Importing necessary Liabraries

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
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,accuracy_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectFromModel,RFE
```

In [2]:

```
company_data=pd.read_csv("Company_Data.csv")
company_data.head(50)
```

Out[2]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Url
0	9.50	138	73	11	276	120	Bad	42	17	
1	11.22	111	48	16	260	83	Good	65	10	
2	10.06	113	35	10	269	80	Medium	59	12	
3	7.40	117	100	4	466	97	Medium	55	14	
4	4.15	141	64	3	340	128	Bad	38	13	
5	10.81	124	113	13	501	72	Bad	78	16	
6	6.63	115	105	0	45	108	Medium	71	15	
7	11.85	136	81	15	425	120	Good	67	10	
8	6.54	132	110	0	108	124	Medium	76	10	
9	4.69	132	113	0	131	124	Medium	76	17	
10	9.01	121	78	9	150	100	Bad	26	10	
11	11.96	117	94	4	503	94	Good	50	13	
12	3.98	122	35	2	393	136	Medium	62	18	
13	10.96	115	28	11	29	86	Good	53	18	
14	11.17	107	117	11	148	118	Good	52	18	
15	8.71	149	95	5	400	144	Medium	76	18	
16	7.58	118	32	0	284	110	Good	63	13	
17	12.29	147	74	13	251	131	Good	52	10	
18	13.91	110	110	0	408	68	Good	46	17	
19	8.73	129	76	16	58	121	Medium	69	12	
20	6.41	125	90	2	367	131	Medium	35	18	
21	12.13	134	29	12	239	109	Good	62	18	
22	5.08	128	46	6	497	138	Medium	42	13	
23	5.87	121	31	0	292	109	Medium	79	10	
24	10.14	145	119	16	294	113	Bad	42	12	
25	14.90	139	32	0	176	82	Good	54	11	
26	8.33	107	115	11	496	131	Good	50	11	
27	5.27	98	118	0	19	107	Medium	64	17	
28	2.99	103	74	0	359	97	Bad	55	11	
29	7.81	104	99	15	226	102	Bad	58	17	
30	13.55	125	94	0	447	89	Good	30	12	
31	8.25	136	58	16	241	131	Medium	44	18	
32	6.20	107	32	12	236	137	Good	64	10	
33	8.77	114	38	13	317	128	Good	50	16	

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Url
34	2.67	115	54	0	406	128	Medium	42	17	
35	11.07	131	84	11	29	96	Medium	44	17	
36	8.89	122	76	0	270	100	Good	60	18	
37	4.95	121	41	5	412	110	Medium	54	10	
38	6.59	109	73	0	454	102	Medium	65	15	
39	3.24	130	60	0	144	138	Bad	38	10	
40	2.07	119	98	0	18	126	Bad	73	17	
41	7.96	157	53	0	403	124	Bad	58	16	
42	10.43	77	69	0	25	24	Medium	50	18	
43	4.12	123	42	11	16	134	Medium	59	13	
44	4.16	85	79	6	325	95	Medium	69	13	
45	4.56	141	63	0	168	135	Bad	44	12	
46	12.44	127	90	14	16	70	Medium	48	15	
47	4.38	126	98	0	173	108	Bad	55	16	
48	3.91	116	52	0	349	98	Bad	69	18	
49	10.61	157	93	0	51	149	Good	32	17	

Initial analysis

In [3]:

```
company_data.isna().sum()
```

Out[3]:

```
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 [4]:

```
company_data.dtypes
```

Out[4]:

```
Sales          float64
CompPrice        int64
Income           int64
Advertising      int64
Population       int64
Price            int64
ShelveLoc       object
Age              int64
Education        int64
Urban           object
US              object
dtype: object
```

In [5]:

```
company_data.shape
```

Out[5]:

```
(400, 11)
```

In [6]:

```
company_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 11 columns):
Sales          400 non-null float64
CompPrice      400 non-null int64
Income         400 non-null int64
Advertising    400 non-null int64
Population     400 non-null int64
Price          400 non-null int64
ShelveLoc      400 non-null object
Age            400 non-null int64
Education      400 non-null int64
Urban          400 non-null object
US             400 non-null object
dtypes: float64(1), int64(7), object(3)
memory usage: 34.5+ KB
```

In [7]:

```
# As per the problem statement, converted into Good and Risky
company_data.loc[company_data['Sales'] <= 9.50, '<= 9.50'] = 'low'
company_data.loc[company_data['Sales'] >9.50 , '> 9.50'] = 'High'
```

In [8]:

```
#Converting all data into numerical data
company_data[['Urban','US']]=pd.get_dummies(company_data[['Urban','US']],drop_first=True)
```

In [9]:

```
company_data['ShelveLoc']=pd.get_dummies(company_data['ShelveLoc'],drop_first=True)
```

In [10]:

```
company_data
```

Out[10]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	U
0	9.50	138	73	11	276	120	0	42	17	
1	11.22	111	48	16	260	83	1	65	10	
2	10.06	113	35	10	269	80	0	59	12	
3	7.40	117	100	4	466	97	0	55	14	
4	4.15	141	64	3	340	128	0	38	13	
...	...	...	...	...	...	...	...	...	...	
395	12.57	138	108	17	203	128	1	33	14	
396	6.14	139	23	3	37	120	0	55	11	
397	7.41	162	26	12	368	159	0	40	18	
398	5.94	100	79	7	284	95	0	50	12	
399	9.71	134	37	0	27	120	1	49	16	

400 rows × 13 columns

In [11]:

```
#deleting the sales column after getting high or low category
del company_data['Sales']
```

In [12]:

```
del company_data['<= 9.50']
```

In [13]:

```
company_data.head(10)
```

Out[13]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	138	73	11	276	120	0	42	17	1	1
1	111	48	16	260	83	1	65	10	1	1
2	113	35	10	269	80	0	59	12	1	1
3	117	100	4	466	97	0	55	14	1	1
4	141	64	3	340	128	0	38	13	1	0
5	124	113	13	501	72	0	78	16	0	1
6	115	105	0	45	108	0	71	15	1	0
7	136	81	15	425	120	1	67	10	1	1
8	132	110	0	108	124	0	76	10	0	0
9	132	113	0	131	124	0	76	17	0	1

In [14]:

```
company_data.fillna(value=0,axis=0,inplace=True)
```

In [15]:

```
company_data.head(10)
```

Out[15]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	138	73	11	276	120	0	42	17	1	1
1	111	48	16	260	83	1	65	10	1	1
2	113	35	10	269	80	0	59	12	1	1
3	117	100	4	466	97	0	55	14	1	1
4	141	64	3	340	128	0	38	13	1	0
5	124	113	13	501	72	0	78	16	0	1
6	115	105	0	45	108	0	71	15	1	0
7	136	81	15	425	120	1	67	10	1	1
8	132	110	0	108	124	0	76	10	0	0
9	132	113	0	131	124	0	76	17	0	1

In [16]:

```
company_data['Sales']=pd.get_dummies(company_data['> 9.50'],drop_first=True)
```

In [17]:

```
company_data.head(10)
```

Out[17]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	138	73	11	276	120	0	42	17	1	1
1	111	48	16	260	83	1	65	10	1	1
2	113	35	10	269	80	0	59	12	1	1
3	117	100	4	466	97	0	55	14	1	1
4	141	64	3	340	128	0	38	13	1	0
5	124	113	13	501	72	0	78	16	0	1
6	115	105	0	45	108	0	71	15	1	0
7	136	81	15	425	120	1	67	10	1	1
8	132	110	0	108	124	0	76	10	0	0
9	132	113	0	131	124	0	76	17	0	1

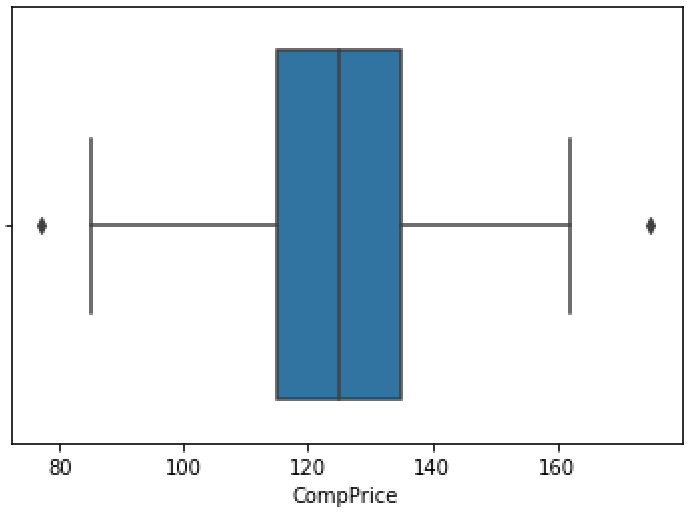
Data Visualization

In [18]:

```
sns.boxplot(x='CompPrice', data=company_data)
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19cf6707088>
```

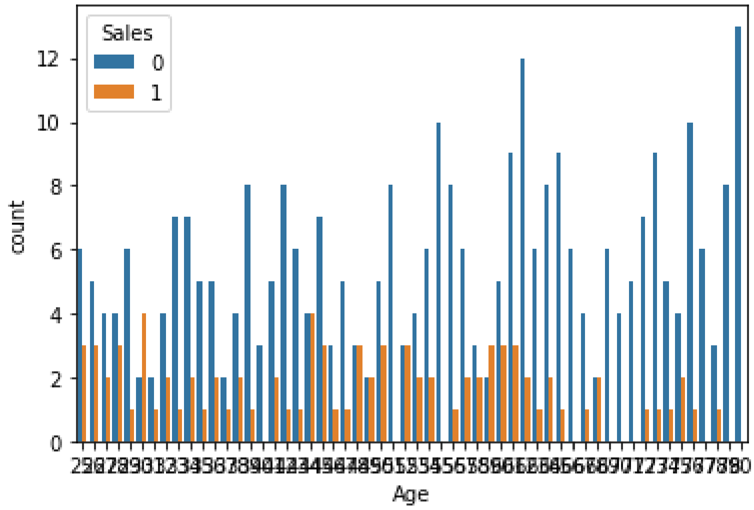


In [19]:

```
sns.countplot(x='Age',hue='Sales',data=company_data)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19c8735d448>
```



In [20]:

```
company_data.dtypes
```

Out[20]:

```
CompPrice       int64
Income          int64
Advertising     int64
Population      int64
Price           int64
ShelveLoc       uint8
Age             int64
Education       int64
Urban           uint8
US              uint8
> 9.50         object
Sales           uint8
dtype: object
```

Data preprocessing

In [21]:

```
#Now we scaled the data by using Standard Scaler
scaled_X=StandardScaler()
x_scale=scaled_X.fit_transform(company_data.iloc[:, :-2])
```

Model Building

In [22]:

```
X=company_data.iloc[:, :-2]
y=company_data[['Sales']]
```

In [23]:

```
X
```

Out[23]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	U
0	138	73	11	276	120	0	42	17	1	
1	111	48	16	260	83	1	65	10	1	
2	113	35	10	269	80	0	59	12	1	
3	117	100	4	466	97	0	55	14	1	
4	141	64	3	340	128	0	38	13	1	
...	...	...	...	...	...	...	...	...	...	
395	138	108	17	203	128	1	33	14	1	
396	139	23	3	37	120	0	55	11	0	
397	162	26	12	368	159	0	40	18	1	
398	100	79	7	284	95	0	50	12	1	
399	134	37	0	27	120	1	49	16	1	

400 rows × 10 columns

In [24]:

```
y
```

Out[24]:

	Sales
0	0
1	1
2	1
3	0
4	0
...	...
395	1
396	0
397	0
398	0
399	1

400 rows × 1 columns

In [26]:

```
X_train,X_test,y_train,y_test=train_test_split(x_scale,y,test_size=0.20,random_state=19)
```

In [27]:

```
X_train
```

Out[27]:

```
array([[-1.23895458, -0.1666307 ,  0.506621  , ...,  1.18444912,
         0.64686916,  0.74188112],
       [ 0.98104309, -0.52439924,  0.35606498, ...,  0.80236876,
        -1.54590766,  0.74188112],
       [ 0.39339665,  0.15536099,  0.95828906, ...,  1.18444912,
         0.64686916,  0.74188112],
       ...,
       [-0.71660219, -0.91794464, -0.99893918, ..., -1.4901134 ,
        -1.54590766,  0.74188112],
       [ 0.1975145 , -0.23818441,  0.05495295, ...,  0.03820804,
         0.64686916,  0.74188112],
       [ 1.30751334, -1.38304374, -0.99893918, ...,  1.18444912,
         0.64686916, -1.34792485]])
```

Decision tree, Random Forest & Gradient Boosting regressor Model

In [28]:

```
#Decison Tree Classifier
d_tree=DecisionTreeClassifier(max_depth=5,random_state=12)
d_tree.fit(X_train,y_train)
```

Out[28]:

```
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=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=12, splitter='best')
```

Model Training

In [29]:

```
y_pred_train=d_tree.predict(X_train)
```

In [30]:

```
y_pred_test=d_tree.predict(X_test)
y_pred_test
```

Out[30]:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0], dtype=uint8)
```

Model Testing Accuracy

In [31]:

```
print("Classification_Report:",classification_report(y_test,y_pred_test))
print("confusion_matrix     :",confusion_matrix(y_test,y_pred_test))
print("Accuracy             :",accuracy_score(y_test,y_pred_test))
```

```
Classification_Report:               precision    recall  f1-score   support

           0       0.85      0.92      0.88        62
           1       0.62      0.44      0.52        18

    accuracy                           0.81        80
   macro avg       0.73      0.68      0.70        80
weighted avg       0.80      0.81      0.80        80

confusion_matrix     : [[57  5]
 [10  8]]
Accuracy             : 0.8125
```

Hyperparameter using GridSearch Cv

In [33]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import GridSearchCV
grid_model = GridSearchCV(estimator = d_tree,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': 'entropy', 'max_depth': 3, 'min_samples_split': 2}
0.8375
```

Fitting the best parameter for better accuracy

In [34]:

```
#Decison Tree Classifier
d_tree=DecisionTreeClassifier(criterion='entropy',max_depth=3,random_state=12)
d_tree.fit(X_train,y_train)
```

Out[34]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
                       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=12, splitter='best')
```

In [35]:

```
y_pred_train1=d_tree.predict(X_train)
```

In [36]:

```
y_pred_test1=d_tree.predict(X_test)
y_pred_test
```

Out[36]:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0], dtype=uint8)
```

In [37]:

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

```
              precision    recall  f1-score   support

           0       0.86      0.92      0.89        62
           1       0.64      0.50      0.56        18

    accuracy                           0.82        80
   macro avg       0.75      0.71      0.73        80
weighted avg       0.81      0.82      0.82        80

[[57  5]
 [ 9  9]]
0.825
```

In [38]:

```
###After fitting the hyperparameter we get better accuracy with reducing false positive and
```

In [39]:

