In [75]:
```
#Importing the Library Files

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

In [76]:
```
HDPP = pd.read_csv(r"C:\Users\Vicky Yewle\Downloads\Machine Learning\Supervised Learnin
HDPP.head()
```

Out[76]:

	age	gender	chest_pain	rest_bps	cholestrol	fasting_blood_sugar	rest_ecg	thalach	exer_angina	o
0	63	1	3	145	233	1	0	150	0	
1	37	1	2	130	250	0	1	187	0	
2	41	0	1	130	204	0	0	172	0	
3	56	1	1	120	236	0	1	178	0	
4	57	0	0	120	354	0	1	163	1	

In [77]:
```
HDPP.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  303 non-null    int64  
 1   gender               303 non-null    int64  
 2   chest_pain           303 non-null    int64  
 3   rest_bps             303 non-null    int64  
 4   cholestrol           303 non-null    int64  
 5   fasting_blood_sugar  303 non-null    int64  
 6   rest_ecg             303 non-null    int64  
 7   thalach              303 non-null    int64  
 8   exer_angina          303 non-null    int64  
 9   old_peak             303 non-null    float64
 10  slope                303 non-null    int64  
 11  ca                   303 non-null    int64  
 12  thalassemia          303 non-null    int64  
 13  target               303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [78]:
```
# Importing train-test-split
from sklearn.model_selection import train_test_split
```

In [79]:
```
# Putting feature variable to X
X = HDPP.drop('target',axis=1)

# Putting response variable to y
y = HDPP['target']
```

In [80]:
```
#splitting of Data
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.20,
                                                    random_state = 100)

X_train.head()
```

Out[80]:

	age	gender	chest_pain	rest_bps	cholestrol	fasting_blood_sugar	rest_ecg	thalach	exer_angina
184	50	1	0	150	243	0	0	128	0
19	69	0	3	140	239	0	1	151	0
118	46	0	1	105	204	0	1	172	0
41	48	1	1	130	245	0	0	180	0
59	57	0	0	128	303	0	0	159	0

In [81]:
```
y_test.shape
```

Out[81]: (61,)

In [82]:
```
# Import Decision Tree Classifier From Decision-Tree
from sklearn.tree import DecisionTreeClassifier

#Fitting the Decision Tree with Default Hyperparameters apart from max depth
# X_train and y_train are fitted into default decision tree
dt_default = DecisionTreeClassifier(max_depth=6)
dt_default.fit(X_train, y_train)
```

Out[82]: DecisionTreeClassifier(max_depth=6)

In [83]:
```
y_train.shape
```

Out[83]: (242,)

In [84]:
```
X_test.head()
```

Out[84]:

	age	gender	chest_pain	rest_bps	cholestrol	fasting_blood_sugar	rest_ecg	thalach	exer_angina
69	62	0	0	124	209	0	1	163	0
300	68	1	0	144	193	1	1	141	0
220	63	0	0	150	407	0	0	154	0
134	41	0	1	126	306	0	1	163	0
7	44	1	1	120	263	0	1	173	0

In [85]:
```
# Let's check the evaluation metrics of our default model

# Importing classification report and confusion matrix from sklearn metrics
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Making predictions
#test an
```

```
y_pred_default = dt_default.predict(X_test)

print(classification_report(y_test, y_pred_default))
```

```
              precision    recall  f1-score   support

           0       0.88      0.70      0.78        33
           1       0.71      0.89      0.79        28

    accuracy                           0.79        61
   macro avg       0.80      0.79      0.79        61
weighted avg       0.81      0.79      0.79        61
```

In [86]:
```
print(confusion_matrix(y_test,y_pred_default))
print(accuracy_score(y_test,y_pred_default))
```

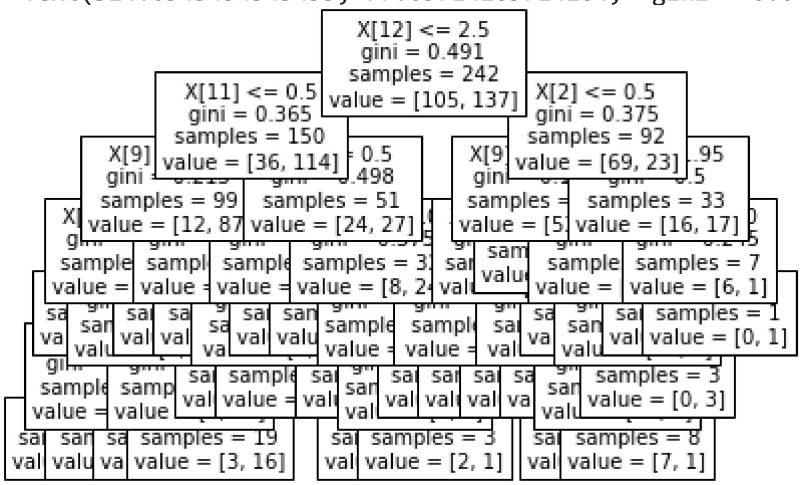
```
[[23 10]
 [ 3 25]]
0.7868852459016393
```

In [87]:
```
# As we have got accuracy score 0.770 whcih is verry less we will do iterative action
```

In [88]:
```
from sklearn import tree
tree.plot_tree(dt_default,fontsize=10)
```

Out[88]:
```
[Text(177.54545454545456, 201.90857142857143, 'X[12] <= 2.5\ngini = 0.491\nsamples = 242
\nvalue = [105, 137]'),
 Text(91.30909090909091, 170.84571428571428, 'X[11] <= 0.5\ngini = 0.365\nsamples = 150
\nvalue = [36, 114]'),
 Text(50.727272727272734, 139.78285714285715, 'X[9] <= 2.7\ngini = 0.213\nsamples = 99\n
value = [12, 87]'),
 Text(30.436363636363637, 108.72, 'X[7] <= 83.5\ngini = 0.172\nsamples = 95\nvalue = [9,
86]'),
 Text(20.290909090909093, 77.65714285714284, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(40.581818181818186, 77.65714285714284, 'X[4] <= 272.0\ngini = 0.156\nsamples = 94
\nvalue = [8, 86]'),
 Text(20.290909090909093, 46.59428571428572, 'X[8] <= 0.5\ngini = 0.079\nsamples = 73\nv
alue = [3, 70]'),
 Text(10.145454545454546, 15.531428571428563, 'gini = 0.032\nsamples = 61\nvalue = [1, 6
0]'),
 Text(30.436363636363637, 15.531428571428563, 'gini = 0.278\nsamples = 12\nvalue = [2, 1
0]'),
 Text(60.872727272727275, 46.59428571428572, 'X[7] <= 132.0\ngini = 0.363\nsamples = 21
\nvalue = [5, 16]'),
 Text(50.727272727272734, 15.531428571428563, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(71.01818181818183, 15.531428571428563, 'gini = 0.266\nsamples = 19\nvalue = [3, 1
6]'),
 Text(71.01818181818183, 108.72, 'X[6] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [3,
1]'),
 Text(60.872727272727275, 77.65714285714284, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(81.16363636363637, 77.65714285714284, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(131.8909090909091, 139.78285714285715, 'X[2] <= 0.5\ngini = 0.498\nsamples = 51\nv
alue = [24, 27]'),
 Text(111.60000000000001, 108.72, 'X[1] <= 0.5\ngini = 0.266\nsamples = 19\nvalue = [16,
3]'),
 Text(101.45454545454547, 77.65714285714284, 'X[7] <= 159.5\ngini = 0.48\nsamples = 5\nv
alue = [2, 3]'),
 Text(91.30909090909091, 46.59428571428572, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(111.60000000000001, 46.59428571428572, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(121.74545454545455, 77.65714285714284, 'gini = 0.0\nsamples = 14\nvalue = [14,
0]'),
 Text(152.1818181818182, 108.72, 'X[0] <= 55.0\ngini = 0.375\nsamples = 32\nvalue = [8,
```

```
24]'),
 Text(142.03636363636366, 77.65714285714284, 'gini = 0.0\nsamples = 15\nvalue = [0, 1
5]'),
 Text(162.32727272727274, 77.65714285714284, 'X[0] <= 59.5\ngini = 0.498\nsamples = 17\n
value = [8, 9]'),
 Text(152.1818181818182, 46.59428571428572, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(172.4727272727273, 46.59428571428572, 'X[1] <= 0.5\ngini = 0.298\nsamples = 11\nva
lue = [2, 9]'),
 Text(162.32727272727274, 15.531428571428563, 'gini = 0.0\nsamples = 8\nvalue = [0,
8]'),
 Text(182.61818181818182, 15.531428571428563, 'gini = 0.444\nsamples = 3\nvalue = [2,
1]'),
 Text(263.7818181818182, 170.84571428571428, 'X[2] <= 0.5\ngini = 0.375\nsamples = 92\nv
alue = [69, 23]'),
 Text(233.34545454545457, 139.78285714285715, 'X[9] <= 0.65\ngini = 0.183\nsamples = 59
\nvalue = [53, 6]'),
 Text(223.20000000000002, 108.72, 'X[11] <= 0.5\ngini = 0.457\nsamples = 17\nvalue = [1
1, 6]'),
 Text(202.90909090909093, 77.65714285714284, 'X[4] <= 237.5\ngini = 0.469\nsamples = 8\n
value = [3, 5]'),
 Text(192.76363636363638, 46.59428571428572, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(213.05454545454546, 46.59428571428572, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(243.4909090909091, 77.65714285714284, 'X[7] <= 124.0\ngini = 0.198\nsamples = 9\nv
alue = [8, 1]'),
 Text(233.34545454545457, 46.59428571428572, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(253.63636363636365, 46.59428571428572, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(243.4909090909091, 108.72, 'gini = 0.0\nsamples = 42\nvalue = [42, 0]'),
 Text(294.21818181818185, 139.78285714285715, 'X[9] <= 1.95\ngini = 0.5\nsamples = 33\nv
alue = [16, 17]'),
 Text(273.92727272727274, 108.72, 'X[4] <= 228.0\ngini = 0.473\nsamples = 26\nvalue = [1
0, 16]'),
 Text(263.7818181818182, 77.65714285714284, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]'),
 Text(284.0727272727273, 77.65714285714284, 'X[4] <= 293.0\ngini = 0.494\nsamples = 18\n
value = [10, 8]'),
 Text(273.92727272727274, 46.59428571428572, 'X[4] <= 259.5\ngini = 0.444\nsamples = 15
\nvalue = [10, 5]'),
 Text(263.7818181818182, 15.531428571428563, 'gini = 0.49\nsamples = 7\nvalue = [3,
4]'),
 Text(284.0727272727273, 15.531428571428563, 'gini = 0.219\nsamples = 8\nvalue = [7,
1]'),
 Text(294.21818181818185, 46.59428571428572, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(314.50909090909096, 108.72, 'X[9] <= 4.0\ngini = 0.245\nsamples = 7\nvalue = [6,
1]'),
 Text(304.3636363636364, 77.65714285714284, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(324.6545454545455, 77.65714285714284, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]')]
```



In [89]:
```
# GridSearchCV to find optimal max_depth
from sklearn.model_selection import KFold
```

```
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 6

# parameters to build the model on
parameters = {'max_depth': range(1,40)}

# instantiate the model
dtree = DecisionTreeClassifier(criterion = "gini",
                               random_state = 100)

# fit tree on training data
tree = GridSearchCV(dtree,
                    parameters,
                    cv=n_folds,
                   scoring="accuracy",
                   return_train_score=True)
tree.fit(X_train, y_train)
```

Out[89]:
```
GridSearchCV(cv=6, estimator=DecisionTreeClassifier(random_state=100),
             param_grid={'max_depth': range(1, 40)}, return_train_score=True,
             scoring='accuracy')
```

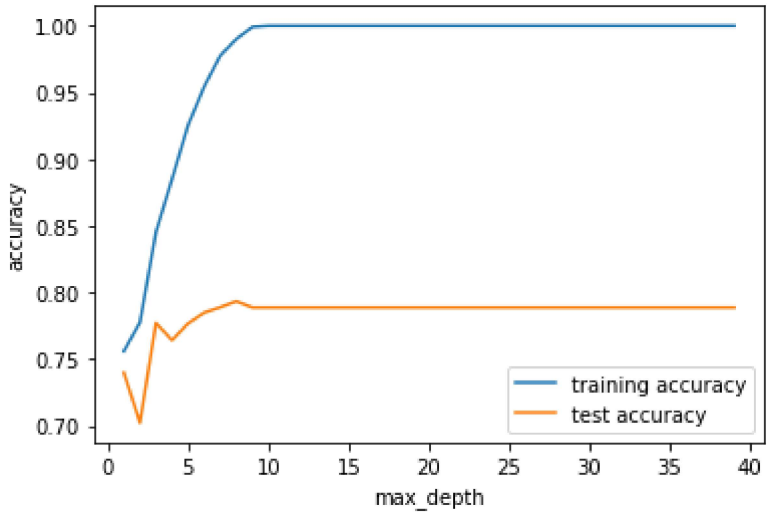
In [90]:
```
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

Out[90]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params	split0_
0	0.005598	0.003998	0.003021	0.004315	1	{'max_depth': 1}	
1	0.010615	0.007518	0.002605	0.005824	2	{'max_depth': 2}	
2	0.007816	0.007816	0.002606	0.005828	3	{'max_depth': 3}	
3	0.006147	0.005671	0.004446	0.005756	4	{'max_depth': 4}	
4	0.000000	0.000000	0.007038	0.007157	5	{'max_depth': 5}	

5 rows × 23 columns

In [91]:
```
plt.figure()
plt.plot(scores["param_max_depth"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_max_depth"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("max_depth")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```



