Data.head()
```

In [31]:
```
X_train_Opt,X_test_Opt,y_train_Opt,y_test_Opt=train_test_split(X_balanced,y_balanced, test_size=0.2,random_state=123)

print(X_train_Opt.shape)
print(y_train_Opt.shape)
```

```
(761, 5)
(761, 1)
```

In [32]:
```
RF_Model_Opt=RandomForestClassifier(criterion='gini',max_depth=3,random_state=123)
```

In [33]:
```
RF_Model_Opt.fit(X_train_Opt,y_train_Opt)
```

Out[33]:
```
RandomForestClassifier(max_depth=3, random_state=123)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [34]:
```
Optimized_Data['y_predicted']=RF_Model_Opt.predict(X_balanced)
```

In [35]:
```
accuracy_score(y_test_Opt,RF_Model_Opt.predict(X_test_Opt))
```

Out[35]:
```
0.6492146596858639
```

In [36]:
```
confusion_matrix(y_test_Opt,RF_Model_Opt.predict(X_test_Opt))
```

Out[36]:
```
array([[66, 18],
       [49, 58]], dtype=int64)
```

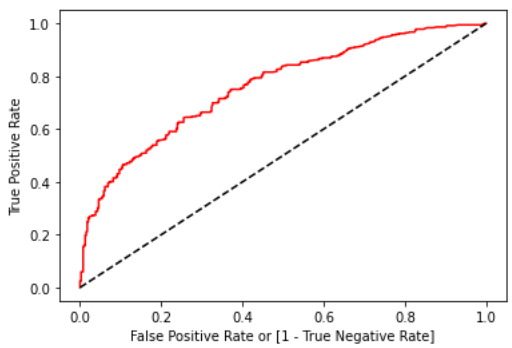
Test Data

In [37]:
```
print(classification_report(y_test_Opt,RF_Model_Opt.predict(X_test_Opt)))
```

```
              precision    recall  f1-score   support

           0       0.57      0.79      0.66        84
           1       0.76      0.54      0.63       107

    accuracy                           0.65       191
   macro avg       0.67      0.66      0.65       191
weighted avg       0.68      0.65      0.65       191
```

In [38]:
```
fpr2, tpr2, threshholds2 = roc_curve(y_train_Opt,RF_Model_Opt.predict_proba (X_train_Opt)[:,1])

auc2 = roc_auc_score(y_train_Opt,RF_Model_Opt.predict_proba (X_train_Opt)[:,1])
print(auc2)

import matplotlib.pyplot as plt
plt.plot(fpr2, tpr2, color='red', label='Random Forest Model ( area = %0.2f)'%auc2)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
plt.ylabel('True Positive Rate')
plt.show()
```

```
0.7609438084176761
```



The Model has precision and recall

Model Deployment

In [39]:
```
from pickle import dump,load
dump(RF_Model_Opt,open('Intel_On_frauddata_RF.pkl','wb'))
```

In [40]:
```
Loaded_Int=load(open('Intel_On_frauddata_RF.pkl','rb'))
Loaded_Int.predict(X_test_Opt.head())
```

Out[40]:
```
array([1, 1, 1, 1, 0], dtype=int64)
```

In [ ]: