Importing the Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Import the dataset

In [2]:

```
dataset=pd.read_csv("C:/Users/navna/Downloads/Data_lab_exam.csv")
```

In [3]:

```
dataset
```

	features	observ_features	price_per_square_foot
0	0.44	0.68	511.14
1	0.99	0.23	717.10
2	0.84	0.29	607.91
3	0.28	0.45	270.40
4	0.07	0.83	289.88
...	...	...	...
95	0.99	0.13	636.22
96	0.28	0.46	272.12
97	0.87	0.36	696.65
98	0.23	0.87	434.53
99	0.77	0.36	593.86

100 rows × 3 columns

In [4]:

```
dataset.shape
```

```
(100, 3)
```

In [5]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 3 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   features              100 non-null    float64
 1   observ_features       100 non-null    float64
 2   price_per_square_foot 100 non-null    float64
dtypes: float64(3)
memory usage: 2.5 KB
```

In [6]:
```
dataset.isna()
```

	features	observ_features	price_per_square_foot
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False
...	...	...	...
95	False	False	False
96	False	False	False
97	False	False	False
98	False	False	False
99	False	False	False

100 rows × 3 columns

In [7]:

```
dataset.isna().sum()
```

```
features                0
observ_features         0
price_per_square_foot   0
dtype: int64
```

In [8]:

```
print(type(dataset))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [9]:

```
dataset.describe()
```

	features	observ_features	price_per_square_foot
count	100.000000	100.000000	100.000000
mean	0.550300	0.501700	554.214600
std	0.293841	0.307124	347.312796
min	0.010000	0.000000	42.080000
25%	0.300000	0.230000	278.172500
50%	0.570000	0.485000	514.285000
75%	0.822500	0.760000	751.752500
max	1.000000	0.990000	1563.820000

In [10]:
```
dataset["observ_features"].value_counts()
```

```
0.23    4
0.62    3
0.69    3
0.40    3
0.64    3
       ..
0.04    1
0.44    1
0.93    1
0.60    1
0.80    1
Name: observ_features, Length: 61, dtype: int64
```

Checking Null Values

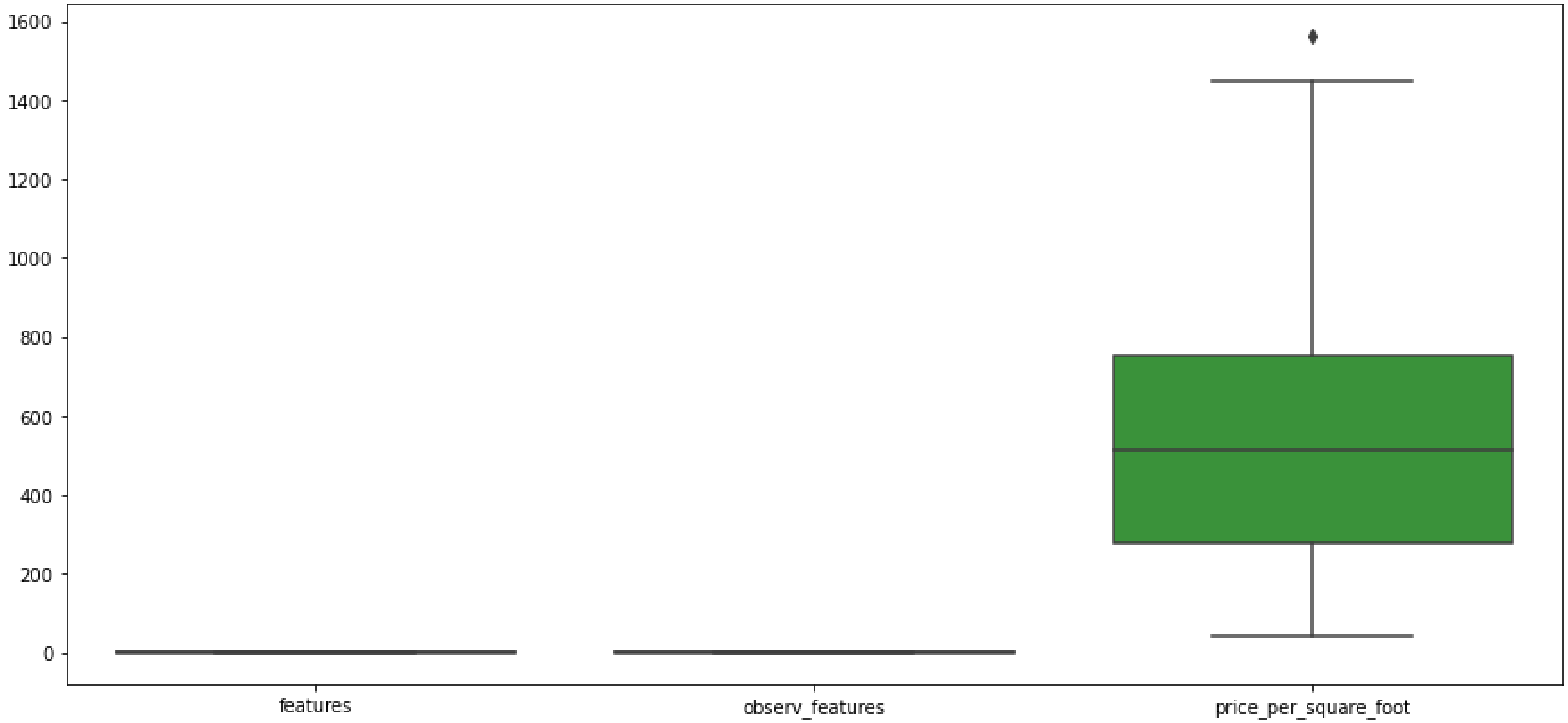
In [11]:
```
dataset.isnull().sum()
```

```
features                0
observ_features         0
price_per_square_foot   0
dtype: int64
```

Checking Outliers

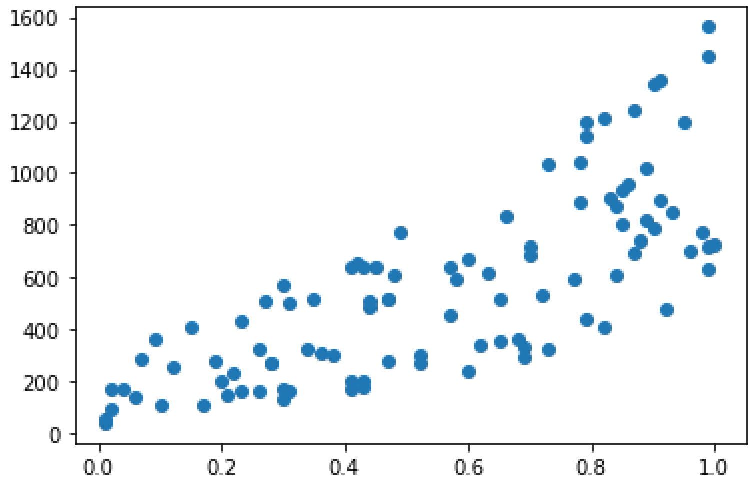
In [12]:

```
plt.figure(figsize=(15,7))
box_plot=sns.boxplot(data=dataset)
#box_plot.set_xticklabels(box_plot.get_xticklabels(),rotation = 60)
```

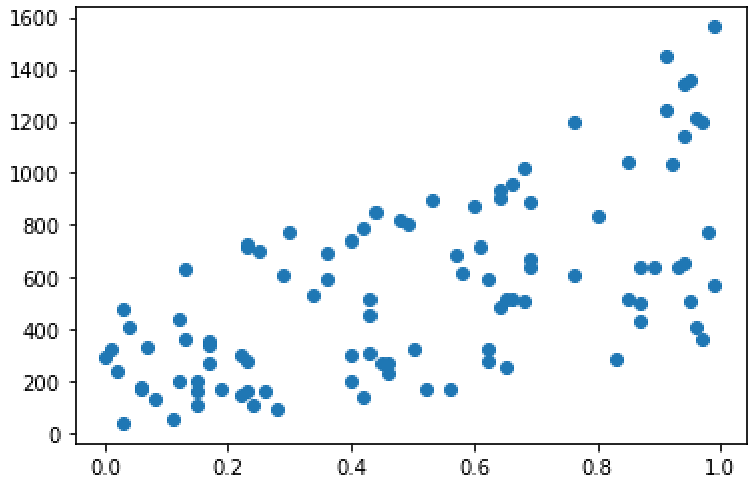


In [13]:

```
plt.scatter(dataset['features'],dataset['price_per_square_foot'])
plt.show()
```

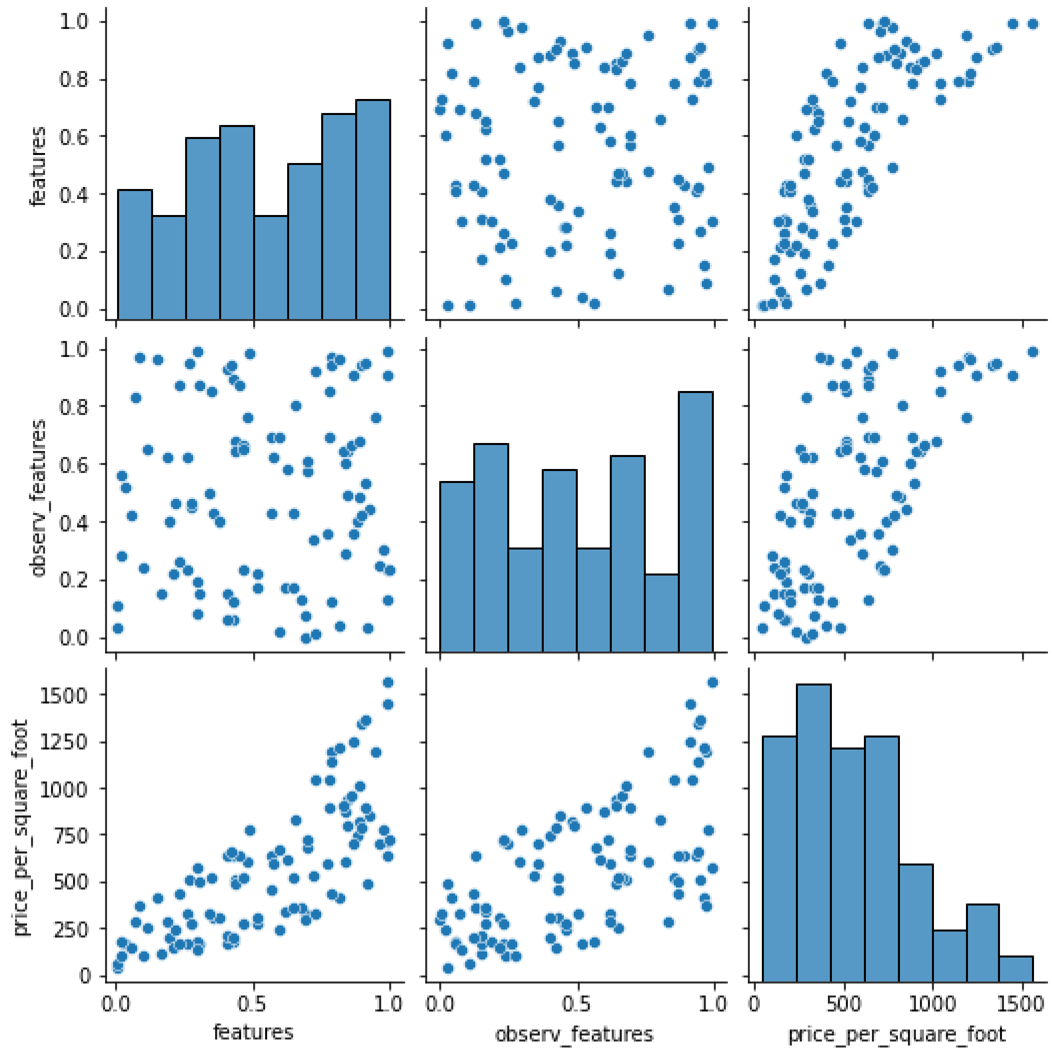


In [14]:
```
plt.scatter(dataset['observ_features'],dataset['price_per_square_foot'])
plt.show()
```



In [15]:
```
sns.pairplot(dataset)
```

```
<seaborn.axisgrid.PairGrid at 0x1197f2f57f0>
```



Conclusion:-

As the number of features and its respective observed_features are incresed then the price is also increases respectively.

Preprocessing

Scaling the data

In [16]:
```
from sklearn.preprocessing import StandardScaler
```

In [17]:
```
scalar=StandardScaler()
scaled=scalar.fit_transform(dataset)
```

In [18]:
```
df=pd.DataFrame(scaled)
```

In [19]:

```
df.head()
```

	0	1	2
0	-0.377265	0.583471	-0.124647
1	1.503928	-0.889115	0.471350
2	0.990875	-0.692770	0.155381
3	-0.924521	-0.169184	-0.821290
4	-1.642794	1.074333	-0.764919

Identifying dependant and independant variables

In [20]:

```
# Independent Variable
X=dataset.iloc[:,0:2].values

# Dependent Variable
y=dataset.iloc[:,-1].values
```

In [21]:
```python
#Check the values of Independent Variable
print(X)
```

```
[[0.44 0.68]
 [0.99 0.23]
 [0.84 0.29]
 [0.28 0.45]
 [0.07 0.83]
 [0.66 0.8 ]
 [0.73 0.92]
 [0.57 0.43]
 [0.43 0.89]
 [0.27 0.95]
 [0.43 0.06]
 [0.87 0.91]
 [0.78 0.69]
 [0.9  0.94]
 [0.41 0.06]
 [0.52 0.17]
 [0.47 0.66]
 [0.65 0.43]
 [0.85 0.64]
 [0.93 0.44]
 [0.41 0.93]
 [0.36 0.43]
 [0.78 0.85]
 [0.69 0.07]
 [0.04 0.52]
 [0.17 0.15]
 [0.68 0.13]
 [0.84 0.6 ]
 [0.38 0.4 ]
 [0.12 0.65]
 [0.62 0.17]
 [0.79 0.97]
 [0.82 0.04]
 [0.91 0.53]
 [0.35 0.85]
 [0.57 0.69]
 [0.52 0.22]
 [0.31 0.15]
```

```
[0.6  0.02]
[0.99 0.91]
[0.48 0.76]
[0.3  0.19]
[0.58 0.62]
[0.65 0.17]
[0.6  0.69]
[0.95 0.76]
[0.47 0.23]
[0.15 0.96]
[0.01 0.03]
[0.26 0.23]
[0.01 0.11]
[0.45 0.87]
[0.09 0.97]
[0.96 0.25]
[0.63 0.58]
[0.06 0.42]
[0.1  0.24]
[0.26 0.62]
[0.41 0.15]
[0.91 0.95]
[0.83 0.64]
[0.44 0.64]
[0.2  0.4 ]
[0.43 0.12]
[0.21 0.22]
[0.88 0.4 ]
[0.31 0.87]
[0.99 0.99]
[0.23 0.26]
[0.79 0.12]
[0.02 0.28]
[0.89 0.48]
[0.02 0.56]
[0.92 0.03]
[0.72 0.34]
[0.3  0.99]
[0.86 0.66]
[0.47 0.65]
[0.79 0.94]
[0.82 0.96]
[0.9  0.42]
```

```
 [0.19 0.62]
 [0.7  0.57]
 [0.7  0.61]
 [0.69 0.  ]
 [0.98 0.3 ]
 [0.3  0.08]
 [0.85 0.49]
 [0.73 0.01]
 [1.   0.23]
 [0.42 0.94]
 [0.49 0.98]
 [0.89 0.68]
 [0.22 0.46]
 [0.34 0.5 ]
 [0.99 0.13]
 [0.28 0.46]
 [0.87 0.36]
 [0.23 0.87]
 [0.77 0.36]]
```

In [22]:

```
#Check the values of dependent Variable
print(y)
```

```
[ 511.14  717.1   607.91  270.4   289.88  830.85 1038.09  455.19  640.17
  511.06  177.03 1242.52  891.37 1339.72  169.88  276.05  517.43  522.25
  932.21  851.25  640.11  308.68 1046.05  332.4   171.85  109.55  361.97
  872.21  303.7   256.38  341.2  1194.63  408.6   895.54  518.25  638.75
  301.9   163.38  240.77 1449.05  609.    174.59  593.45  355.96  671.46
 1193.7   278.88  411.4    42.08  166.19   58.62  642.45  368.14  702.78
  615.74  143.79  109.    328.28  205.16 1360.49  905.83  487.33  202.76
  202.01  148.87  745.3   503.04 1563.82  165.21  438.4    98.47  819.63
  174.44  483.13  534.24  572.31  957.61  518.29 1143.49 1211.31  784.74
  283.7   684.38  719.46  292.23  775.68  130.77  801.6   323.55  726.9
  661.12  771.11 1016.14  237.69  325.89  636.22  272.12  696.65  434.53
  593.86]
```

Splitting the dataset(Training and Testing)

In [23]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25, random_state=28)
```

In [24]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((75, 2), (25, 2), (75,), (25,))
```

In [25]:
```
# Check the values of X_train data
X_train
```

```
array([[0.3 , 0.99],
       [0.47, 0.65],
       [0.42, 0.94],
       [0.27, 0.95],
       [0.31, 0.87],
       [0.19, 0.62],
       [0.41, 0.93],
       [0.89, 0.68],
       [0.01, 0.03],
       [0.86, 0.66],
       [0.06, 0.42],
       [0.26, 0.23],
       [0.87, 0.91],
       [0.93, 0.44],
       [0.34, 0.5 ],
       [0.63, 0.58],
       [0.88, 0.4 ],
       [0.96, 0.25],
       [0.3 , 0.19],
       [0.98, 0.3 ],
       [0.23, 0.87],
       [0.79, 0.97],
       [0.7 , 0.57],
       [0.84, 0.6 ],
       [0.6 , 0.02],
       [0.47, 0.66],
       [0.1 , 0.24],
       [0.87, 0.36],
       [0.89, 0.48],
       [0.15, 0.96],
       [0.17, 0.15],
       [0.84, 0.29],
       [0.91, 0.95],
       [0.9 , 0.42],
       [0.44, 0.64],
       [0.65, 0.43],
       [0.43, 0.06],
       [0.43, 0.12],
```

```
       [0.7 , 0.61],
       [0.41, 0.15],
       [0.26, 0.62],
       [0.73, 0.01],
       [0.28, 0.46],
       [0.6 , 0.69],
       [1.  , 0.23],
       [0.69, 0.  ],
       [0.07, 0.83],
       [0.68, 0.13],
       [0.52, 0.17],
       [0.38, 0.4 ],
       [0.23, 0.26],
       [0.02, 0.28],
       [0.31, 0.15],
       [0.49, 0.98],
       [0.35, 0.85],
       [0.83, 0.64],
       [0.95, 0.76],
       [0.79, 0.94],
       [0.77, 0.36],
       [0.57, 0.43],
       [0.65, 0.17],
       [0.04, 0.52],
       [0.43, 0.89],
       [0.79, 0.12],
       [0.22, 0.46],
       [0.69, 0.07],
       [0.45, 0.87],
       [0.21, 0.22],
       [0.78, 0.69],
       [0.85, 0.49],
       [0.28, 0.45],
       [0.82, 0.04],
       [0.78, 0.85],
       [0.66, 0.8 ],
       [0.99, 0.23]])
```

In [26]:
```
# Check the values of X_train data
y_train
```

```
array([ 572.31,  518.29,  661.12,  511.06,  503.04,  283.7 ,  640.11,
       1016.14,   42.08,  957.61,  143.79,  166.19, 1242.52,  851.25,
        325.89,  615.74,  745.3 ,  702.78,  174.59,  775.68,  434.53,
       1194.63,  684.38,  872.21,  240.77,  517.43,  109.  ,  696.65,
        819.63,  411.4 ,  109.55,  607.91, 1360.49,  784.74,  487.33,
        522.25,  177.03,  202.01,  719.46,  205.16,  328.28,  323.55,
        272.12,  671.46,  726.9 ,  292.23,  289.88,  361.97,  276.05,
        303.7 ,  165.21,   98.47,  163.38,  771.11,  518.25,  905.83,
       1193.7 , 1143.49,  593.86,  455.19,  355.96,  171.85,  640.17,
        438.4 ,  237.69,  332.4 ,  642.45,  148.87,  891.37,  801.6 ,
        270.4 ,  408.6 , 1046.05,  830.85,  717.1 ])
```

In [27]:
```
y_train
```


```
array([ 572.31,  518.29,  661.12,  511.06,  503.04,  283.7 ,  640.11,
       1016.14,   42.08,  957.61,  143.79,  166.19, 1242.52,  851.25,
        325.89,  615.74,  745.3 ,  702.78,  174.59,  775.68,  434.53,
       1194.63,  684.38,  872.21,  240.77,  517.43,  109.  ,  696.65,
        819.63,  411.4 ,  109.55,  607.91, 1360.49,  784.74,  487.33,
        522.25,  177.03,  202.01,  719.46,  205.16,  328.28,  323.55,
        272.12,  671.46,  726.9 ,  292.23,  289.88,  361.97,  276.05,
        303.7 ,  165.21,   98.47,  163.38,  771.11,  518.25,  905.83,
       1193.7 , 1143.49,  593.86,  455.19,  355.96,  171.85,  640.17,
        438.4 ,  237.69,  332.4 ,  642.45,  148.87,  891.37,  801.6 ,
        270.4 ,  408.6 , 1046.05,  830.85,  717.1 ])
```

In [28]:
```
y_test
```

```
array([1563.82,  169.88,  511.14,  130.77, 1339.72,  932.21,  341.2 ,
        301.9 ,  368.14,  636.22, 1449.05,  483.13,  895.54,  609.  ,
        534.24,  638.75,  174.44,  256.38, 1038.09,  308.68,  593.45,
       1211.31,   58.62,  278.88,  202.76])
```

Multiple Linear Regression Model

In [29]:
```
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
reg.fit(X_train,y_train)
```

```
LinearRegression()
```

In [30]:
```

y_pred=reg.predict(X_test)
```

Checking R_2 score of model

In [31]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

```
0.9222052976343534
```

In [32]:
```
import sklearn.metrics as metrics
```

In [33]:
```
metrics.mean_squared_error(y_test,y_pred)
```

```
14133.650472082853
```

As r_2 score is very close to 1 our model is prefectly build using Multiple Linear regression model.

In [34]:
```
reg.predict([[1.503928,0.889115]])
```

```
array([1597.86943476])
```

In [35]:
```
reg.predict([[-1.642794,1.074333]])
```

```
array([-944.28576593])
```

Using Polynomial regression

In [36]:
```
from sklearn.preprocessing import PolynomialFeatures
poly=PolynomialFeatures(degree=4)
lig_ploy=LinearRegression()
lig_ploy.fit(X_train,y_train)
```

```
LinearRegression()
```

In [37]:
```
y_predict=lig_ploy.predict(X_test)
```

In [38]:
```
y_predict
```

```
array([1227.5851301 ,  146.08456401,  565.59687469,   65.81689213,
       1119.72212936,  886.73801098,  393.52668936,  340.78389756,
        454.10622055,  680.85607985, 1176.7266138 ,  558.11237933,
        867.52516976,  650.2671368 ,  586.130401  ,  681.84236308,
        134.28576935,  276.03096469,  963.30758064,  339.04051965,
        645.79409776, 1064.81326684, -160.24632129,  304.87652985,
        184.72159285])
```

In [39]:
```
from sklearn import metrics
r2_score=metrics.r2_score(y_test,y_predict)
r2_score
```

```
0.9222052976343534
```

In [40]:
```
metrics.mean_squared_error(y_test,y_predict)
```

```
14133.650472082853
```

Using Linear regression model hence we can say both are good models to predict house prices

Problem Statement 2

In [41]:
```
data=pd.read_csv("C:/Users/navna/Downloads/Data_cust_problem.csv")
```

In [42]:
```
data
```

	cust_id	age	income	gender	marital_status	buys
0	1	< 21	High	male	single	no
1	2	< 21	High	male	marrid	no
2	3	21 - 35	High	male	single	yes
3	4	> 35	Medium	male	single	yes
4	5	> 35	Low	female	single	yes
5	6	> 35	Low	female	marrid	no
6	7	21 - 35	Low	female	marrid	yes
7	8	< 21	Medium	male	single	no
8	9	< 21	Low	female	marrid	yes
9	10	> 35	Medium	female	single	yes
10	11	< 21	Medium	female	marrid	yes
11	12	21 - 35	Medium	male	marrid	yes
12	13	21 - 35	High	female	single	yes
13	14	> 35	Medium	male	marrid	no

In [43]:
```
data.shape
```

```
(14, 6)
```

In [44]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   cust_id         14 non-null     int64 
 1   age             14 non-null     object
 2   income          14 non-null     object
 3   gender          14 non-null     object
 4   marital_status  14 non-null     object
 5   buys            14 non-null     object
dtypes: int64(1), object(5)
memory usage: 800.0+ bytes
```

In [45]:
```
data.isna()
```

	cust_id	age	income	gender	marital_status	buys
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False
7	False	False	False	False	False	False
8	False	False	False	False	False	False
9	False	False	False	False	False	False
10	False	False	False	False	False	False
11	False	False	False	False	False	False
12	False	False	False	False	False	False
13	False	False	False	False	False	False

In [46]:
```
data.isna().sum()
```

```
cust_id           0
age               0
income            0
gender            0
marital_status    0
buys              0
dtype: int64
```

In [47]:

```
print(type(data))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [48]:

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

```
yes    9
no     5
Name: buys, dtype: int64
```

In [49]:

```
df=data.groupby(["age"],as_index=False).agg(count=("buys",'count'))
df
```

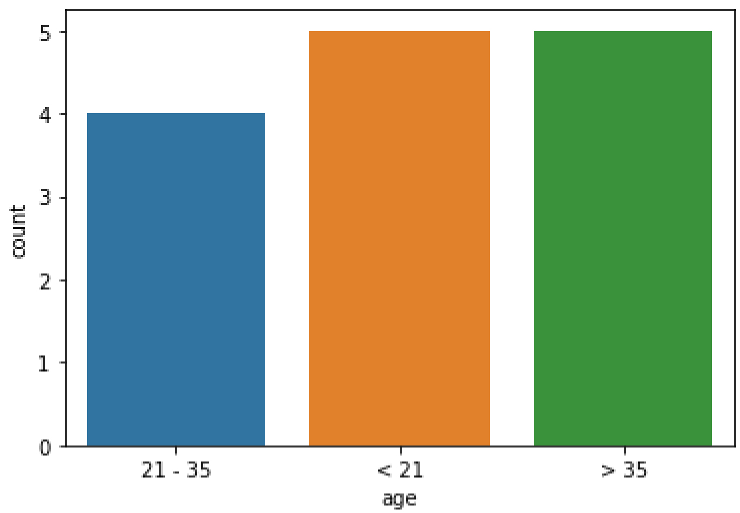
	age	count
0	21 - 35	4
1	< 21	5
2	> 35	5

In [50]:

```
sns.barplot(df["age"],df["count"],data=df)
```

```
C:\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='age', ylabel='count'>
```



from above graph we can conclude that age>35 and age<25 having higher no. of buying lipstick

In [51]:

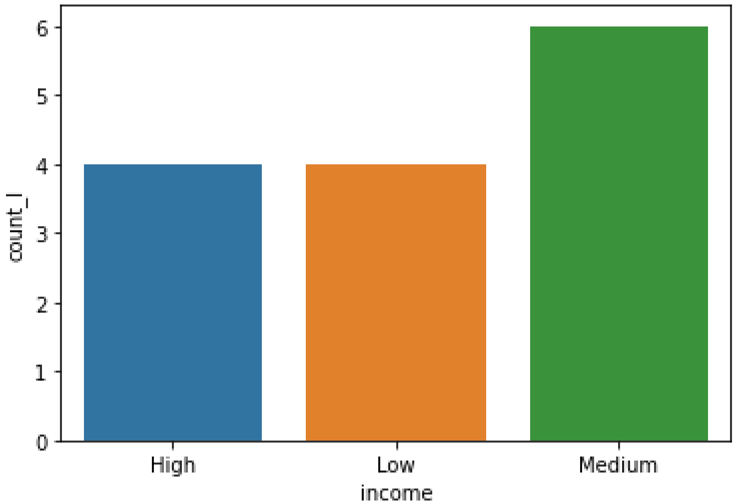
```
df_income=data.groupby(["income"],as_index=False).agg(count_I=("buys",'count'))
df_income
```

	income	count_I
0	High	4
1	Low	4
2	Medium	6

In [52]:
```
sns.barplot(df_income["income"],df_income["count_I"],data=df_income)
```

```
C:\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='income', ylabel='count_I'>
```



from above graph we can conclude that medium income having higher no. of buying lipstick

In [53]:

```
df_gen=data.groupby(["gender"],as_index=False).agg(count_G=("buys",'count'))
df_gen
```

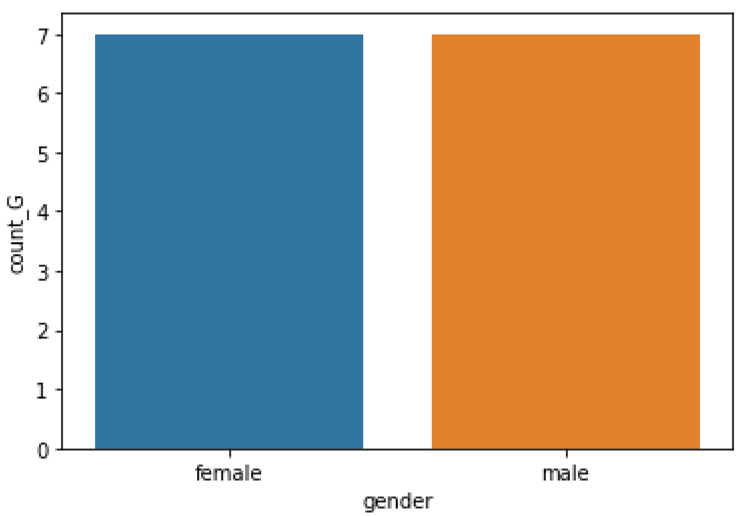
	gender	count_G
0	female	7
1	male	7

In [54]:

```
sns.barplot(df_gen["gender"],df_gen["count_G"],data=df_gen)
```

```
C:\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error o
r misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='gender', ylabel='count_G'>
```



In [55]:

```
df_marital_status=data.groupby(["marital_status"],as_index=False).agg(count_M=("buys",'count'))
df_marital_status
```

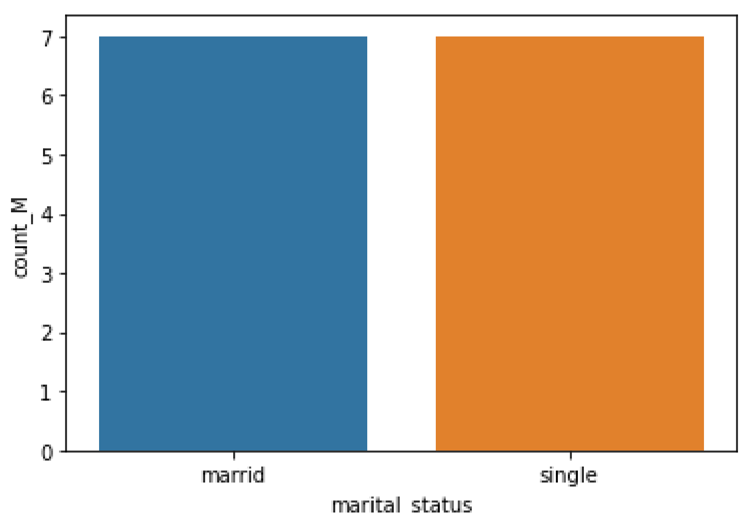
	marital_status	count_M
0	marrid	7
1	single	7

In [56]:

```
sns.barplot(df_marital_status["marital_status"],df_marital_status["count_M"],data=df_marital_status)
```

```
C:\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='marital_status', ylabel='count_M'>
```



Lable Encoding

In [57]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
var_mod = data.select_dtypes(include='object').columns
for i in var_mod:
    data[i] = le.fit_transform(data[i])
```

In [58]:

```
data.head()
```

	cust_id	age	income	gender	marital_status	buys
0	1	1	0	1	1	0
1	2	1	0	1	0	0
2	3	0	0	1	1	1
3	4	2	2	1	1	1
4	5	2	1	0	1	1

age<21 == 1 , age 21-35 == 0 , age>35 == 2

income High == 0 , income medium == 2 , income low == 1

Gender Male == 1 , Gender FeMale == 0

Marital Status Married == 0 , Marital Status Single == 1

Buying Yes == 1 , Buying No == 0

Below we are traning the whole data and according to question's test data we are predicting the output

In [59]:

```
X_t=data.drop(['buys','cust_id'],axis=1)
y_t=data.buys
```

In [60]:

```
from sklearn.tree import DecisionTreeClassifier
dts=DecisionTreeClassifier(criterion='entropy')
dts.fit(X_t,y_t)
```

```
DecisionTreeClassifier(criterion='entropy')
```

test data: [Age < 21, Income = Low, Gender = Female, Marital Status = Married]

In [61]:

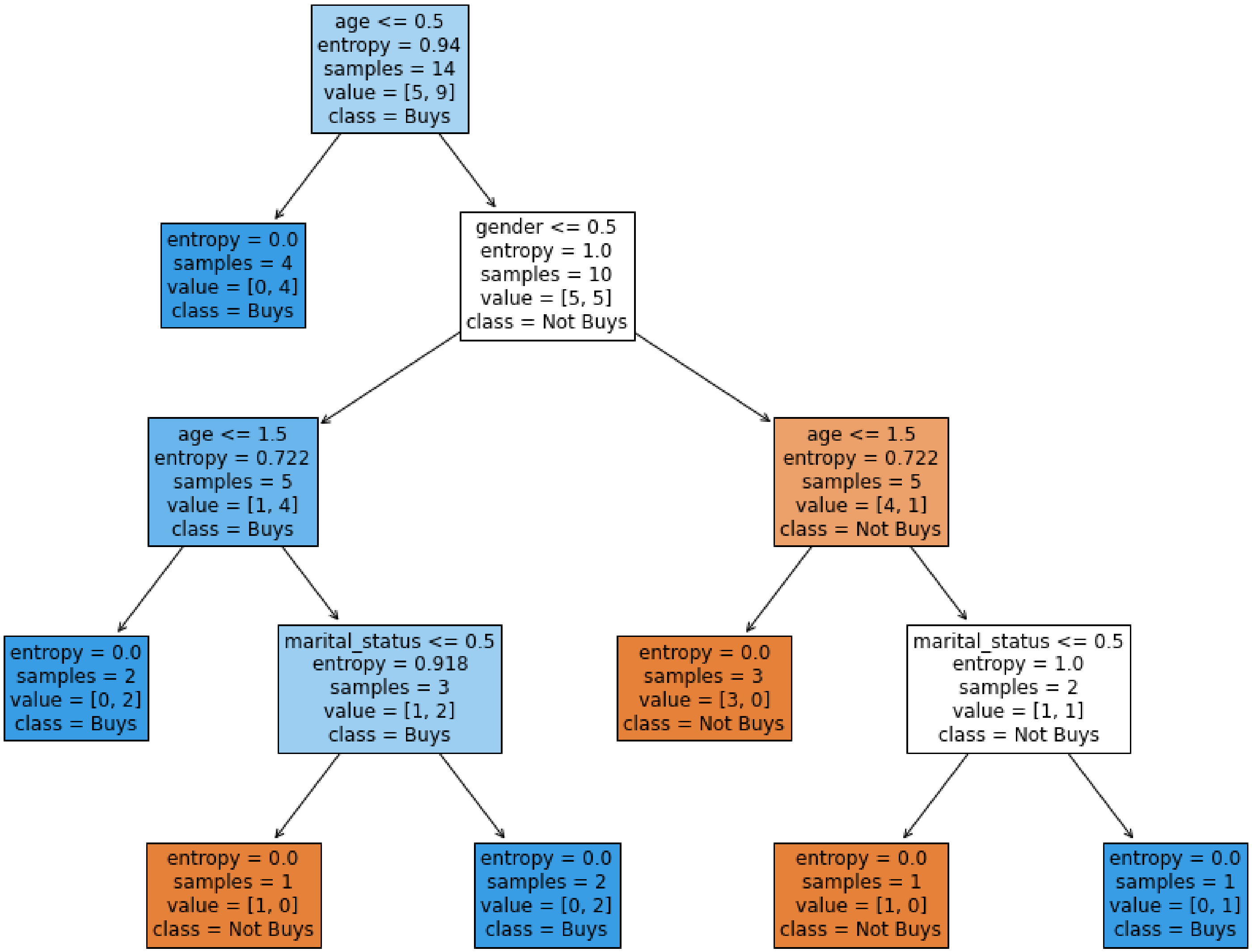
```
dts.predict([[1,1,0,0]])
```

```
C:\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitte
d with feature names
  warnings.warn(
```