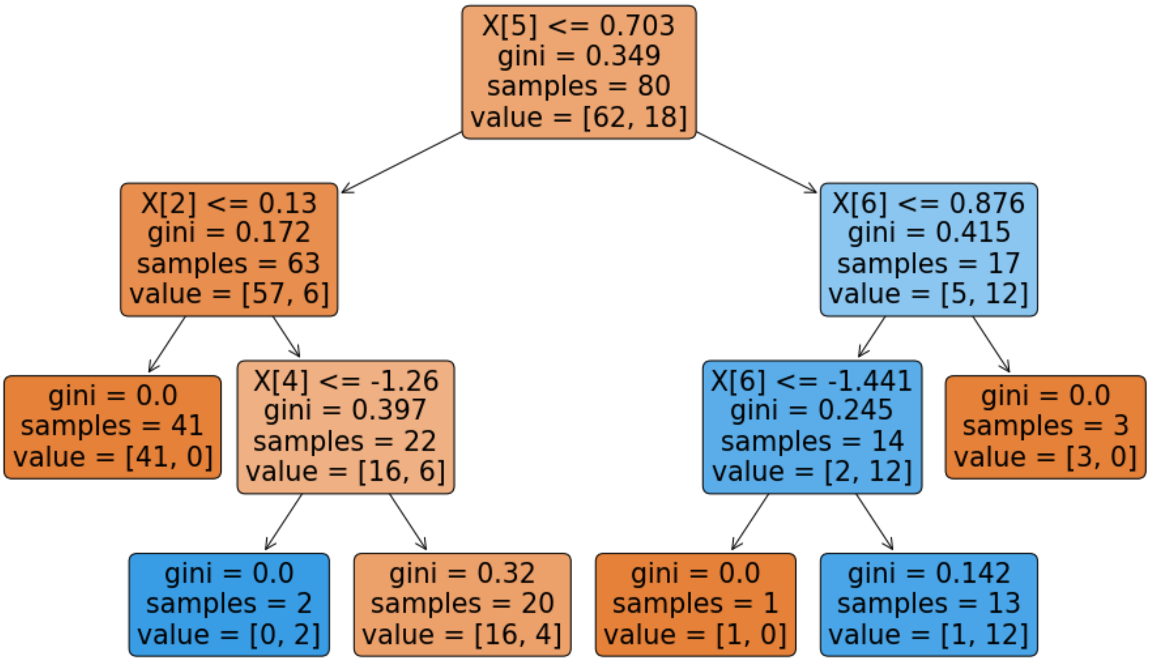
```
roc_auc=roc_auc_score(y_test,y_pred_test1)
roc_auc
```

Out[39]:

```
0.7096774193548386
```

In [40]:

```
model_all_params_max_depth_3 = DecisionTreeClassifier(max_depth = 3,min_samples_split=2,rar
# Prepare a plot figure with set size.
plt.figure(figsize = (16,10))
# Plot the decision tree.
plot_tree(model_all_params_max_depth_3,
          rounded = True,
          filled = True           )
# Display the tree plot figure.
plt.show()
```



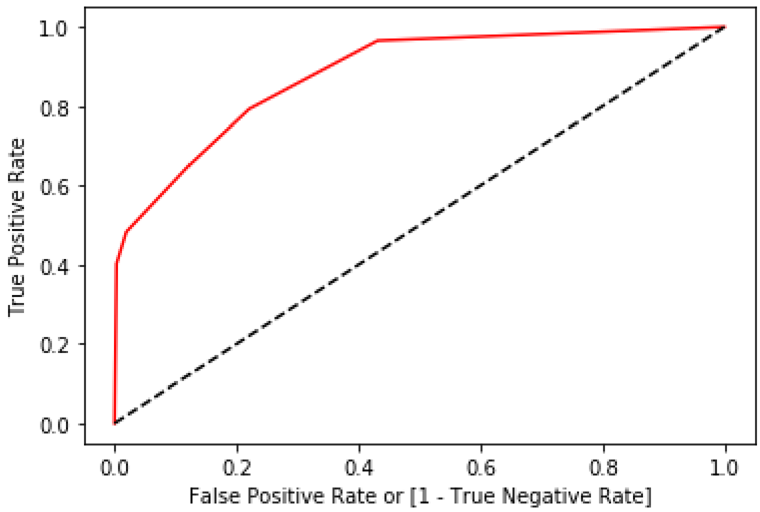
In [42]:

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

fpr, tpr, thresholds = roc_curve(y, d_tree.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='d_tree ( 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')
```

0.7096774193548386

Out[42]:

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



In [43]:

```
## ROC curve gives a level of probability and AUC curve gives a level of seperability
```