In [92]:
```
# GridSearchCV to find optimal max_depth
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 6

# parameters to build the model on
parameters = {'min_samples_leaf': range(2,50,10)}

# instantiate the model
dtree = DecisionTreeClassifier(criterion = "gini",
                               random_state = 100)

# fit tree on training data
tree = GridSearchCV(dtree,
                    parameters,
                    cv=n_folds,
                   scoring="accuracy",
                   return_train_score=True)
tree.fit(X_train, y_train)
```

Out[92]:
```
GridSearchCV(cv=6, estimator=DecisionTreeClassifier(random_state=100),
             param_grid={'min_samples_leaf': range(2, 50, 10)},
             return_train_score=True, scoring='accuracy')
```

In [93]:
```
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

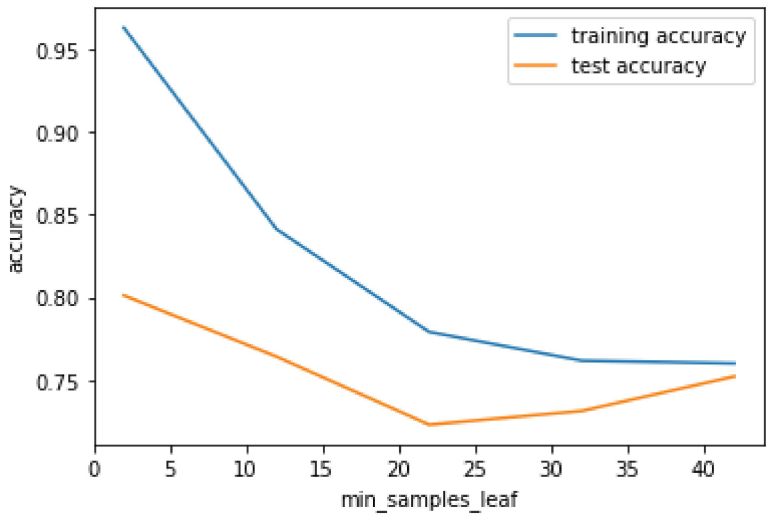
Out[93]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	para
0	0.008401	0.000772	0.000169	0.000378	2	{'min_samples_l
1	0.004002	0.004002	0.005310	0.003757	12	{'min_samples_l
2	0.003232	0.006786	0.000341	0.000762	22	{'min_samples_l
3	0.007818	0.007818	0.002606	0.005827	32	{'min_samples_l

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	para
4	0.006752	0.003022	0.005271	0.005853	42	{'min_samples_l

5 rows × 23 columns

In [94]:
```
plt.figure()
plt.plot(scores["param_min_samples_leaf"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_min_samples_leaf"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("min_samples_leaf")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```



In [95]:
```
# GridSearchCV to find optimal min_samples_split
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 6

# parameters to build the model on
parameters = {'min_samples_split': range(5, 30, 5)}

# instantiate the model
dtree = DecisionTreeClassifier(criterion = "gini",
                               random_state = 100)

# fit tree on training data
tree = GridSearchCV(dtree, parameters,
                    cv=n_folds,
                   scoring="accuracy",
```

```
                    return_train_score=True)
tree.fit(X_train, y_train)
```

Out[95]:
```
GridSearchCV(cv=6, estimator=DecisionTreeClassifier(random_state=100),
             param_grid={'min_samples_split': range(5, 30, 5)},
             return_train_score=True, scoring='accuracy')
```

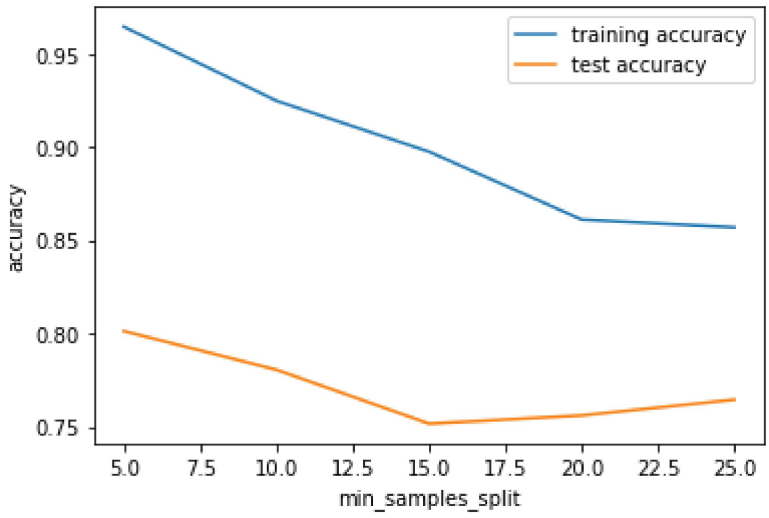
In [96]:
```
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

Out[96]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	pa
0	0.008256	0.000563	0.001333	0.002980	5	{'min_samples_
1	0.005867	0.007045	0.005210	0.007368	10	{'min_samples_
2	0.010421	0.007368	0.002605	0.005825	15	{'min_samples_
3	0.009332	0.002980	0.002669	0.003774	20	{'min_samples_
4	0.008002	0.000003	0.004061	0.004063	25	{'min_samples_

5 rows × 23 columns

In [97]:
```
plt.figure()
plt.plot(scores["param_min_samples_split"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_min_samples_split"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("min_samples_split")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```



In [98]:
```
### create a final model
```

```
param_grid= {
    'max_depth': range(1,8,2),
    'min_samples_leaf': range(2,50,10),
    'min_samples_split': range(5,30,5),
    'criterion' :["entropy","gini"]
}
n_folds = 6

dtree = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator = dtree,
                param_grid= param_grid,
                cv = n_folds,
                verbose=1,
                n_jobs=-1
)
grid_search.fit(X_train, y_train)
```

```
Fitting 6 folds for each of 200 candidates, totalling 1200 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    8.9s
[Parallel(n_jobs=-1)]: Done 976 tasks      | elapsed:   12.1s
[Parallel(n_jobs=-1)]: Done 1200 out of 1200 | elapsed:   12.9s finished
```

Out[98]:
```
GridSearchCV(cv=6, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['entropy', 'gini'],
                         'max_depth': range(1, 8, 2),
                         'min_samples_leaf': range(2, 50, 10),
                         'min_samples_split': range(5, 30, 5)},
             verbose=1)
```

In [99]:
```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

Out[99]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	p
0	0.014439	0.002882	0.012327	0.007830	entropy	1	
1	0.018800	0.005927	0.012252	0.006137	entropy	1	
2	0.016554	0.005204	0.012095	0.004094	entropy	1	
3	0.011097	0.005742	0.010146	0.005695	entropy	1	
4	0.013657	0.002789	0.013026	0.005825	entropy	1	
...	...	...	...	...	...	...	...

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	p
195	0.008003	0.000008	0.008007	0.000004	gini	7	
196	0.017547	0.011195	0.006643	0.003069	gini	7	
197	0.011179	0.004167	0.009830	0.003119	gini	7	
198	0.010561	0.005765	0.010422	0.007369	gini	7	
199	0.010420	0.007368	0.005208	0.007366	gini	7	

200 rows × 18 columns

In [100…
```
print("best accuracy",grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
best accuracy 0.7932926829268293
DecisionTreeClassifier(max_depth=5, min_samples_leaf=2, min_samples_split=5)
```

In [101…
```
clf_gini = DecisionTreeClassifier(criterion = "entropy",
                                  max_depth = 5,
                                  min_samples_leaf=2,
                                  min_samples_split = 5,
                                  random_state = 100
                                  )
clf_gini.fit(X_train,y_train)
```

Out[101…
```
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=2,
                       min_samples_split=5, random_state=100)
```

In [102…
```
clf_gini.score(X_test,y_test)
```

Out[102…
```
0.819672131147541
```

In [103…
```
from sklearn.metrics import classification_report,confusion_matrix
y_pred= clf_gini.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
                precision    recall  f1-score   support

           0       0.96      0.70      0.81        33
           1       0.73      0.96      0.83        28

    accuracy                           0.82        61
   macro avg       0.84      0.83      0.82        61
weighted avg       0.85      0.82      0.82        61
```

In [104…
```
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
```

```
[[23 10]
 [ 1 27]]
0.819672131147541
```

In [105…
```
print(confusion_matrix(y_test,y_pred_default))
print(accuracy_score(y_test,y_pred_default))
```

```
[[23 10]
 [ 3 25]]
0.7868852459016393
```

In [106…
```
y_pred
```

Out[106…
```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1], dtype=int64)
```