FRAUD CHECK

In [1]:
```
#Import the libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier #importing decision tree classifi
from sklearn.model_selection import train_test_split #importing train_test_split
from sklearn.metrics import accuracy_score#importing metrics for accuracy calcula
from sklearn.ensemble import BaggingClassifier#bagging combines the results of mu
from sklearn.metrics import classification_report, confusion_matrix
```

In [2]:
```
#Import the dataset
#reading the dataset
Fraud_check=pd.read_csv("C:\\Users\\nishi\\Desktop\\Assignments\\Decision_Trees\\
```

In [3]:
```
Fraud_check
```

Out[3]:

	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 [4]:
```
Fraud_check.head()
```

Out[4]:

	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

In [5]:
```
#viewing the types
Fraud_check.dtypes
```

Out[5]:
```
Undergrad          object
Marital.Status     object
Taxable.Income      int64
City.Population     int64
Work.Experience     int64
Urban              object
dtype: object
```

In [6]:
```
Fraud_check.rename(columns={'Work.Experience':'Work_Experience',"Taxable.Income":
```

In [7]:
```
Fraud_check
```

Out[7]:

	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 [8]:
```
#------------------converting categorical data---------------------------------
Fraud_check['Risky_1'] = Fraud_check.Taxable_Income.map(lambda x: 1 if x <= 30000
Fraud_check['Undergrad']=Fraud_check['Undergrad'].astype('category')
Fraud_check['Marital_Status']=Fraud_check['Marital_Status'].astype('category')
Fraud_check['Urban']=Fraud_check['Urban'].astype('category')
Fraud_check.dtypes
Fraud_check.head()
```

Out[8]:

	Undergrad	Marital_Status	Taxable_Income	City_Population	Work_Experience	Urban	Risky_1
0	NO	Single	68833	50047	10	YES	0
1	YES	Divorced	33700	134075	18	YES	0
2	NO	Married	36925	160205	30	YES	0
3	YES	Single	50190	193264	15	YES	0
4	NO	Married	81002	27533	28	NO	0

In [9]:
```
#label encoding to convert categorical values into numeric.
Fraud_check['Undergrad']=Fraud_check['Undergrad'].cat.codes
Fraud_check['Urban']=Fraud_check['Urban'].cat.codes
Fraud_check['Marital_Status']=Fraud_check['Marital_Status'].cat.codes
Fraud_check.head()
```

Out[9]:

	Undergrad	Marital_Status	Taxable_Income	City_Population	Work_Experience	Urban	Risky_1
0	0	2	68833	50047	10	1	0
1	1	0	33700	134075	18	1	0
2	0	1	36925	160205	30	1	0
3	1	2	50190	193264	15	1	0
4	0	1	81002	27533	28	0	0

In [10]:
```
#----------------------- setting feature and target variables ------------------
feature_cols=['Undergrad','Marital_Status','City_Population','Work_Experience','U
#x = fraud.drop(['Taxable_Income','Risky_1'], axis=1)
x = Fraud_check[feature_cols]
y = Fraud_check.Risky_1
print(x)
print(y)
```

```
     Undergrad  Marital_Status  City_Population  Work_Experience  Urban
0            0               2            50047               10      1
1            1               0           134075               18      1
2            0               1           160205               30      1
3            1               2           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 x 5 columns]
0      0
1      0
2      0
3      0
4      0
      ..
595    0
596    0
597    0
598    0
599    0
Name: Risky_1, Length: 600, dtype: int64
```

In [11]:
```
#----------------------- splitting into train and test data --------------------
x_train,x_test,y_train,y_test= train_test_split(x,y, test_size=0.20,random_state=
```

In [12]:
```
#-------------------------building decision tree model---------------------------
Fraud_model =  BaggingClassifier(DecisionTreeClassifier(max_depth = 6), random_st
fraudmodel = Fraud_model.fit(x_train,y_train) #train decision tree
y_predict = Fraud_model.predict(x_test)
```

In [13]:
```
#-----------Finding the accuracy-----------------------------------#
print("Accuracy : ", accuracy_score(y_test,y_predict)*100 )
print(confusion_matrix(y_test,y_predict))
print(classification_report(y_test,y_predict))
```

```
Accuracy :  80.0
[[96  1]
 [23  0]]
              precision    recall  f1-score   support

           0       0.81      0.99      0.89        97
           1       0.00      0.00      0.00        23

    accuracy                           0.80       120
   macro avg       0.40      0.49      0.44       120
weighted avg       0.65      0.80      0.72       120
```

COMPANY DATA

In [63]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier as DT
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics
from sklearn import externals
from io import StringIO
import pydotplus
from sklearn.metrics import accuracy_score
import matplotlib.image as mpimg
from sklearn.tree import export_graphviz


Company_data = pd.read_csv("C:\\Users\\nishi\\Desktop\\Assignments\\Decision_Tree
Company_data.head()
```

Out[63]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban
0	9.50	138	73	11	276	120	Bad	42	17	Yes
1	11.22	111	48	16	260	83	Good	65	10	Yes
2	10.06	113	35	10	269	80	Medium	59	12	Yes
3	7.40	117	100	4	466	97	Medium	55	14	Yes
4	4.15	141	64	3	340	128	Bad	38	13	Yes

In [64]:
```
corr=Company_data.corr()
```

In [65]:
```
Company_data=pd.get_dummies(Company_data,columns=['ShelveLoc','Urban','US'])
```

In [66]:
```
corr=Company_data.corr()
```

In [67]:
```
plt.figure(figsize=(10,10))
sns.heatmap(corr,annot=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-67-4a59abe243aa> in <module>
      1 plt.figure(figsize=(10,10))
----> 2 sns.heatmap(corr,annot=True)

NameError: name 'sns' is not defined


<Figure size 720x720 with 0 Axes>
```

In [68]:
```
Company_data['Sales'].mean()
```

Out[68]:
```
7.496325
```

In [69]:
```
# Sales variable is continuous, we take values <=7.49 as "less" and >7.49 as "mor
Company_data["sales"]="less"
Company_data.loc[Company_data["Sales"]>7.49,"sales"]="more"
Company_data.drop(["Sales"],axis=1,inplace=True)
```

In [70]:
```
Company_data.isnull().any()#to check if we have null valuesin dataset or not
#so there are no null values in the dataset
```

Out[70]:
```
CompPrice          False
Income             False
Advertising        False
Population         False
Price              False
Age                False
Education          False
ShelveLoc_Bad      False
ShelveLoc_Good     False
ShelveLoc_Medium   False
Urban_No           False
Urban_Yes          False
US_No              False
US_Yes             False
sales              False
dtype: bool
```

In [71]:
```
Company_data.dtypes# to check the data types
```

Out[71]:
```
CompPrice            int64
Income               int64
Advertising          int64
Population           int64
Price                int64
Age                  int64
Education            int64
ShelveLoc_Bad        uint8
ShelveLoc_Good       uint8
ShelveLoc_Medium     uint8
Urban_No             uint8
Urban_Yes            uint8
US_No                uint8
US_Yes               uint8
sales               object
dtype: object
```

In [72]:
```
Company_data.describe()### to check the summary of the dataset
```

Out[72]:

	CompPrice	Income	Advertising	Population	Price	Age	Education	Shelv
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	4
mean	124.975000	68.657500	6.635000	264.840000	115.795000	53.322500	13.900000	
std	15.334512	27.986037	6.650364	147.376436	23.676664	16.200297	2.620528	
min	77.000000	21.000000	0.000000	10.000000	24.000000	25.000000	10.000000	
25%	115.000000	42.750000	0.000000	139.000000	100.000000	39.750000	12.000000	
50%	125.000000	69.000000	5.000000	272.000000	117.000000	54.500000	14.000000	
75%	135.000000	91.000000	12.000000	398.500000	131.000000	66.000000	16.000000	
max	175.000000	120.000000	29.000000	509.000000	191.000000	80.000000	18.000000	

In [73]:
```
Company_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   CompPrice         400 non-null    int64
 1   Income            400 non-null    int64
 2   Advertising       400 non-null    int64
 3   Population        400 non-null    int64
 4   Price             400 non-null    int64
 5   Age               400 non-null    int64
 6   Education         400 non-null    int64
 7   ShelveLoc_Bad     400 non-null    uint8
 8   ShelveLoc_Good    400 non-null    uint8
 9   ShelveLoc_Medium  400 non-null    uint8
 10  Urban_No          400 non-null    uint8
 11  Urban_Yes         400 non-null    uint8
 12  US_No             400 non-null    uint8
 13  US_Yes            400 non-null    uint8
 14  sales             400 non-null    object
dtypes: int64(7), object(1), uint8(7)
memory usage: 27.9+ KB
```

In [74]:
```
X=Company_data.iloc[:,0:14]
Y=Company_data.iloc[:,14]
```

In [75]:
```
Y
```

Out[75]:
```
0      more
1      more
2      more
3      less
4      less
       ...
395    more
396    less
397    less
398    less
399    more
Name: sales, Length: 400, dtype: object
```

In [76]:
```
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,stratify=Y)
```

In [77]:
```
y_train.value_counts()
```

Out[77]:
```
less    161
more    159
Name: sales, dtype: int64
```

In [78]:
```
model=DT(criterion="entropy")
model.fit(x_train,y_train)
```

Out[78]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [79]:
```
pred_train=model.predict(x_train)
```

In [80]:
```
accuracy_score(y_train,pred_train)
```

Out[80]:
```
1.0
```

In [81]:
```
confusion_matrix(y_train,pred_train)
```

Out[81]:
```
array([[161,   0],
       [  0, 159]], dtype=int64)
```

In [82]:
```
pred_test=model.predict(x_test)
```

In [83]:
```
accuracy_score(y_test,pred_test)
```

Out[83]:
```
0.85
```

In [84]:
```
confusion_matrix(y_test,pred_test)
```

Out[84]:
```
array([[36,  4],
       [ 8, 32]], dtype=int64)
```

In [85]:
```
Company_data_t=pd.DataFrame({'Actual':y_test,'predicted':pred_test})
```

In [86]:
```
Company_data_t
```

Out[86]:

	Actual	predicted
252	more	less
113	less	more
117	more	more
194	less	less
139	more	more
...	...	...
354	less	less
28	less	less
175	more	less
210	less	less
257	more	more

80 rows × 2 columns

In [87]:
```
columns=list(Company_data.columns)
```

In [88]:
```
predictors=columns[0:14]
target=columns[14]
```

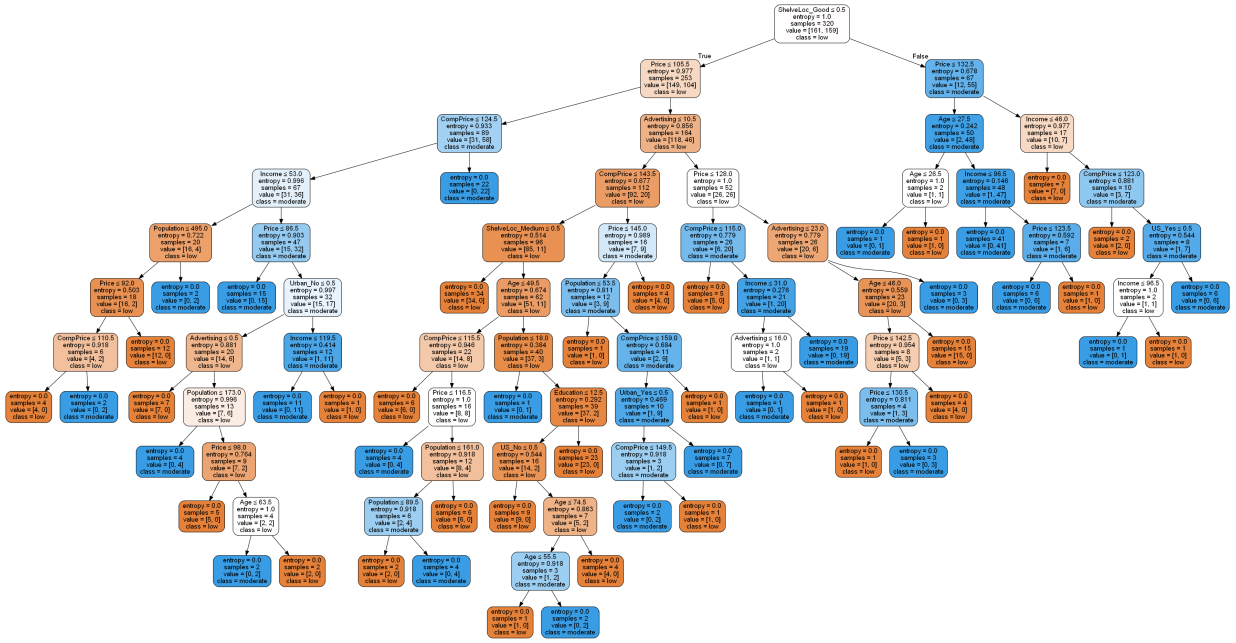
In [89]:
```
dot_data=StringIO()
```

In [90]:
```
export_graphviz(model, out_file=dot_data,filled=True, rounded=True,special_charac
```

In [91]:
```
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
```

In [92]:
```
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Sales.png')
Image(graph.create_png())
```

Out[92]:



In [93]:
```
model.feature_importances_
```

Out[93]:
```
array([0.19310425, 0.09772033, 0.08338138, 0.08959172, 0.28686097,
       0.0812294 , 0.00838736, 0.        , 0.08550806, 0.0234122 ,
       0.02912005, 0.00604726, 0.00829769, 0.00733932])
```

In [94]:
```
fea=pd.DataFrame({'feature':list(x_train.columns),'importance':model.feature_impo
sort_values('importance',ascending=False)
```

In [95]:
```
fea
```

Out[95]:

	feature	importance
4	Price	0.286861
0	CompPrice	0.193104
1	Income	0.097720
3	Population	0.089592
8	ShelveLoc_Good	0.085508
2	Advertising	0.083381
5	Age	0.081229
10	Urban_No	0.029120
9	ShelveLoc_Medium	0.023412
6	Education	0.008387
12	US_No	0.008298
13	US_Yes	0.007339
11	Urban_Yes	0.006047
7	ShelveLoc_Bad	0.000000

In [ ]:
```

```

In [ ]:
```

```