```
array([1])
```

In [62]:

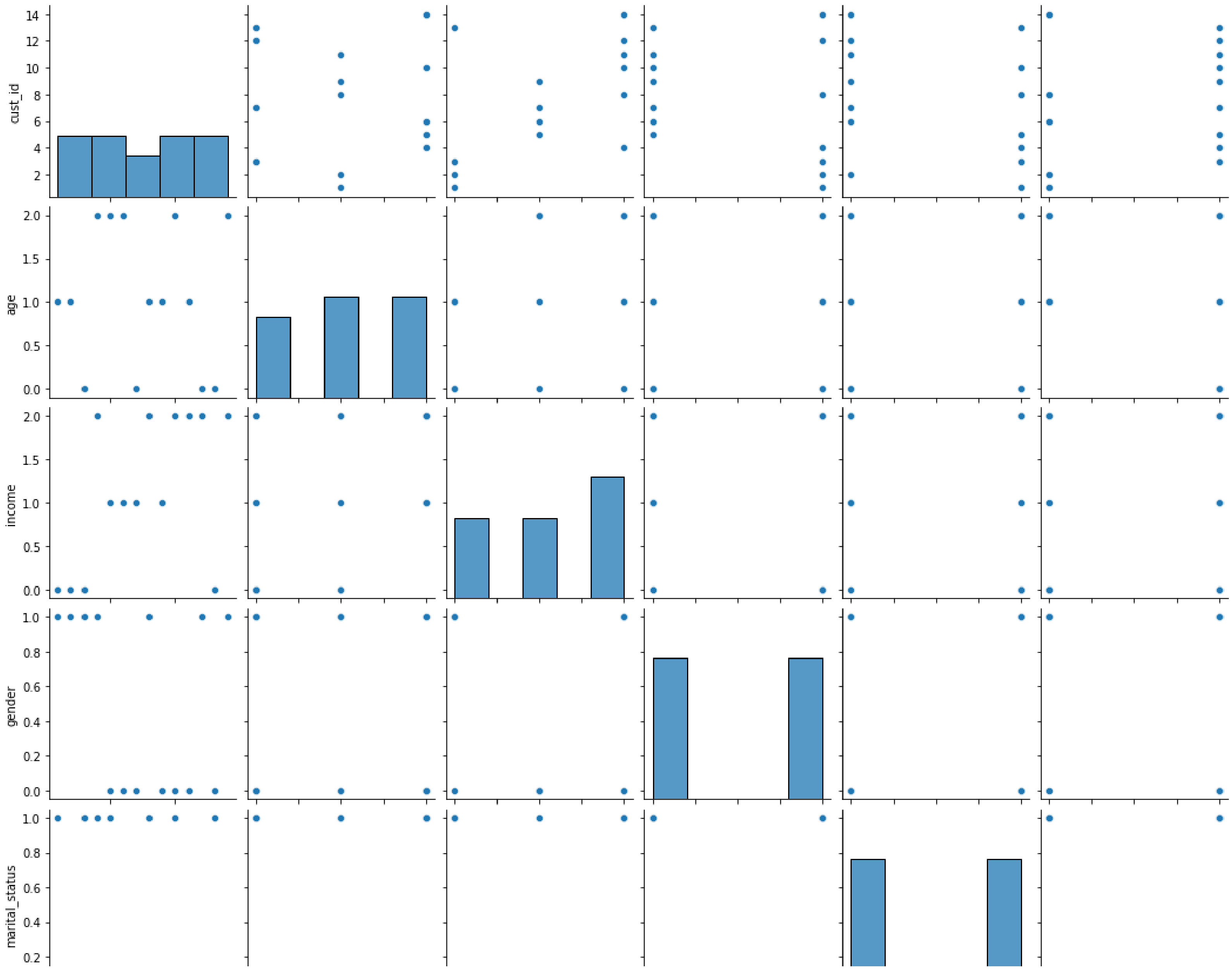
```
from sklearn.tree import plot_tree

fig=plt.figure(figsize=(16,12))
a= plot_tree(dts,feature_names=X_t.columns,fontsize=12,filled=True,class_names=['Not Buys','Buys'])
```

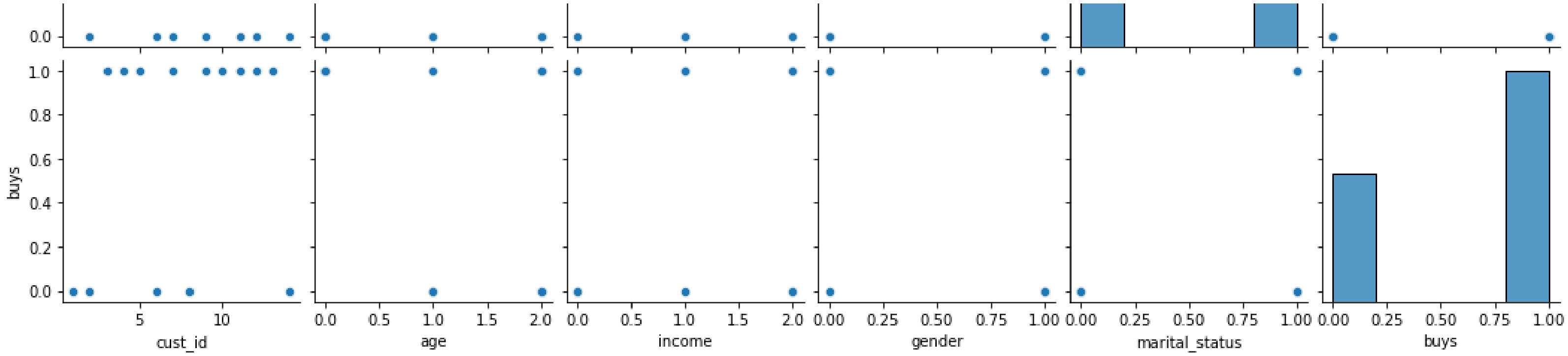


Root Node for Desicion tree is Age

In [63]:
```
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x11902121490>
```





Using Test train split

In [64]:
```
X_t_train,X_t_test,Y_t_train,Y_t_test=train_test_split(X_t,y_t , test_size=0.35 , random_state=7)
```

In [65]:
```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(criterion='entropy')
dt.fit(X_t_train,Y_t_train)
```

```
DecisionTreeClassifier(criterion='entropy')
```

In [66]:
```
y_pred_dt=dt.predict(X_t_test)
```

In [67]:
```
dt.predict([[1,1,0,0]])
```

```
C:\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitte
d with feature names
  warnings.warn(
```

```
array([1])
```

In [68]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_t_test,y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.33      0.50      0.40         2
           1       0.50      0.33      0.40         3

    accuracy                           0.40         5
   macro avg       0.42      0.42      0.40         5
weighted avg       0.43      0.40      0.40         5
```

In [69]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm=confusion_matrix(Y_t_test,y_pred_dt)
cm
```

```
array([[1, 1],
       [2, 1]], dtype=int64)
```

In [70]:

```
### What is the probability of buy
probability_of_buy = data['buys']
probability_of_buy.groupby(probability_of_buy).size()
```

```
buys
0    5
1    9
Name: buys, dtype: int64
```

In [71]:
```
total_cust=data.buys.count()
total_cust
```

```
14
```

In [72]:
```
total_buy=9
total_buy
```

```
9
```

In [73]:
```
percent_buy=(total_buy/total_cust)*100
percent_buy
```

```
64.28571428571429
```

In [ ]: