In [1]:

```python
#import dependencies
import numpy as np
import pandas as pd
from sklearn import linear_model
from sklearn.model_selection import train_test_split
```

In [9]:

```python
#Load the Boston Housing data set from sklearn.datasets and print it
from sklearn.datasets import load_boston
boston=load_boston()
print(boston)
```

```
{'data': 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]]), 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
        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', 'IND
US', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', '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 Att
ributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.\n\n    :Attribute Info
rmation (in order):\n        - CRIM     per capita crime rate by town\n        - ZN       proportion of residential l
and zoned for lots over 25,000 sq.ft.\n        - INDUS    proportion of non-retail business acres per town\n        -
CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n        - NOX      nitric oxides conc
entration (parts per 10 million)\n        - RM       average number of rooms per dwelling\n        - AGE      proport
ion of owner-occupied units built prior to 1940\n        - DIS      weighted distances to five Boston employment cent
res\n        - RAD      index of accessibility to radial highways\n        - TAX      full-value property-tax rate pe
r $10,000\n        - PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2 where Bk is the propo
rtion of blacks by town\n        - LSTAT    % lower status of the population\n        - MEDV     Median value of owne
r-occupied homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D. and Rubinfeld, 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 Boston
house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air', J. Environ. Econ
omics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley, 198
0.   N.B. Various transformations are used in the table on\npages 244-261 of the latter.\n\nThe Boston house-price da
```

```
ta has been used in many machine learning papers that address regression\nproblems.   \n     \n.. topic:: References
\n\n   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', W
iley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on th
e Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufman
n.\n", 'filename': 'C:\\Users\\HP\\anaconda3\\lib\\site-packages\\sklearn\\datasets\\data\\boston_house_prices.csv'}
```

In [10]:

```
#Transform the Dataset into a Data Frame
#data= The data we want or the independent variable also known as the x values
#feature_names= The column names of the data
# target=T he target variable or the price of the houses or the dependent variable also known as the y value
df_x=pd.DataFrame(boston.data, columns=boston.feature_names)
df_y=pd.DataFrame(boston.target)
```

In [13]:

```
# Get some statistics from the data set, count , mean, etc.
df_x.describe()
```

Out[13]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.67
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.29
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.32
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.37
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.44
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.22
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.90

In [14]:

```
# Initialize your linear Regression Model
reg= linear_model.LinearRegression()
```

In [15]:

```
#Split the data into 67% training and 33% testing data
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.33, random_state=42)
```

In [16]:

```
#train the model with your training data
reg.fit(x_train, y_train)
```

Out[16]: LinearRegression()

In [17]:
```
#Print the coefficient/weights for each feature/column of your model
print(reg.coef_)
```

```
[[-1.28749718e-01  3.78232228e-02  5.82109233e-02  3.23866812e+00
  -1.61698120e+01  3.90205116e+00 -1.28507825e-02 -1.42222430e+00
   2.34853915e-01 -8.21331947e-03 -9.28722459e-01  1.17695921e-02
  -5.47566338e-01]]
```

In [19]:
```
# print the predictions on your test data
y_pred=reg.predict(x_test)
print(y_pred)
```

```
[[28.53469469]
 [36.6187006 ]
 [15.63751079]
 [25.5014496 ]
 [18.7096734 ]
 [23.16471591]
 [17.31011035]
 [14.07736367]
 [23.01064388]
 [20.54223482]
 [24.91632351]
 [18.41098052]
 [-6.52079687]
 [21.83372604]
 [19.14903064]
 [26.0587322 ]
 [20.30232625]
 [ 5.74943567]
 [40.33137811]
 [17.45791446]
 [27.47486665]
 [30.2170757 ]
 [10.80555625]
 [23.87721728]
 [17.99492211]
 [16.02608791]
 [23.268288  ]
 [14.36825207]
 [22.38116971]
 [19.3092068 ]
```

```
 [22.17284576]
 [25.05925441]
 [25.13780726]
 [18.46730198]
 [16.60405712]
 [17.46564046]
 [30.71367733]
 [20.05106788]
 [23.9897768 ]
 [24.94322408]
 [13.97945355]
 [31.64706967]
 [42.48057206]
 [17.70042814]
 [26.92507869]
 [17.15897719]
 [13.68918087]
 [26.14924245]
 [20.2782306 ]
 [29.99003492]
 [21.21260347]
 [34.03649185]
 [15.41837553]
 [25.95781061]
 [39.13897274]
 [22.96118424]
 [18.80310558]
 [33.07865362]
 [24.74384155]
 [12.83640958]
 [22.41963398]
 [30.64804979]
 [31.59567111]
 [16.34088197]
 [20.9504304 ]
 [16.70145875]
 [20.23215646]
 [26.1437865 ]
 [31.12160889]
 [11.89762768]
 [20.45432404]
 [27.48356359]
 [10.89034224]
 [16.77707214]
 [24.02593714]
```

```
[ 5.44691807]
[21.35152331]
[41.27267175]
[18.13447647]
[ 9.8012101 ]
[21.24024342]
[13.02644969]
[21.80198374]
[ 9.48201752]
[22.99183857]
[31.90465631]
[18.95594718]
[25.48515032]
[29.49687019]
[20.07282539]
[25.5616062 ]
[ 5.59584382]
[20.18410904]
[15.08773299]
[14.34562117]
[20.85155407]
[24.80149389]
[-0.19785401]
[13.57649004]
[15.64401679]
[22.03765773]
[24.70314482]
[10.86409112]
[19.60231067]
[23.73429161]
[12.08082177]
[18.40997903]
[25.4366158 ]
[20.76506636]
[24.68588237]
[ 7.4995836 ]
[18.93015665]
[21.70801764]
[27.14350579]
[31.93765208]
[15.19483586]
[34.01357428]
[12.85763091]
[21.06646184]
[28.58470042]
```

```
 [15.77437534]
 [24.77512495]
 [ 3.64655689]
 [23.91169589]
 [25.82292925]
 [23.03339677]
 [25.35158335]
 [33.05655447]
 [20.65930467]
 [38.18917361]
 [14.04714297]
 [25.26034469]
 [17.6138723 ]
 [20.60883766]
 [ 9.8525544 ]
 [21.06756951]
 [22.20145587]
 [32.2920276 ]
 [31.57638342]
 [15.29265938]
 [16.7100235 ]
 [29.10550932]
 [25.17762329]
 [16.88159225]
 [ 6.32621877]
 [26.70210263]
 [23.3525851 ]
 [17.24168182]
 [13.22815696]
 [39.49907507]
 [16.53528575]
 [18.14635902]
 [25.06620426]
 [23.70640231]
 [22.20167772]
 [21.22272327]
 [16.89825921]
 [23.15518273]
 [28.69699805]
 [ 6.65526482]
 [23.98399958]
 [17.21004545]
 [21.0574427 ]
 [25.01734597]
 [27.65461859]
```

```
 [20.70205823]
 [40.38214871]]
```

In [20]:

```
#print the actual values
print (y_test)
```

```
        0
173  23.6
274  32.4
491  13.6
72   22.8
452  16.1
..    ...
110  21.7
321  23.1
265  22.8
29   21.0
262  48.8

[167 rows x 1 columns]
```

In [23]:

```
# Check the model performance /accuracy using Mean Squared error (MSE)
print(np.mean((y_pred-y_test)**2))
```

```
0    20.724023
dtype: float64
```

In [24]:

```
# Use sklearn.metrics to check accuracy with MSE
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))
```

```
20.724023437339703
```

In [ ]: