In [1]:
```
#import required packages
import pandas as pd
```

[DATASET (https://www.kaggle.com/abbasit/kyphosis-dataset)](https://www.kaggle.com/abbasit/kyphosis-dataset)

In [2]:

```
df = pd.read_csv("kyphosis.csv")
df
```

Out[2]:

	Kyphosis	Age	Number	Start
0	absent	71	3	5
1	absent	158	3	14
2	present	128	4	5
3	absent	2	5	1
4	absent	1	4	15
5	absent	1	2	16
6	absent	61	2	17
7	absent	37	3	16
8	absent	113	2	16
9	present	59	6	12
10	present	82	5	14
11	absent	148	3	16
12	absent	18	5	2
13	absent	1	4	12
14	absent	168	3	18
15	absent	1	3	16
16	absent	78	6	15
17	absent	175	5	13
18	absent	80	5	16
19	absent	27	4	9
20	absent	22	2	16
21	present	105	6	5
22	present	96	3	12
23	absent	131	2	3
24	present	15	7	2
25	absent	9	5	13
26	absent	8	3	6
27	absent	100	3	14
28	absent	4	3	16
29	absent	151	2	16
...	...	...	...	...
51	absent	9	2	17
52	present	139	10	6

	Kyphosis	Age	Number	Start
53	absent	2	2	17
54	absent	140	4	15
55	absent	72	5	15
56	absent	2	3	13
57	present	120	5	8
58	absent	51	7	9
59	absent	102	3	13
60	present	130	4	1
61	present	114	7	8
62	absent	81	4	1
63	absent	118	3	16
64	absent	118	4	16
65	absent	17	4	10
66	absent	195	2	17
67	absent	159	4	13
68	absent	18	4	11
69	absent	15	5	16
70	absent	158	5	14
71	absent	127	4	12
72	absent	87	4	16
73	absent	206	4	10
74	absent	11	3	15
75	absent	178	4	15
76	present	157	3	13
77	absent	26	7	13
78	absent	120	2	13
79	present	42	7	6
80	absent	36	4	13

81 rows × 4 columns

In [3]:
```
df.shape
```

Out[3]:
```
(81, 4)
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 4 columns):
Kyphosis    81 non-null object
Age         81 non-null int64
Number      81 non-null int64
Start       81 non-null int64
dtypes: int64(3), object(1)
memory usage: 2.6+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	Age	Number	Start
count	81.000000	81.000000	81.000000
mean	83.654321	4.049383	11.493827
std	58.104251	1.619423	4.883962
min	1.000000	2.000000	1.000000
25%	26.000000	3.000000	9.000000
50%	87.000000	4.000000	13.000000
75%	130.000000	5.000000	16.000000
max	206.000000	10.000000	18.000000

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Kyphosis    0
Age         0
Number      0
Start       0
dtype: int64
```

In [7]:
```
df.head()
```

Out[7]:

	Kyphosis	Age	Number	Start
0	absent	71	3	5
1	absent	158	3	14
2	present	128	4	5
3	absent	2	5	1
4	absent	1	4	15

In [8]:
```
df.Kyphosis.value_counts()
```

Out[8]:
```
absent     64
present    17
Name: Kyphosis, dtype: int64
```

In [9]:
```
df.Number.value_counts()
```

Out[9]:
```
3     23
4     18
5     17
2     12
7      5
6      4
10     1
9      1
Name: Number, dtype: int64
```

In [10]:
```
# features
x = df.drop("Kyphosis",axis=1)
```

In [11]:
```
#target
y = df["Kyphosis"]
```

In [12]:
```
# splitting the data for training and testing

from sklearn.model_selection import train_test_split
```

In [13]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_st
```

In [14]:
```
# model
from sklearn.tree import DecisionTreeClassifier
```

In [15]:
```
dtc = DecisionTreeClassifier()
```

In [16]:
```
dtc.fit(x_train,y_train)
```

Out[16]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            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=None,
            splitter='best')
```

