Assignment-14-Decision-Tree

Q1). Use decision trees to prepare a model on fraud data treating those who have taxable_income <= 30000 as "Risky" and others are "Good"

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import  DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import classification_report
from sklearn import preprocessing
```

In [2]:

```
data=pd.read_csv("C:/Users/LENOVO/Documents/Custom Office Templates/Fraud_check.csv")
data.head()
```

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

In [3]:

```
data.shape
```

Out[3]:

```
(600, 6)
```

In [4]:

```
data.columns
```

Out[4]:

```
Index(['Undergrad', 'Marital.Status', 'Taxable.Income', 'City.Population',
       'Work.Experience', 'Urban'],
      dtype='object')
```

In [5]:

```
data.describe()
```

Out[5]:

	Taxable.Income	City.Population	Work.Experience
count	600.000000	600.000000	600.000000
mean	55208.375000	108747.368333	15.558333
std	26204.827597	49850.075134	8.842147
min	10003.000000	25779.000000	0.000000
25%	32871.500000	66966.750000	8.000000
50%	55074.500000	106493.500000	15.000000
75%	78611.750000	150114.250000	24.000000
max	99619.000000	199778.000000	30.000000

In [6]:

```
data.loc[data['Taxable.Income']>=30000, 'Income']='Good'
data.loc[data['Taxable.Income']<=30000, 'Income']='Risky'
```

In [7]:

```
data
```

Out[7]:

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

600 rows × 7 columns

In [8]:

```
label_encoder= preprocessing.LabelEncoder()
data['Income']= label_encoder.fit_transform(data['Income'])
data['Undergrad']= label_encoder.fit_transform(data['Undergrad'])
data['Urban']= label_encoder.fit_transform(data['Urban'])
data['Marital.Status']= label_encoder.fit_transform(data['Marital.Status'])
```

In [9]:

```
data
```

Out[9]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban	Incom
0	0	2	68833	50047	10	1	
1	1	0	33700	134075	18	1	
2	0	1	36925	160205	30	1	
3	1	2	50190	193264	15	1	
4	0	1	81002	27533	28	0	
...	...	...	...	...	...	...	.
595	1	0	76340	39492	7	1	
596	1	0	69967	55369	2	1	
597	0	0	47334	154058	0	1	
598	1	1	98592	180083	17	0	
599	0	0	96519	158137	16	0	

600 rows × 7 columns

In [10]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Undergrad        600 non-null    int32
 1   Marital.Status   600 non-null    int32
 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    int32
 6   Income           600 non-null    int32
dtypes: int32(4), int64(3)
memory usage: 23.6 KB
```

In [11]:

```
#Droping the Taxable income variable - we taken that as Income description
data.drop(["Taxable.Income"],axis=1,inplace=True)
```

In [12]:

```
data.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    int32
 1   Marital.Status   600 non-null    int32
 2   City.Population  600 non-null    int64
 3   Work.Experience  600 non-null    int64
 4   Urban            600 non-null    int32
 5   Income           600 non-null    int32
dtypes: int32(4), int64(2)
memory usage: 18.9 KB
```

No Null values

In [13]:

```
data['Income'].value_counts()
```

Out[13]:

```
0    476
1    124
Name: Income, dtype: int64
```

In [14]:

```
#diving data for traing and test
x = data.iloc[:,0:5]
y = data.iloc[:,5]
```

In [15]:

```
x
```

Out[15]:

	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 × 5 columns

In [16]:

```
y
```

Out[16]:

```
0      0
1      0
2      0
3      0
4      0
      ..
595    0
596    0
597    0
598    0
599    0
Name: Income, Length: 600, dtype: int32
```

In [17]:

```
data['Income'].unique()
```

Out[17]:

```
array([0, 1])
```

In [18]:

```
# Splitting data into training and testing data
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.20, random_state=40)
```

In [19]:

```
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

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

Building Decision Tree Clasifier using Entropy Criteria

In [20]:

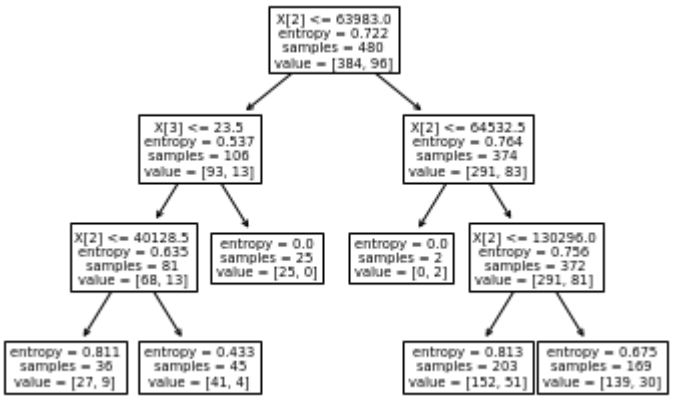
```
model= DecisionTreeClassifier(criterion='entropy', max_depth=3)
model.fit(x_train, y_train)
```

Out[20]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

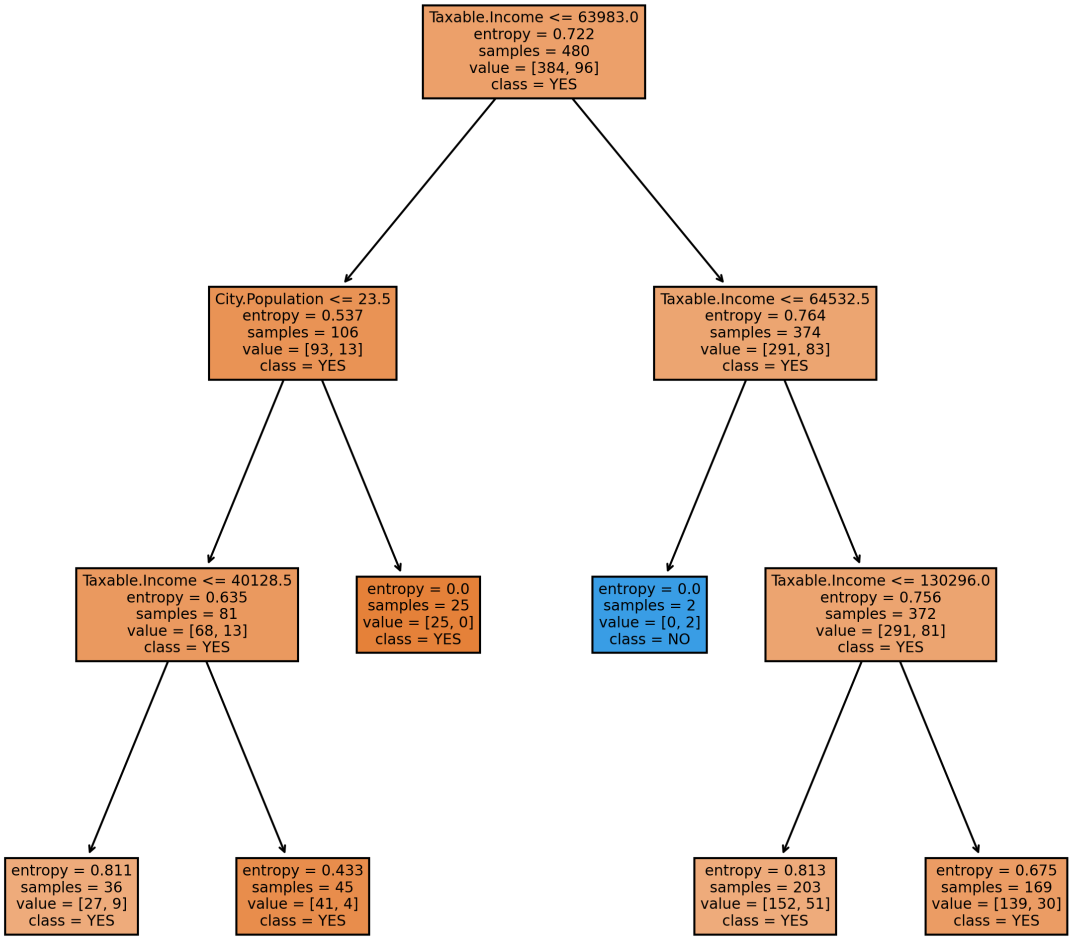
In [21]:

```
tree.plot_tree(model);
```



In [22]:

```
fn=['Undergrad', 'Marital.Status', 'Taxable.Income', 'City.Population',
       'Work.Experience', 'Urban']
cn=['YES','NO']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (10,10), dpi=300)
tree.plot_tree(model,
               feature_names = fn,
               class_names=cn,
               filled = True);
```



In [23]:

```
model.fit(x_train, y_train)
model.score(x_train, y_train)
```

Out[23]:

```
0.8041666666666667
```

In [24]:

```
model.feature_importances_
```

Out[24]:

```
array([0.        , 0.        , 0.71003667, 0.28996333, 0.        ])
```

In [25]:

```
y_pred= model.predict(x_test)
```

In [26]:

```
# Here first column is for 'Good' , & second is for 'Risky'

model.predict_proba(x_test)
```

Out[26]:

```
array([[0.75      , 0.25      ],
       [1.        , 0.        ],
       [0.74876847, 0.25123153],
       [0.82248521, 0.17751479],
       [0.74876847, 0.25123153],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [0.74876847, 0.25123153],
       [0.91111111, 0.08888889],
       [0.91111111, 0.08888889],
       [0.82248521, 0.17751479],
       [0.75      , 0.25      ],
       [0.82248521, 0.17751479],
       [0.91111111, 0.08888889],
       [0.82248521, 0.17751479],
       [0.82248521, 0.17751479],
       [0.82248521, 0.17751479],
       [0.82248521, 0.17751479],
```

Building Decision Tree Classifier(CART) using Gini Criteria

In [27]:

```
from sklearn.tree import DecisionTreeClassifier
model_gini = DecisionTreeClassifier(criterion = 'gini', max_depth=3)
```

In [28]:

```
model_gini.fit(x_train, y_train)
```

Out[28]:

```
DecisionTreeClassifier(max_depth=3)
```

In [29]:

```
# Prediction and computing the accuracy

pred = model.predict(x_test)
pred
np.mean(pred == y_test)
```

Out[29]:

```
0.7666666666666667
```

In [30]:

```
# Decision Tree Regression Example
from sklearn.tree import DecisionTreeRegressor
```

In [31]:

```
array = data.values
X = array[:,0:3]
Y = array[:,3]
```

In [32]:

```
Y
```

Out[32]:

```
array([10, 18, 30, 15, 28,  0,  8,  3, 12,  4, 19,  6, 14, 16, 13, 29, 29,
       25,  6, 30, 26,  7, 14, 12, 30, 27, 15, 12,  5, 30,  0, 21, 23, 21,
        1, 30,  5,  5,  3, 16, 26,  1, 12, 10, 22, 14, 27,  2, 12, 15,  3,
       30, 23,  7, 23, 16, 25, 13, 15, 11, 23,  8, 13,  9, 15, 29, 12, 24,
        5, 29,  3, 22, 27,  2, 17, 19, 11, 26,  0,  7, 26, 24, 26, 18,  4,
       12, 29, 23, 24, 20, 29, 10, 13,  3,  6, 28, 17, 19, 18, 12, 25, 20,
       23,  5, 14, 15, 20,  6, 10,  8, 19, 11, 26,  7,  7, 21, 12, 29, 10,
       30, 11, 16, 26, 26, 13, 11,  3, 28, 27, 26,  7, 16, 12, 14, 28, 19,
        9,  6, 25, 20,  4, 23, 26, 11, 29,  2, 28, 28, 21, 25, 28, 10, 16,
       28, 29,  7,  3,  5, 11,  1,  0, 30, 21, 15, 28, 23, 22, 25, 30,  6,
       10, 24, 23, 16,  8, 13, 28, 13, 15, 22, 19, 20,  4, 30, 15,  0, 19,
        5, 10, 11, 19,  7, 10, 25, 28, 13, 21,  4, 28,  9, 17, 27,  0,  4,
       14, 20, 16, 28,  3, 25, 13,  8, 14, 13, 16, 22, 22, 19, 19,  9, 30,
       14, 27, 17, 25,  0, 28, 17,  9, 10, 14,  2,  2, 19, 14,  9,  3,  3,
        9, 26, 30, 18,  2, 10, 27, 24, 25,  1, 17,  0, 16,  7, 28, 29, 26,
        8, 14, 17, 28,  7, 28,  7, 21, 21,  5,  4, 17, 11,  3, 13,  5, 20,
        1, 20, 25,  0, 17, 11, 11, 12,  1,  7, 29, 22, 22, 10,  6,  1, 21,
       14, 30, 26,  1, 21,  2, 10, 26, 15, 21,  7, 30, 11, 27, 13, 23, 29,
       12, 25, 30, 20, 22, 19, 18,  2,  4, 19, 25, 23, 14, 24, 11,  6, 24,
       13, 15, 16, 19,  7, 12, 10, 29, 15,  1, 13, 13, 15,  5,  7, 23, 10,
       21,  8,  7, 17,  5,  6, 30,  0,  2, 30, 25, 12, 16, 29, 27, 17, 27,
       28, 24, 11,  5, 20, 17,  7, 17, 10, 30, 24, 14, 28,  5,  5,  9, 21,
        9,  2, 26, 30,  6, 13,  2,  4, 29,  5, 25, 27, 27,  9,  4,  5, 17,
        1, 15, 30, 21,  5, 21, 14,  2, 21, 16, 20, 11, 27, 21, 28,  2,  6,
       22,  1, 27, 19, 11, 11,  1, 21, 30,  2, 18, 20,  2, 13, 14, 19,  0,
       27, 25, 10, 24, 30,  8, 25, 11, 14, 19,  4,  4, 20, 14,  8, 13, 24,
       27,  9, 14, 12, 28,  5, 28, 11,  8, 10, 10, 26, 15, 23, 23, 24, 18,
       21, 10,  6, 17,  3, 16, 21, 16, 20, 18, 13, 26, 21, 11, 16,  0, 12,
        7, 29,  6, 21, 26, 13, 12, 10, 28, 23, 29, 13, 10,  6, 24, 10, 18,
       29, 13, 24,  5,  3, 24,  2, 11,  4,  1, 17, 19, 15,  8, 24, 19, 27,
       14,  5, 22, 10,  6, 13, 18, 26, 26, 25,  8, 15, 10, 12, 27, 23, 12,
       27, 10, 14, 17, 14, 16, 19, 13,  3, 14,  6, 12,  8, 22, 30, 27, 13,
       21,  8,  6, 25,  9, 11,  2,  0,  5, 13, 27, 10, 12, 10, 12, 26, 24,
       29, 12,  6, 11, 28, 28, 30, 22, 12, 29,  7, 13, 20, 29, 13, 15,  5,
        6,  7,  1, 10, 12, 19,  4,  2, 30, 26, 27, 10, 14, 18, 16, 10,  9,
        7,  2,  0, 17, 16], dtype=int64)
```

In [33]:

```
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.33, random_state=1)
```

In [34]:

```
model1 = DecisionTreeRegressor()
model.fit(X_train, y_train)
```

Out[34]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [35]:

```
# Find the accuracy
model.score(X_test, y_test)
```

Out[35]:

```
0.7878787878787878
```

In [ ]:

```

```

Q2.) A decision tree can be built with target variable Sale (we will first convert it in categorical variable) & all other variable will be independent in the analysis.

In [36]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import classification_report
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
```

In [37]:

```
company=pd.read_csv("C:/Users/LENOVO/Documents/Custom Office Templates/Company_Data.csv")
```

In [38]:

```
company.head()
```

Out[38]:

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

In [39]:

```
company.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Sales        400 non-null    float64
 1   CompPrice    400 non-null    int64  
 2   Income       400 non-null    int64  
 3   Advertising  400 non-null    int64  
 4   Population   400 non-null    int64  
 5   Price        400 non-null    int64  
 6   ShelveLoc    400 non-null    object 
 7   Age          400 non-null    int64  
 8   Education    400 non-null    int64  
 9   Urban        400 non-null    object 
 10  US           400 non-null    object 
dtypes: float64(1), int64(7), object(3)
memory usage: 34.5+ KB
```

In [40]:

```
company.describe()
```

Out[40]:

	Sales	CompPrice	Income	Advertising	Population	Price	Age	E
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	40
mean	7.496325	124.975000	68.657500	6.635000	264.840000	115.795000	53.322500	1
std	2.824115	15.334512	27.986037	6.650364	147.376436	23.676664	16.200297	
min	0.000000	77.000000	21.000000	0.000000	10.000000	24.000000	25.000000	1
25%	5.390000	115.000000	42.750000	0.000000	139.000000	100.000000	39.750000	1
50%	7.490000	125.000000	69.000000	5.000000	272.000000	117.000000	54.500000	1
75%	9.320000	135.000000	91.000000	12.000000	398.500000	131.000000	66.000000	1
max	16.270000	175.000000	120.000000	29.000000	509.000000	191.000000	80.000000	1

In [41]:

```
# Duplicates
company[company.duplicated()].shape
```

Out[41]:

```
(0, 11)
```

In [42]:

```
company.isnull().sum()
```

Out[42]:

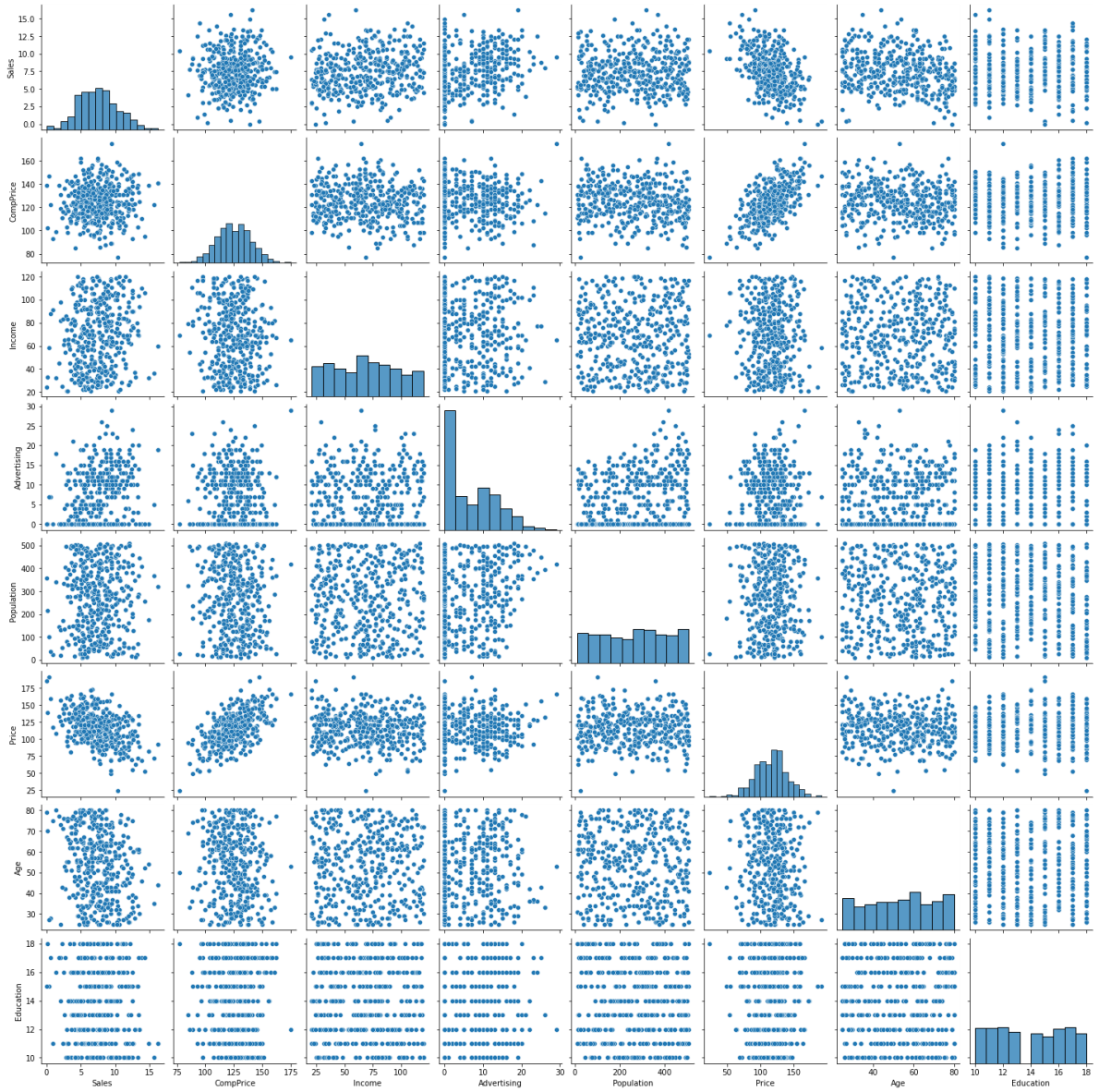
```
Sales          0
CompPrice      0
Income         0
Advertising    0
Population     0
Price          0
ShelveLoc      0
Age            0
Education      0
Urban          0
US             0
dtype: int64
```

