Importing necessary Liabraries

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import GridSearchCV
```

Importing Dataset

In [2]:

```
fraud_data=pd.read_csv("Fraud_check.csv")
fraud_data.head(20)
```

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
5	NO	Divorced	33329	116382	0	NO
6	NO	Divorced	83357	80890	8	YES
7	YES	Single	62774	131253	3	YES
8	NO	Single	83519	102481	12	YES
9	YES	Divorced	98152	155482	4	YES
10	NO	Single	29732	102602	19	YES
11	NO	Single	61063	94875	6	YES
12	NO	Divorced	11794	148033	14	YES
13	NO	Married	61830	86649	16	YES
14	NO	Married	64070	57529	13	YES
15	NO	Divorced	69869	107764	29	NO
16	YES	Divorced	24987	34551	29	NO
17	YES	Married	39476	57194	25	NO
18	YES	Divorced	97957	59269	6	NO
19	NO	Single	10987	126953	30	YES

Initial investigation

In [3]:

```
fraud_data.isna().sum()
```

Out[3]:

```
Undergrad          0
Marital.Status     0
Taxable.Income     0
City.Population    0
Work.Experience    0
Urban              0
dtype: int64
```

In [4]:

```
fraud_data.info()
```

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

In [5]:

```
fraud_data.dtypes
```

Out[5]:

```
Undergrad          object
Marital.Status     object
Taxable.Income      int64
City.Population     int64
Work.Experience     int64
Urban              object
dtype: object
```

In [6]:

```
fraud_data.shape
```

Out[6]:

```
(600, 6)
```

In [7]:

```
fraud_data.describe(include='all')
```

Out[7]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
count	600	600	600.000000	600.000000	600.000000	600
unique	2	3	NaN	NaN	NaN	2
top	YES	Single	NaN	NaN	NaN	YES
freq	312	217	NaN	NaN	NaN	302
mean	NaN	NaN	55208.375000	108747.368333	15.558333	NaN
std	NaN	NaN	26204.827597	49850.075134	8.842147	NaN
min	NaN	NaN	10003.000000	25779.000000	0.000000	NaN
25%	NaN	NaN	32871.500000	66966.750000	8.000000	NaN
50%	NaN	NaN	55074.500000	106493.500000	15.000000	NaN
75%	NaN	NaN	78611.750000	150114.250000	24.000000	NaN
max	NaN	NaN	99619.000000	199778.000000	30.000000	NaN

In [8]:

```
# As per the problem statement, converted into Good and Risky
fraud_data.loc[fraud_data['Taxable.Income'] <= 30000, '<= 30000'] = 'Risky'
fraud_data.loc[fraud_data['Taxable.Income'] >30000 , '> 30000'] = 'Good'
```

In [9]:

```
fraud_data.head(50)
```

Out[9]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban	300
0	NO	Single	68833	50047	10	YES	Na
1	YES	Divorced	33700	134075	18	YES	Na
2	NO	Married	36925	160205	30	YES	Na
3	YES	Single	50190	193264	15	YES	Na
4	NO	Married	81002	27533	28	NO	Na
5	NO	Divorced	33329	116382	0	NO	Na
6	NO	Divorced	83357	80890	8	YES	Na
7	YES	Single	62774	131253	3	YES	Na
8	NO	Single	83519	102481	12	YES	Na
9	YES	Divorced	98152	155482	4	YES	Na
10	NO	Single	29732	102602	19	YES	Ris
11	NO	Single	61063	94875	6	YES	Na
12	NO	Divorced	11794	148033	14	YES	Ris
13	NO	Married	61830	86649	16	YES	Na
14	NO	Married	64070	57529	13	YES	Na
15	NO	Divorced	69869	107764	29	NO	Na
16	YES	Divorced	24987	34551	29	NO	Ris
17	YES	Married	39476	57194	25	NO	Na
18	YES	Divorced	97957	59269	6	NO	Na
19	NO	Single	10987	126953	30	YES	Ris
20	YES	Single	88636	147222	26	NO	Na
21	YES	Divorced	14310	29106	7	YES	Ris
22	YES	Divorced	78969	155342	14	NO	Na
23	NO	Single	92040	50495	12	YES	Na
24	NO	Divorced	38239	28495	30	NO	Na
25	NO	Divorced	31417	124606	27	YES	Na
26	YES	Divorced	55299	169128	15	NO	Na
27	YES	Single	87778	28542	12	YES	Na
28	YES	Single	10379	128766	5	YES	Ris
29	YES	Divorced	94033	41863	30	YES	Na
30	YES	Divorced	73854	117788	0	YES	Na
31	NO	Divorced	64007	147414	21	NO	Na
32	YES	Married	97200	51911	23	NO	Na
33	YES	Single	82071	157251	21	NO	Na

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban	300
34	YES	Divorced	12514	183767	1	YES	Ris
35	YES	Married	31336	41117	30	YES	Na
36	YES	Married	10163	193995	5	YES	Ris
37	NO	Married	68513	66912	5	YES	Na
38	NO	Single	14912	177575	3	NO	Ris
39	NO	Married	74010	54981	16	YES	Na
40	NO	Single	50777	199697	26	YES	Na
41	YES	Married	49436	91524	1	NO	Na
42	NO	Single	96485	51666	12	NO	Na
43	YES	Divorced	70339	50020	10	NO	Na
44	YES	Divorced	33614	98880	22	NO	Na
45	YES	Married	81079	183095	14	YES	Na
46	YES	Married	31532	137346	27	YES	Na
47	YES	Single	44034	34964	2	NO	Na
48	NO	Married	16264	35480	12	NO	Ris
49	NO	Divorced	45706	160195	15	YES	Na

In [10]:

```
#Checking the values of count in Undergrad column
fraud_data['Undergrad'].value_counts()
```

Out[10]:

```
YES    312
NO     288
Name: Undergrad, dtype: int64
```

In [11]:

```
# Checking the correlation
fraud_data.corr()['Taxable.Income'].sort_values()
```

Out[11]:

```
City.Population   -0.064387
Work.Experience   -0.001818
Taxable.Income     1.000000
Name: Taxable.Income, dtype: float64
```

In [12]:

```
sns.heatmap(fraud_data.corr(),annot=True)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x263a12702c8>
```

Data Preprocesing

In [13]:

```
#Deletting Taxable.Income due to we converted as output Risky and Good
del fraud_data['Taxable.Income']
```

In [14]:

```
# Deletting because, need only one column instead of two. also want to get dummies for fitt
del fraud_data['<= 30000']
```

In [15]:

```
fraud_data.head(10)
```

Out[15]:

	Undergrad	Marital.Status	City.Population	Work.Experience	Urban	> 30000
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
5	NO	Divorced	116382	0	NO	Good
6	NO	Divorced	80890	8	YES	Good
7	YES	Single	131253	3	YES	Good
8	NO	Single	102481	12	YES	Good
9	YES	Divorced	155482	4	YES	Good

In [16]:

```
#Filling nan values by 0
fraud_data.fillna(value=0,axis=0,inplace=True)
```

In [17]:

```
fraud_data.head(50)
```

Out[17]:

	Undergrad	Marital.Status	City.Population	Work.Experience	Urban	> 30000
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
5	NO	Divorced	116382	0	NO	Good
6	NO	Divorced	80890	8	YES	Good
7	YES	Single	131253	3	YES	Good
8	NO	Single	102481	12	YES	Good
9	YES	Divorced	155482	4	YES	Good
10	NO	Single	102602	19	YES	0
11	NO	Single	94875	6	YES	Good
12	NO	Divorced	148033	14	YES	0
13	NO	Married	86649	16	YES	Good
14	NO	Married	57529	13	YES	Good
15	NO	Divorced	107764	29	NO	Good
16	YES	Divorced	34551	29	NO	0
17	YES	Married	57194	25	NO	Good
18	YES	Divorced	59269	6	NO	Good
19	NO	Single	126953	30	YES	0
20	YES	Single	147222	26	NO	Good
21	YES	Divorced	29106	7	YES	0
22	YES	Divorced	155342	14	NO	Good
23	NO	Single	50495	12	YES	Good
24	NO	Divorced	28495	30	NO	Good
25	NO	Divorced	124606	27	YES	Good
26	YES	Divorced	169128	15	NO	Good
27	YES	Single	28542	12	YES	Good
28	YES	Single	128766	5	YES	0
29	YES	Divorced	41863	30	YES	Good
30	YES	Divorced	117788	0	YES	Good
31	NO	Divorced	147414	21	NO	Good
32	YES	Married	51911	23	NO	Good
33	YES	Single	157251	21	NO	Good

	Undergrad	Marital.Status	City.Population	Work.Experience	Urban	> 30000
34	YES	Divorced	183767	1	YES	0
35	YES	Married	41117	30	YES	Good
36	YES	Married	193995	5	YES	0
37	NO	Married	66912	5	YES	Good
38	NO	Single	177575	3	NO	0
39	NO	Married	54981	16	YES	Good
40	NO	Single	199697	26	YES	Good
41	YES	Married	91524	1	NO	Good
42	NO	Single	51666	12	NO	Good
43	YES	Divorced	50020	10	NO	Good
44	YES	Divorced	98880	22	NO	Good
45	YES	Married	183095	14	YES	Good
46	YES	Married	137346	27	YES	Good
47	YES	Single	34964	2	NO	Good
48	NO	Married	35480	12	NO	0
49	NO	Divorced	160195	15	YES	Good

In [18]:

```
#getting dummies
fraud_data['Tax.Income']=pd.get_dummies(fraud_data['> 30000'],drop_first=True)
```

In [19]:

```
fraud_data.shape
```

Out[19]:

```
(600, 7)
```

In [20]:

```
fraud_data[["Undergrad","Urban"]]=pd.get_dummies(fraud_data[["Undergrad","Urban"]],drop_fir
```

In [21]:

```
fraud_data['Marital.Status']=pd.get_dummies(fraud_data['Marital.Status'],drop_first=True)
```

In [22]:

```
fraud_data.head(20)
```

Out[22]:

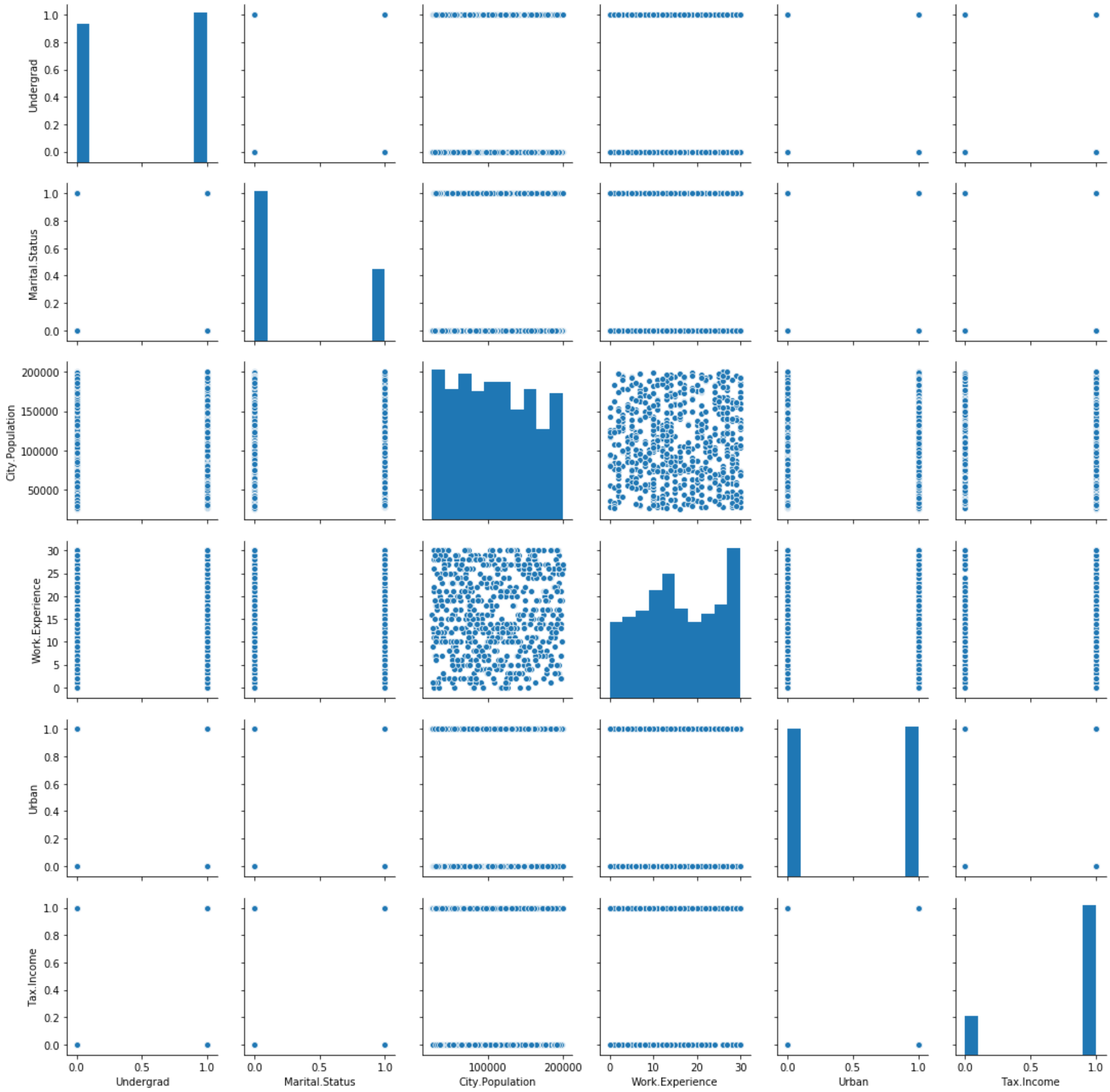
	Undergrad	Marital.Status	City.Population	Work.Experience	Urban	> 30000	Tax.Income
0	0	0	50047	10	1	Good	1
1	1	0	134075	18	1	Good	1
2	0	1	160205	30	1	Good	1
3	1	0	193264	15	1	Good	1
4	0	1	27533	28	0	Good	1
5	0	0	116382	0	0	Good	1
6	0	0	80890	8	1	Good	1
7	1	0	131253	3	1	Good	1
8	0	0	102481	12	1	Good	1
9	1	0	155482	4	1	Good	1
10	0	0	102602	19	1	0	0
11	0	0	94875	6	1	Good	1
12	0	0	148033	14	1	0	0
13	0	1	86649	16	1	Good	1
14	0	1	57529	13	1	Good	1
15	0	0	107764	29	0	Good	1
16	1	0	34551	29	0	0	0
17	1	1	57194	25	0	Good	1
18	1	0	59269	6	0	Good	1
19	0	0	126953	30	1	0	0

In [23]:

```
sns.pairplot(fraud_data)
```

Out[23]:

```
<seaborn.axisgrid.PairGrid at 0x263a13b2ec8>
```



In [24]:

```
#scaling the data
scaled_data=StandardScaler()
x_scale=scaled_data.fit_transform(fraud_data.iloc[:, :-2])
```

Model Building

In [25]:

```
X=fraud_data.iloc[:, :-2]
y=fraud_data[['Tax.Income']]
```

In [26]:

```
X
```

Out[26]:

	Undergrad	Marital.Status	City.Population	Work.Experience	Urban
0	0	0	50047	10	1
1	1	0	134075	18	1
2	0	1	160205	30	1
3	1	0	193264	15	1
4	0	1	27533	28	0
...	...	...	...	...	...
595	1	0	39492	7	1
596	1	0	55369	2	1
597	0	0	154058	0	1
598	1	1	180083	17	0
599	0	0	158137	16	0

600 rows × 5 columns

In [27]:

```
X_train,X_test,y_train,y_test=train_test_split(x_scale,y,test_size=0.20,random_state=12)
```

Model Training

In [28]:

```
dt_model=DecisionTreeClassifier()
dt_model.fit(X_train,y_train)
```

Out[28]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [29]:

```
X_train
```

Out[29]:

```
array([[-1.040833  , -0.6912543 ,  1.17762255, -0.96870973,  0.99335541],
       [-1.040833  , -0.6912543 , -0.267552  , -0.96870973,  0.99335541],
       [-1.040833  , -0.6912543 ,  0.44306945, -0.28957535, -1.00668904],
       ...,
       [ 0.96076892, -0.6912543 , -1.55335622,  0.27636996,  0.99335541],
       [ 0.96076892, -0.6912543 , -0.63224868,  1.52144966, -1.00668904],
       [ 0.96076892, -0.6912543 , -1.03242121,  0.1631809 , -1.00668904]])
```

In [30]:

```
y_pred_train=dt_model.predict(X_train)
y_pred_test=dt_model.predict(X_test)
```

Model Evaluation| Model Test Accuracy

In [31]:

```
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
print(accuracy_score(y_test,y_pred_test))
```

```
[[ 2 11]
 [32 75]]
              precision    recall  f1-score   support

           0       0.06      0.15      0.09        13
           1       0.87      0.70      0.78       107

    accuracy                           0.64       120
   macro avg       0.47      0.43      0.43       120
weighted avg       0.78      0.64      0.70       120

0.6416666666666667
```

Hyperparmeter GridSearchCV

In [32]:

```
#Using GridSearchCv for getting best hyperparameter
grid_model = GridSearchCV(estimator = dt_model,param_grid = {'criterion' :['gini','entropy'
                                                              'max_depth' :[3,5,7,8,10],
                                                              'min_samples_split' :[2,3,4]})

grid_model.fit(X_train,y_train)
print(grid_model.best_params_)
print(grid_model.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 3}
0.7666666666666667
```

In [33]:

```
#fitting into model as per GridSearchCV
dt_model1=DecisionTreeClassifier(criterion='gini',max_depth=5,min_samples_split=3)
dt_model1.fit(X_train,y_train)
```

Out[33]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=4,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [35]:

```
y_pred_train1=dt_model1.predict(X_train)
y_pred_test1=dt_model1.predict(X_test)
```

In [36]:

```
print(confusion_matrix(y_test,y_pred_test1))
print(classification_report(y_test,y_pred_test1))
print(accuracy_score(y_test,y_pred_test1))
```

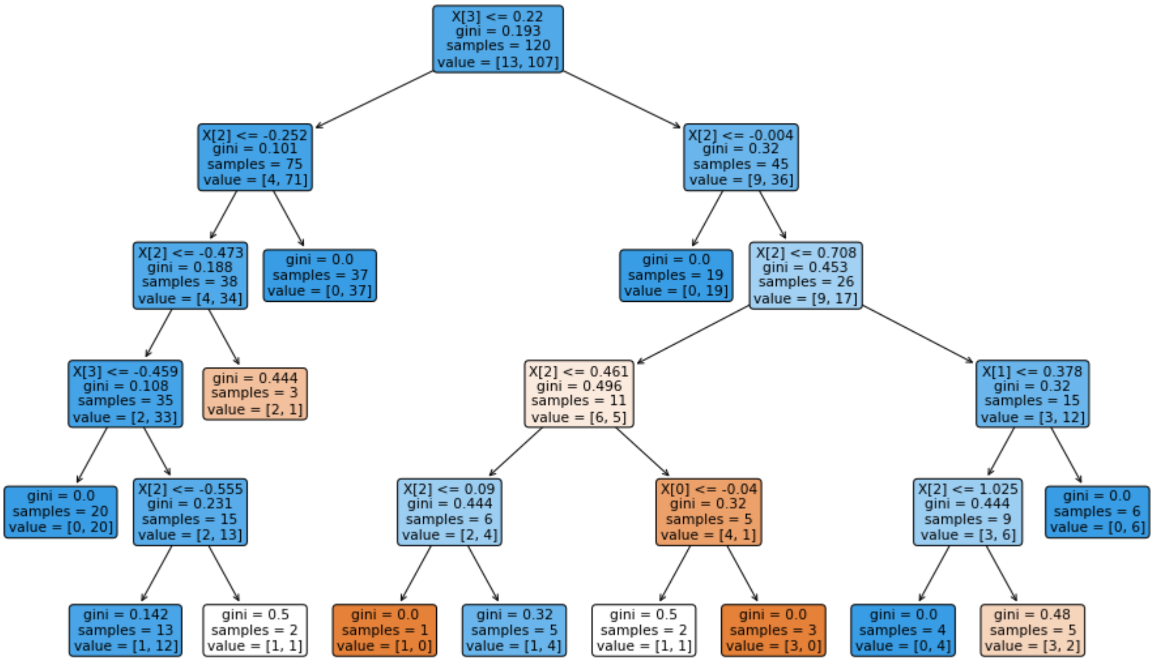
```
[[  0  13]
 [  6 101]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        13
           1       0.89      0.94      0.91       107

    accuracy                           0.84       120
   macro avg       0.44      0.47      0.46       120
weighted avg       0.79      0.84      0.82       120

0.8416666666666667
```

In [41]:

```
#True positive rate is decreased
```

In [38]:

```
model_all_params_max_depth_5 = DecisionTreeClassifier(max_depth = 5,min_samples_split=4,rar
# Prepare a plot figure with set size.
plt.figure(figsize = (16,10))
# Plot the decision tree.
plot_tree(model_all_params_max_depth_5,
          rounded = True,
          filled = True           )
# Display the tree plot figure.
plt.show()
```



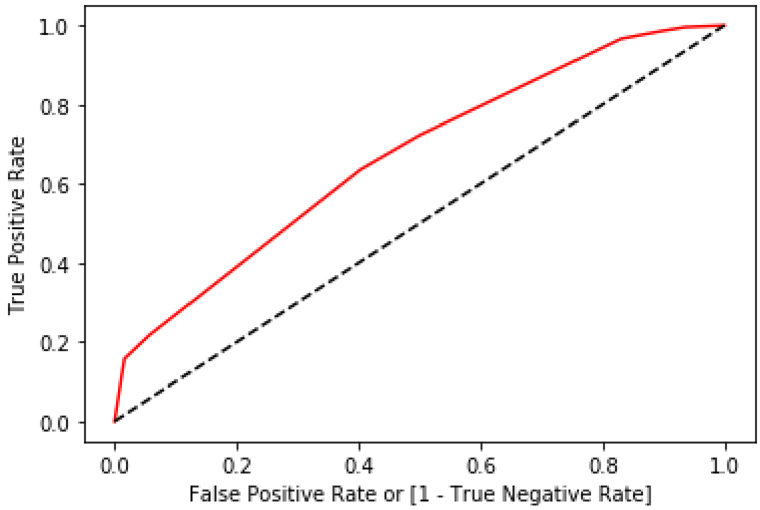
In [40]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

fpr, tpr, thresholds = roc_curve(y, dt_model1.predict_proba (x_scale)[:,1])

auc = roc_auc_score(y_test, y_pred_test1)
print(auc)

import matplotlib.pyplot as plt
plt.plot(fpr, tpr, color='red', label='dt_model1 ( 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')
```

0.4719626168224299

Out[40]:

Text(0, 0.5, 'True Positive Rate')



In [42]:

```
#Here, The true positive value is low,it means that fraudlent probabilty in the data is low
```