In [17]:
```
pred = dtc.predict(x_test)
```

In [18]:
```
pred
```

Out[18]:
```
array(['absent', 'absent', 'present', 'absent', 'absent', 'absent',
       'absent', 'absent', 'absent', 'absent', 'present', 'absent',
       'absent', 'absent', 'present', 'present', 'present', 'absent',
       'absent', 'absent', 'absent', 'absent', 'absent', 'absent',
       'present'], dtype=object)
```

In [19]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_r
```

In [20]:
```
accuracy_score(y_test,pred)
```

Out[20]:
```
0.8
```

In [21]:
```
confusion_matrix(y_test,pred)
```

Out[21]:
```
array([[16,  2],
       [ 3,  4]], dtype=int64)
```

In [22]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

      absent       0.84      0.89      0.86        18
     present       0.67      0.57      0.62         7

   micro avg       0.80      0.80      0.80        25
   macro avg       0.75      0.73      0.74        25
weighted avg       0.79      0.80      0.80        25
```

Decicion Tree Regressor

- Target - continous
- MSE,RMSE,R2_SCORE

[DATASET (https://drive.google.com/file/d/1vcicKFtqFPzR9EsE7_NzElqxF6iB1b0G/view?usp=sharing)](https://drive.google.com/file/d/1vcicKFtqFPzR9EsE7_NzElqxF6iB1b0G/view?usp=sharing)

In [23]:
```
data = pd.read_csv("IceCreamData.csv")
data.head()
```

Out[23]:

	Temperature	Revenue
0	24.566884	534.799028
1	26.005191	625.190122
2	27.790554	660.632289
3	20.595335	487.706960
4	11.503498	316.240194

In [24]:
```
data.shape
```

Out[24]:
```
(500, 2)
```

In [25]:
```
data.isnull().sum()
```

Out[25]:
```
Temperature    0
Revenue        0
dtype: int64
```

In [26]:
```
# spliting data for features and target
data.columns
```

Out[26]:
```
Index(['Temperature', 'Revenue'], dtype='object')
```

In [27]:
```
#feature
x = data[["Temperature"]]
```

In [28]:
```
#target
y = data[["Revenue"]]
```

In [29]:
```
# split the data for training and testing
```

In [30]:
```
from sklearn.model_selection import train_test_split
```

In [31]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_st
```

In [32]:
```
# model
from sklearn.tree import DecisionTreeRegressor
```

In [33]:
```
reg = DecisionTreeRegressor(max_depth=3)
```

In [34]:
```
reg.fit(x_train,y_train)
```

Out[34]:
```
DecisionTreeRegressor(criterion='mse', 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=None, splitter='best')
```

In [35]:
```
# predict the results
y_pred = reg.predict(x_test)
```

In [36]:
```
y_pred
```

Out[36]:
```
array([723.58870803, 632.88586938, 723.58870803, 539.92351342,
       539.92351342, 452.92759761, 632.88586938, 378.70426705,
       861.59281911, 632.88586938, 539.92351342, 539.92351342,
       304.40899262, 452.92759761, 632.88586938, 632.88586938,
       539.92351342, 539.92351342, 632.88586938, 539.92351342,
       539.92351342, 304.40899262, 304.40899262, 304.40899262,
       185.56296183, 378.70426705, 632.88586938, 452.92759761,
       378.70426705, 632.88586938, 452.92759761, 452.92759761,
       632.88586938, 539.92351342, 632.88586938, 378.70426705,
       452.92759761, 539.92351342, 304.40899262, 539.92351342,
       539.92351342, 539.92351342, 452.92759761, 632.88586938,
       632.88586938, 632.88586938, 452.92759761, 304.40899262,
       452.92759761, 539.92351342, 723.58870803, 632.88586938,
       452.92759761, 452.92759761, 632.88586938, 723.58870803,
       185.56296183, 304.40899262, 723.58870803, 632.88586938,
       304.40899262, 723.58870803, 539.92351342, 539.92351342,
       539.92351342, 539.92351342, 452.92759761, 452.92759761,
       539.92351342, 452.92759761, 185.56296183, 632.88586938,
       304.40899262, 539.92351342, 452.92759761, 304.40899262,
       452.92759761, 861.59281911, 632.88586938, 452.92759761,
       861.59281911, 632.88586938, 452.92759761, 452.92759761,
       539.92351342, 723.58870803, 378.70426705, 452.92759761,
       452.92759761, 539.92351342, 632.88586938, 452.92759761,
       539.92351342, 452.92759761, 632.88586938, 378.70426705,
       632.88586938, 861.59281911, 723.58870803, 632.88586938,
       632.88586938, 539.92351342, 632.88586938, 452.92759761,
       539.92351342, 452.92759761, 539.92351342, 632.88586938,
       723.58870803, 452.92759761, 304.40899262, 861.59281911,
       539.92351342, 861.59281911, 378.70426705, 632.88586938,
       185.56296183, 539.92351342, 452.92759761, 539.92351342,
       632.88586938, 304.40899262, 452.92759761, 539.92351342,
       185.56296183, 539.92351342, 632.88586938, 452.92759761,
       539.92351342, 378.70426705, 452.92759761, 723.58870803,
       632.88586938, 723.58870803, 632.88586938, 304.40899262,
       304.40899262, 304.40899262, 539.92351342, 861.59281911,
       185.56296183, 539.92351342, 539.92351342, 304.40899262,
       539.92351342, 861.59281911, 452.92759761, 452.92759761,
       378.70426705, 304.40899262])
```

In [37]:
```
reg.score(x_test,y_pred)
```
Out[37]: 1.0

In [38]:
```
1.0*100
```
Out[38]: 100.0

In [39]:
```
from sklearn.metrics import mean_squared_error
```

In [40]:
```
mean_squared_error(x_test,y_pred)
```
Out[40]: 270065.8700413512

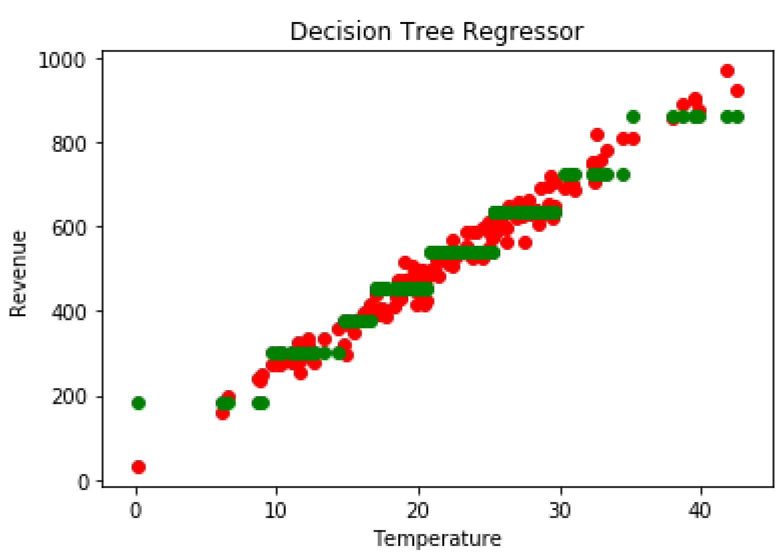
In [41]:
```
import math
```

In [42]:
```
math.sqrt(mean_squared_error(x_test,y_pred))
```
Out[42]: 519.6786218821698

In [43]:
```
import matplotlib.pyplot as plt
```

In [44]:
```
fig, ax = plt.subplots()
ax.scatter(x_test,y_test,color = "red",label = "original data")
ax.scatter(x_test,y_pred,color = "green",label = "predicted data")

plt.title("Decision Tree Regressor")
plt.xlabel("Temperature")
plt.ylabel("Revenue")
plt.show()
```



In [45]:
```
from sklearn.datasets import load_boston
```

In [46]:
```
boston = load_boston()
boston
```

Out[46]:
```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+0
2,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 1
5. ,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
        21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
        35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
        19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
        20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
        23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
        33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
        21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
        20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
        23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
        15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
        17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
        25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
        23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
        32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
        34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
        20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
        26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
        31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
        22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
        42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
        36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
        32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
        20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
        20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
```