In [43]:

```
# Scatter plot and Correaltion Analysis
sns.pairplot(company)
```

Out[43]:

```
<seaborn.axisgrid.PairGrid at 0x4046b38fa0>
```



In [44]:

```
company.corr()
```

Out[44]:

	Sales	CompPrice	Income	Advertising	Population	Price	Age	Ed
Sales	1.000000	0.064079	0.151951	0.269507	0.050471	-0.444951	-0.231815	-0
CompPrice	0.064079	1.000000	-0.080653	-0.024199	-0.094707	0.584848	-0.100239	0
Income	0.151951	-0.080653	1.000000	0.058995	-0.007877	-0.056698	-0.004670	-0
Advertising	0.269507	-0.024199	0.058995	1.000000	0.265652	0.044537	-0.004557	-0
Population	0.050471	-0.094707	-0.007877	0.265652	1.000000	-0.012144	-0.042663	-0
Price	-0.444951	0.584848	-0.056698	0.044537	-0.012144	1.000000	-0.102177	0
Age	-0.231815	-0.100239	-0.004670	-0.004557	-0.042663	-0.102177	1.000000	0
Education	-0.051955	0.025197	-0.056855	-0.033594	-0.106378	0.011747	0.006488	1

In [45]:

```
company_df= company.copy()
```

In [46]:

```
# Converting the sales data in to Categorical
company_df.loc[company['Sales']>=8, 'Sale']='1'  # High=1 Taken based on the mean value
company_df.loc[company['Sales']<=8, 'Sale']='0'  # Low=0
```

In [47]:

```
company_df.head()
```

Out[47]:

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

In [48]:

```
company_df= company_df.drop('Age', axis=1)
company_df= company_df.drop('Education', axis=1)
company_df= company_df.drop('Urban', axis=1)
company_df= company_df.drop('US', axis=1)
company_df=company_df.drop('Sales', axis=1)
```

In [49]:

```
company_df.head()
```

Out[49]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Sale
0	138	73	11	276	120	Bad	1
1	111	48	16	260	83	Good	1
2	113	35	10	269	80	Medium	1
3	117	100	4	466	97	Medium	0
4	141	64	3	340	128	Bad	0

In [50]:

```
label_encoder= preprocessing.LabelEncoder()
company_df['ShelveLoc']=label_encoder.fit_transform(company_df['ShelveLoc'])
```

In [51]:

```
company_df.head()
```

Out[51]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Sale
0	138	73	11	276	120	0	1
1	111	48	16	260	83	1	1
2	113	35	10	269	80	2	1
3	117	100	4	466	97	2	0
4	141	64	3	340	128	0	0

Note- ShelveLoc= Bad=0, Good=1, Medium=2 and Sales= High=1, Low=0

In [52]:

```
company_df=company_df.astype({'Sale':'int'})
```

Normalization Function

In [53]:

```
def norm_func(i):
    x=(i-i.min())/(i.max()-i.min())
    return(x)
```

In [54]:

```
company_df=norm_func(company_df)
```

In [55]:

```
company_df.head()
```

Out[55]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Sale
0	0.622449	0.525253	0.379310	0.533066	0.574850	0.0	1.0
1	0.346939	0.272727	0.551724	0.501002	0.353293	0.5	1.0
2	0.367347	0.141414	0.344828	0.519038	0.335329	1.0	1.0
3	0.408163	0.797980	0.137931	0.913828	0.437126	1.0	0.0
4	0.653061	0.434343	0.103448	0.661323	0.622754	0.0	0.0

In [56]:

```
company_df1=company_df.astype({"ShelveLoc":'int',"Sale":'int'})
```

In [57]:

```
company_df.head()
```

Out[57]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Sale
0	0.622449	0.525253	0.379310	0.533066	0.574850	0.0	1.0
1	0.346939	0.272727	0.551724	0.501002	0.353293	0.5	1.0
2	0.367347	0.141414	0.344828	0.519038	0.335329	1.0	1.0
3	0.408163	0.797980	0.137931	0.913828	0.437126	1.0	0.0
4	0.653061	0.434343	0.103448	0.661323	0.622754	0.0	0.0

In [58]:

```
x= company_df1.iloc[:,0:6]
y= company_df1.iloc[:,6]
```

In [59]:

```
x
```

Out[59]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc
0	0.622449	0.525253	0.379310	0.533066	0.574850	0
1	0.346939	0.272727	0.551724	0.501002	0.353293	0
2	0.367347	0.141414	0.344828	0.519038	0.335329	1
3	0.408163	0.797980	0.137931	0.913828	0.437126	1
4	0.653061	0.434343	0.103448	0.661323	0.622754	0
...	...	...	...	...	...	...
395	0.622449	0.878788	0.586207	0.386774	0.622754	0
396	0.632653	0.020202	0.103448	0.054108	0.574850	1
397	0.867347	0.050505	0.413793	0.717435	0.808383	1
398	0.234694	0.585859	0.241379	0.549098	0.425150	0
399	0.581633	0.161616	0.000000	0.034068	0.574850	0

400 rows × 6 columns

In [60]:

```
y   # Sale - High=1, Low=0
```

Out[60]:

```
0      1
1      1
2      1
3      0
4      0
      ..
395    1
396    0
397    0
398    0
399    1
Name: Sale, Length: 400, dtype: int32
```

In [61]:

```
company_df1['Sale'].unique()
```

Out[61]:

```
array([1, 0])
```

In [62]:

```
company_df1.Sale.value_counts()
```

Out[62]:

```
0    236
1    164
Name: Sale, dtype: int64
```

Splitting data into Training and Testing Dataset

In [63]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=40)
```

Building Decision Tree Classifier using Entropy Criteria

In [64]:

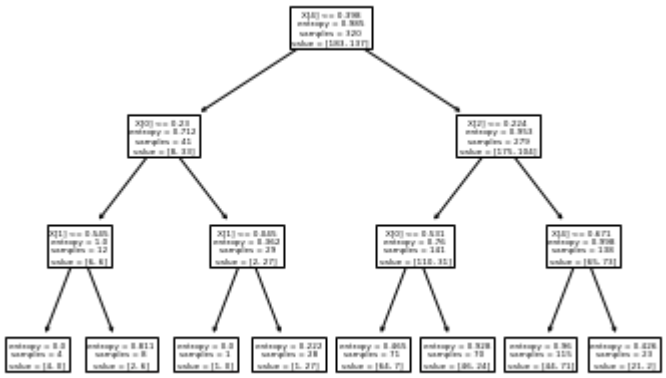
```
model= DecisionTreeClassifier(criterion='entropy', max_depth=3)
model.fit(x_train,y_train)
```

Out[64]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

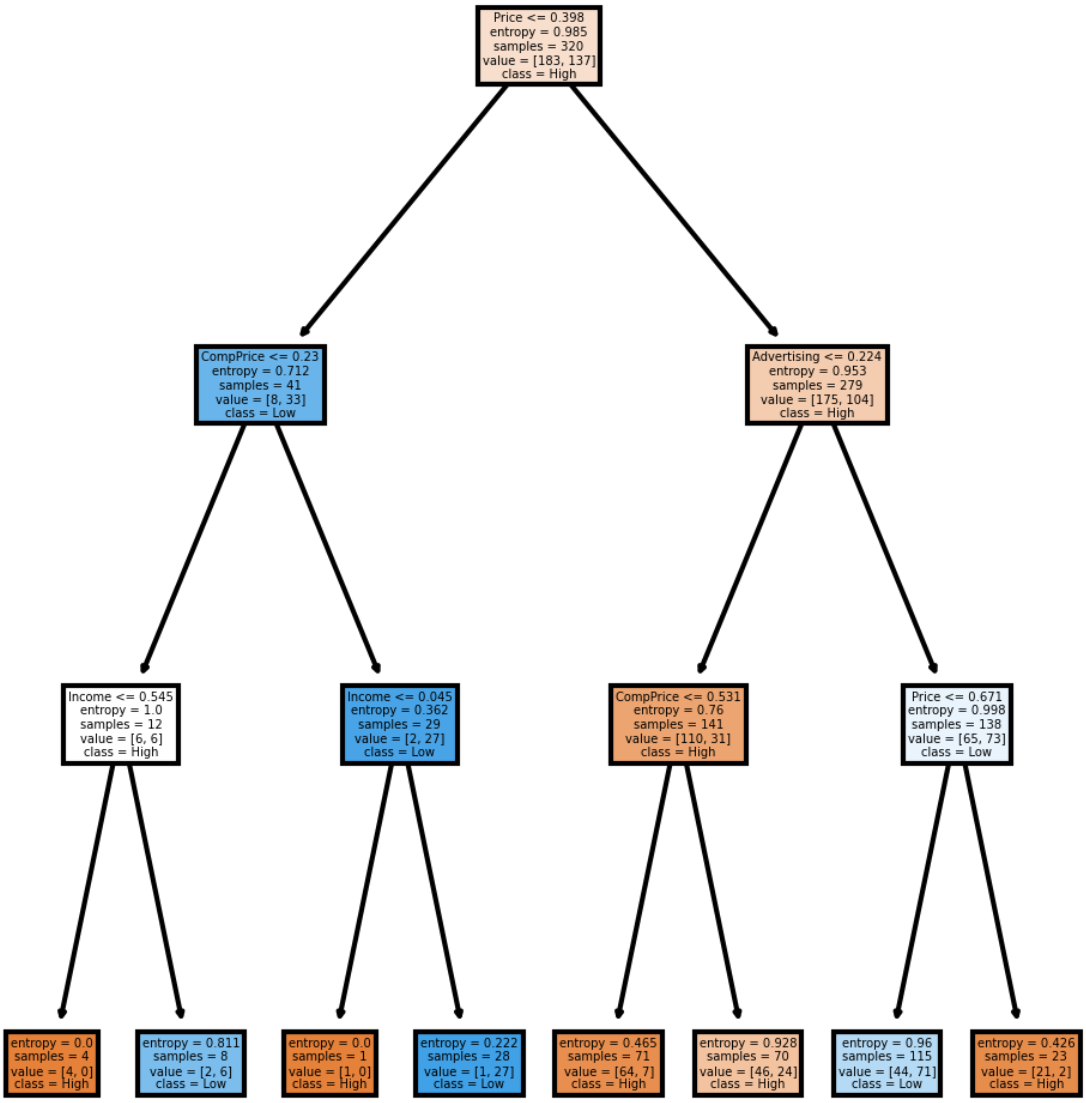
In [65]:

```
# Plot the Decision tree
tree.plot_tree(model);
```



In [66]:

```
fn=['CompPrice','Income','Advertising','Population','Price','ShelveLoc','Sale']
cn=['High','Low']
fig, axes=plt.subplots(nrows=1,ncols=1,figsize=(4,5),dpi=300)
tree.plot_tree(model,feature_names=fn,
               class_names=cn,filled=True);
```



In [67]:

```
# Predict on tesst data
preds= model.predict(x_test)
pd.Series(preds).value_counts()
```

Out[67]:

```
0    51
1    29
dtype: int64
```

In [68]:

```
pred
```

Out[68]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Confusion Matrix

In [69]:

```
pd.crosstab(y_test, preds)
```

Out[69]:

col_0	0	1
Sale		
0	41	12
1	10	17

Accuracy

In [70]:

```
np.mean(preds==y_test)
```

Out[70]:

```
0.725
```

In [ ]:

```

```