```
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DI
S', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n----------------
-----------\n\n**Data Set Characteristics:**  \n\n    :Number of Instances: 506
\n\n    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.\n\n    :Attribute Information (in orde
r):\n        - CRIM     per capita crime rate by town\n        - ZN       propo
rtion of residential land zoned for lots over 25,000 sq.ft.\n        - INDUS
proportion of non-retail business acres per town\n        - CHAS     Charles Ri
ver dummy variable (= 1 if tract bounds river; 0 otherwise)\n        - NOX
nitric oxides concentration (parts per 10 million)\n        - RM       average
number of rooms per dwelling\n        - AGE      proportion of owner-occupied u
nits built prior to 1940\n        - DIS      weighted distances to five Boston
employment centres\n        - RAD      index of accessibility to radial highway
s\n        - TAX      full-value property-tax rate per $10,000\n        - PTRAT
IO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2 where Bk
is the proportion of blacks by town\n        - LSTAT    % lower status of the p
opulation\n        - MEDV     Median value of owner-occupied homes in $1000's\n
\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D. and Rubinfe
ld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttps://archive.ics.uci.
edu/ml/machine-learning-databases/housing/\n\n\nThis dataset was taken from the
StatLib library which is maintained at Carnegie Mellon University.\n\nThe Bosto
n house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the
demand for clean air', J. Environ. Economics & Management,\nvol.5, 81-102, 197
8.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley, 1980.
N.B. Various transformations are used in the table on\npages 244-261 of the lat
ter.\n\nThe Boston house-price data has been used in many machine learning pape
rs that address regression\nproblems.   \n     \n.. topic:: References\n\n   -
Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data an
d Sources of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Comb
ining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Inte
rnational Conference of Machine Learning, 236-243, University of Massachusetts,
Amherst. Morgan Kaufmann.\n",
 'filename': 'C:\\Users\\Alekhya\\Anaconda3\\lib\\site-packages\\sklearn\\datas
ets\\data\\boston_house_prices.csv'}
```

In [47]:

```
x = boston["data"]
```

In [48]:
```
x
```

Out[48]:
```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]])
```

In [49]:
```
x1 = pd.DataFrame(boston["data"],columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', '
         'TAX', 'PTRATIO', 'B', 'LSTAT'])
x1.head()
```

Out[49]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [50]:
```
y = pd.DataFrame(boston["target"],columns=["target"])
```

In [51]:
```
y.head()
```

Out[51]:

	target
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2

In [52]:
```
x1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
dtypes: float64(13)
memory usage: 51.5 KB
```

In [53]:
```
y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 1 columns):
target    506 non-null float64
dtypes: float64(1)
memory usage: 4.0 KB
```

In [54]:
```
x1.isnull().sum()
```

Out[54]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
dtype: int64
```

In [55]:
```
y.isnull().sum()
```

Out[55]:
```
target    0
dtype: int64
```

In [56]:
```python
from sklearn.model_selection import train_test_split
```

In [57]:
```python
X_train,X_test,y_train,y_test = train_test_split(x1,y,test_size=0.3,random_s
```

In [58]:
```python
#Training the Decision Tree Regression model on the training set
# Fitting Decision Tree Regression to the dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(max_depth=3)
regressor.fit(X_train, y_train)
```

Out[58]:
```
DecisionTreeRegressor(criterion='mse', 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=None, splitter='best')
```

In [59]:
```python
#Predicting the Results
y_pred = regressor.predict(X_test)
```

In [60]:
```python
regressor.score(X_test,y_pred)
```

Out[60]:
```
1.0
```