



## Machine Learning Pipeline Steps

- Step 1: Data Preprocessing. ...
- Step 2: Data Cleaning. ...
- Step 3: Feature Engineering. ...
- Step 4: Model Selection. ...
- Step 5: Prediction Generation.

## Required Packages

In [324]:

```
1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 %matplotlib inline
6 import warnings
7 warnings.filterwarnings('ignore')
```

## Data Importing

In [325]:

```
1 from sklearn.datasets import load_boston
```

```
In [326]: 1 boston=load_boston()
2 boston
```

```
Out[326]: {'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. , 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 black people by town\n      - LSTAT    % lower status of
the population\n      - MEDV    Median value of owner-occupied homes in $100
0's\n\n :Missing Attribute Values: None\n\n :Creator: Harrison, D. and Ru
binfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\n\n\n\nThis dataset was taken fr
om the StatLib library which is maintained at Carnegie Mellon University.\n\nTh
e Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices
and the demand for clean air', J. Environ. Economics & Management,\nvol.5, 81-1
02, 1978. 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 latter.\n\nThe Boston house-price data has been used in many machine lea
rning papers that address regression\nproblems. \n\n .. topic:: Reference
s\n\n - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influenti
al Data and Sources of Collinearity', Wiley, 1980. 244-261.\n - Quinlan,R. (1
993). Combining Instance-Based and Model-Based Learning. In Proceedings on the
Tenth International Conference of Machine Learning, 236-243, University of Mass
achusetts, Amherst. Morgan Kaufmann.\n",
'filename': 'C:\\\\Users\\\\soume\\\\anaconda3\\\\lib\\\\site-packages\\\\sklearn\\\\dataset
s\\\\data\\\\boston_house_prices.csv'}

```

In [327]:

1 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. , 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]), 'fea-
ture_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD',
'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'), 'DESCR': "... _boston_data
set:\n\nBoston house prices dataset\n-----\n**Data Set
Characteristics:**\n\n :Number of Instances: 506 \n\n :Number of Attrib
utes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually
the target.\n\n :Attribute Information (in order):\n      - CRIM    per c
apita crime rate by town\n      - ZN      proportion of residential land zon
ed for lots over 25,000 sq.ft.\n      - INDUS   proportion of non-retail bus
iness acres per town\n      - CHAS    Charles River dummy variable (= 1 if t
ract 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 units built prior to 1940\n
- DIS      weighted distances to five Boston employment centres\n      - RAD
index of accessibility to radial highways\n      - TAX      full-value proper
ty-tax rate per $10,000\n      - PTRATIO  pupil-teacher ratio by town\n
- B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
\n      - LSTAT    % lower status of the population\n      - MEDV     Media
n value of owner-occupied homes in $1000's\n\n :Missing Attribute Values: No
ne\n\n :Creator: Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI M
L housing dataset.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/h
ousing/\n\nThis dataset was taken from the StatLib library which is maintaine
d 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.
Economics & Management,\nvol.5, 81-102, 1978. 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 latter.\n\nThe Boston house-price da
ta has been used in many machine learning papers that address regression\nprobl
ems. \n  .. topic:: References\n\n      - Belsley, Kuh & Welsch, 'Regressio
n diagnostics: Identifying Influential Data and Sources of Collinearity', Wile
y, 1980. 244-261.\n      - Quinlan,R. (1993). Combining Instance-Based and Model-B
ased Learning. In Proceedings on the Tenth International Conference of Machine
Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.\n",
'filename': 'C:\\\\Users\\\\soume\\\\anaconda3\\\\lib\\\\site-packages\\\\sklearn\\\\datasets
\\\\data\\\\boston_house_prices.csv'}
```

In [328]: 1 boston.keys()

Out[328]: dict\_keys(['data', 'target', 'feature\_names', 'DESCR', 'filename'])

In [329]: 1 print(boston.DESCR)

```

.. _boston_dataset:

Boston house prices dataset
-----
**Data Set Characteristics:**

:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

:Attribute Information (in order):
- CRIM      per capita crime rate by town
- ZN        proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS    proportion of non-retail business acres per town
- CHAS      Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX       nitric oxides concentration (parts per 10 million)
- RM        average number of rooms per dwelling
- AGE       proportion of owner-occupied units built prior to 1940
- DIS       weighted distances to five Boston employment centres
- RAD       index of accessibility to radial highways
- TAX       full-value property-tax rate per $10,000
- PTRATIO   pupil-teacher ratio by town
- B         1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
- LSTAT    % lower status of the population
- MEDV     Median value of owner-occupied homes in $1000's

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://archive.ics.uci.edu/ml/machine-learning-databases/housing/)

```

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression problems.

.. topic:: References

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
- Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.

In [330]: 1 print(boston.data)

```
[[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 [331]: 1 print(boston.target)

```
[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. 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]
```

## Fetures Names

In [332]: 1 print(boston.feature\_names)

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [333]: 1 dataset=pd.DataFrame(boston.data,columns=boston.feature\_names)

In [334]: 1 dataset.head()

Out[334]:

	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 [335]: 1 dataset['Price']=boston.target

In [336]: 1 dataset.head()

Out[336]:

	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

## Check types of dataset

In [337]: 1 dataset.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   CRIM      506 non-null    float64
 1   ZN        506 non-null    float64
 2   INDUS     506 non-null    float64
 3   CHAS      506 non-null    float64
 4   NOX       506 non-null    float64
 5   RM         506 non-null    float64
 6   AGE        506 non-null    float64
 7   DIS        506 non-null    float64
 8   RAD        506 non-null    float64
 9   TAX        506 non-null    float64
 10  PTRATIO   506 non-null    float64
 11  B          506 non-null    float64
 12  LSTAT     506 non-null    float64
 13  Price      506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

## Summary of dataset

In [338]: 1 dataset.describe()

Out[338]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
<b>count</b>	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
<b>mean</b>	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901
<b>std</b>	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861
<b>min</b>	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000
<b>25%</b>	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000
<b>50%</b>	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000
<b>75%</b>	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000
<b>max</b>	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000

## Checking Null Value

In [339]: 1 dataset.isnull().sum()

Out[339]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
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							
Price	0							
dtype:	int64							

## Exploratory Data Analysis

### Summary Of Data

In [340]: 1 dataset.corr()

Out[340]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.
Price	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.

```
In [341]: 1 import seaborn as sns
```

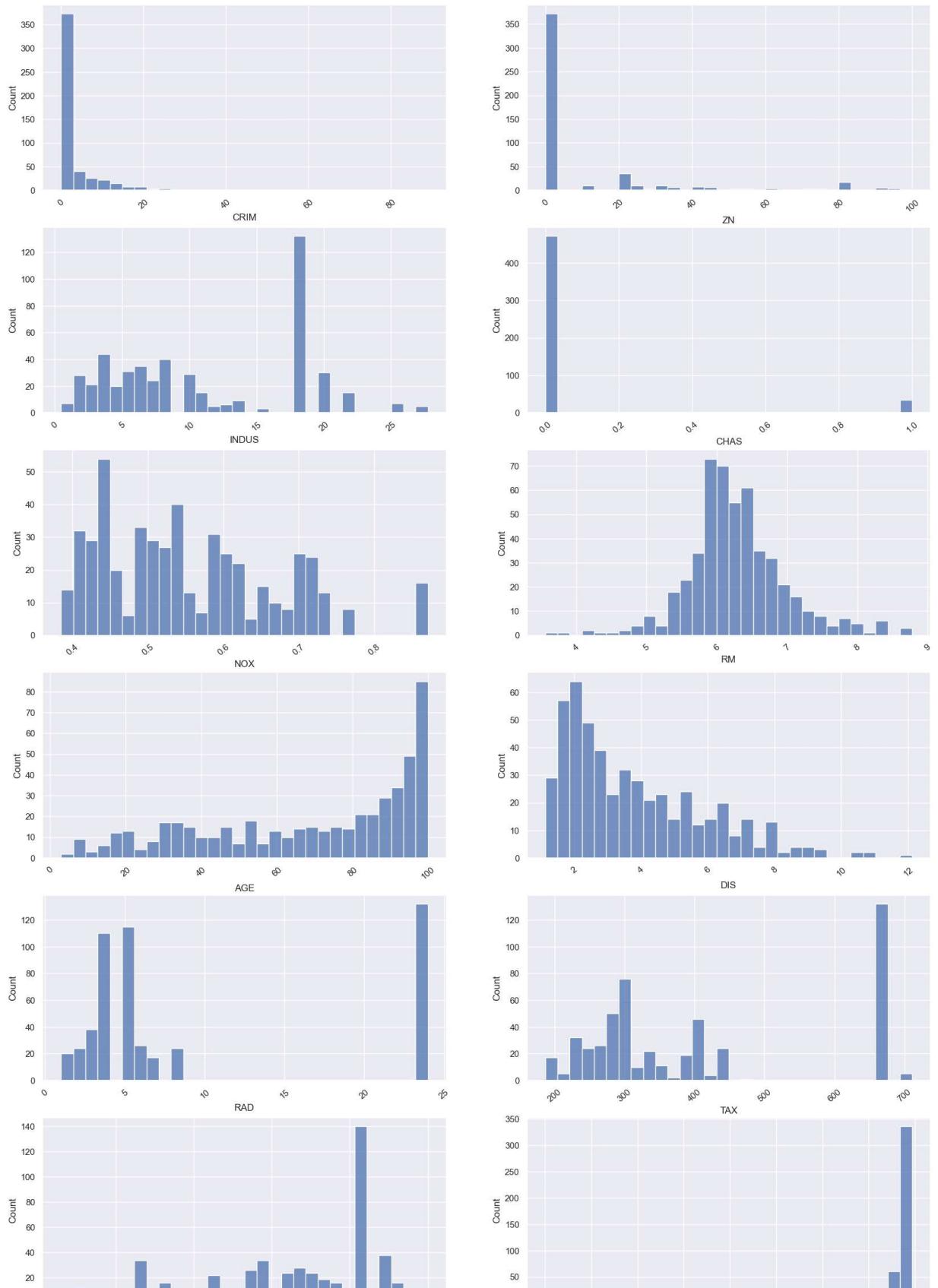
## Visualize distribution off all independent features

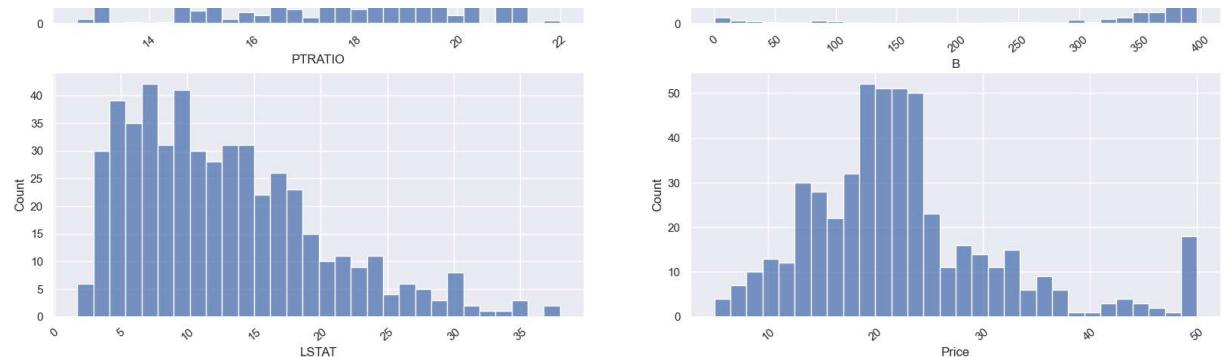
In [397]:

```

1 plt.figure(figsize=(20,100))
2 for i in enumerate(dataset):
3     plt.subplot(20, 2, i[0]+1)
4     sns.set(rc={'figure.figsize':(5,5)})
5     sns.histplot(data=dataset, x=i[1], bins=30)
6     plt.xticks(rotation=40)

```





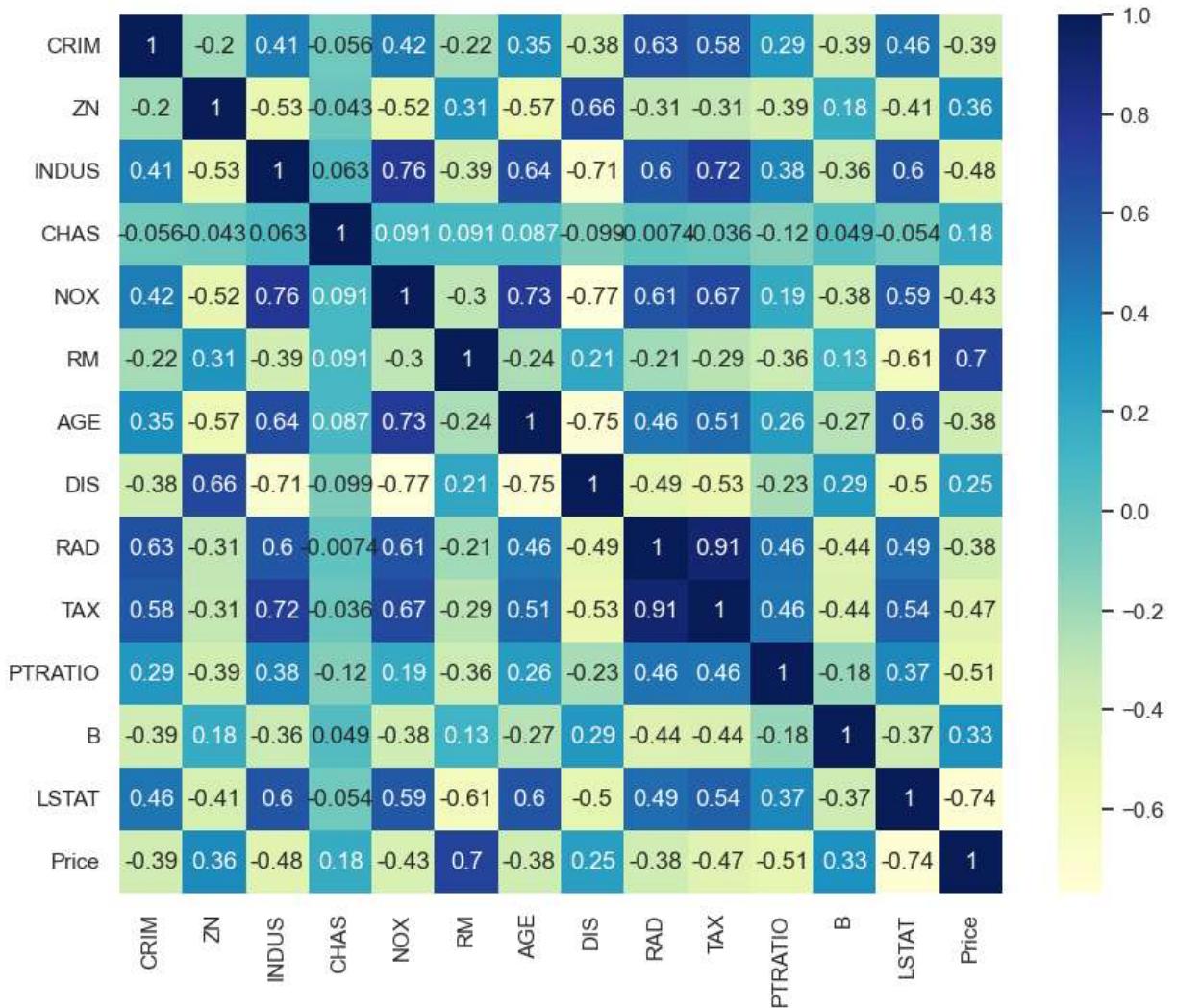
In [ ]:

1

## Visualize correlation All Variables

```
In [343]: 1 sns.set(rc={'figure.figsize':(10,8)})
2 sns.heatmap(dataset.corr(), annot=True, cmap="YlGnBu")
```

Out[343]: &lt;AxesSubplot:&gt;

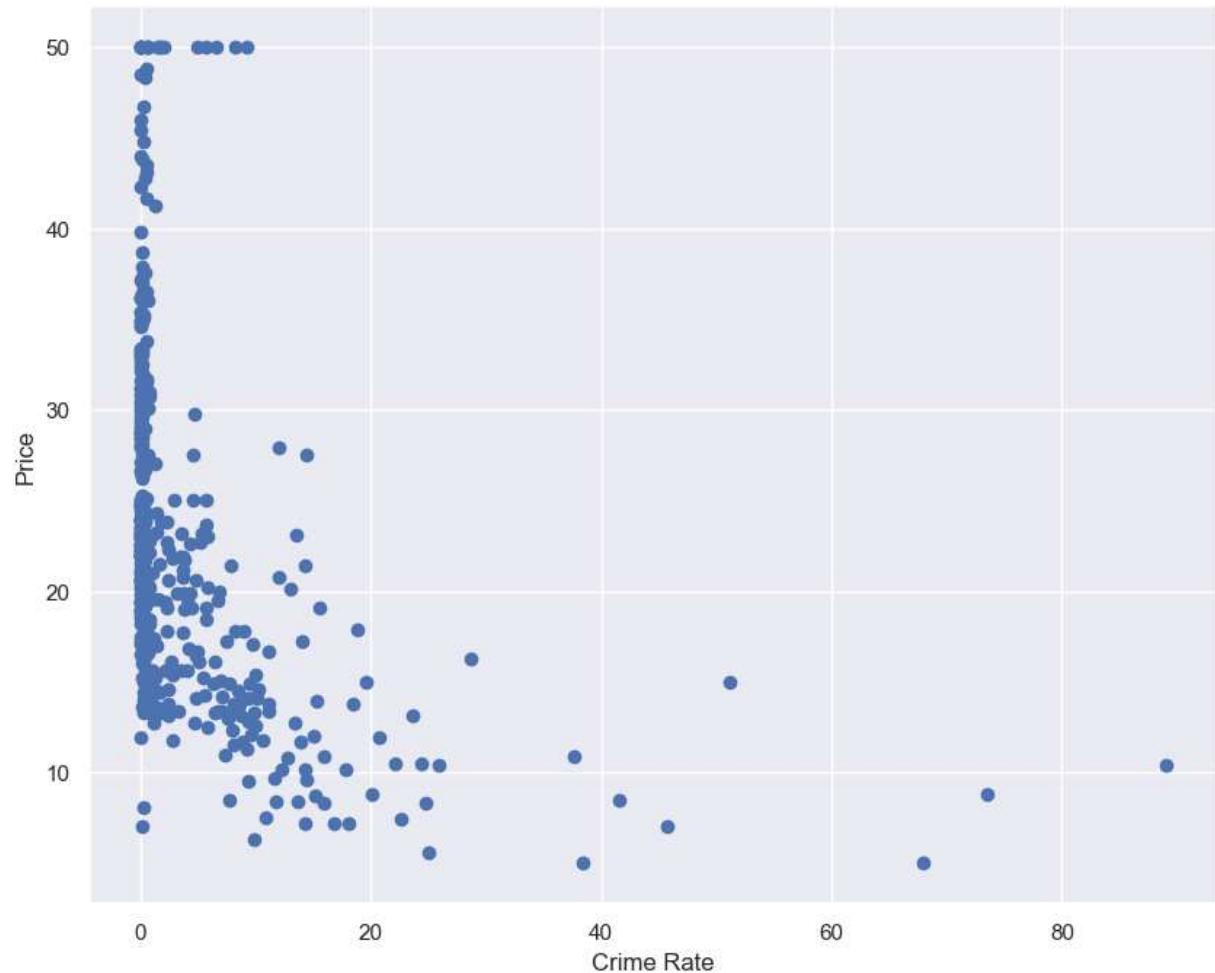


## Scatter plot for visualize the relation of independent variables with

## dependent variables

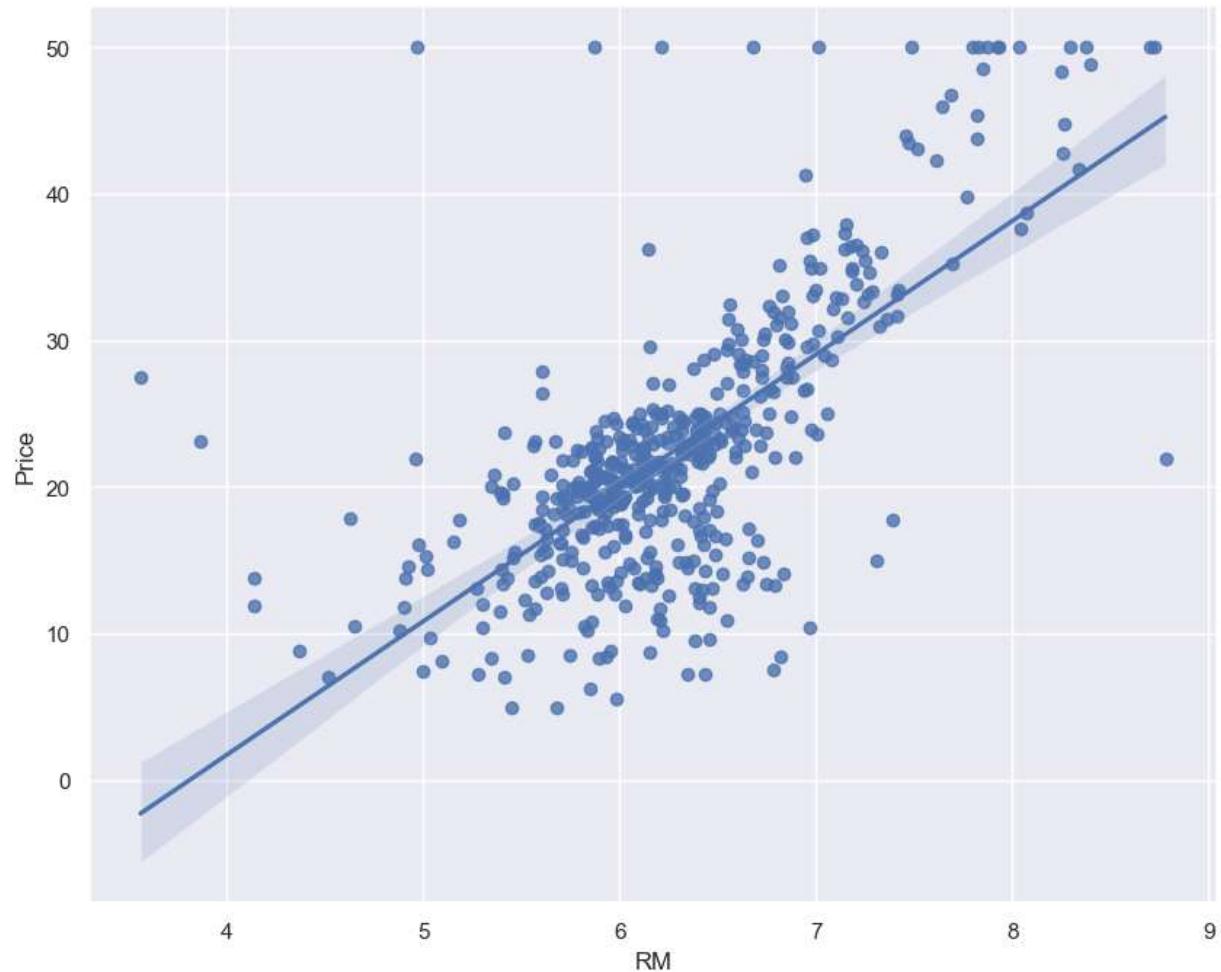
```
In [344]: 1 plt.scatter(dataset['CRIM'],dataset['Price'])  
2 plt.xlabel("Crime Rate")  
3 plt.ylabel("Price")
```

Out[344]: Text(0, 0.5, 'Price')



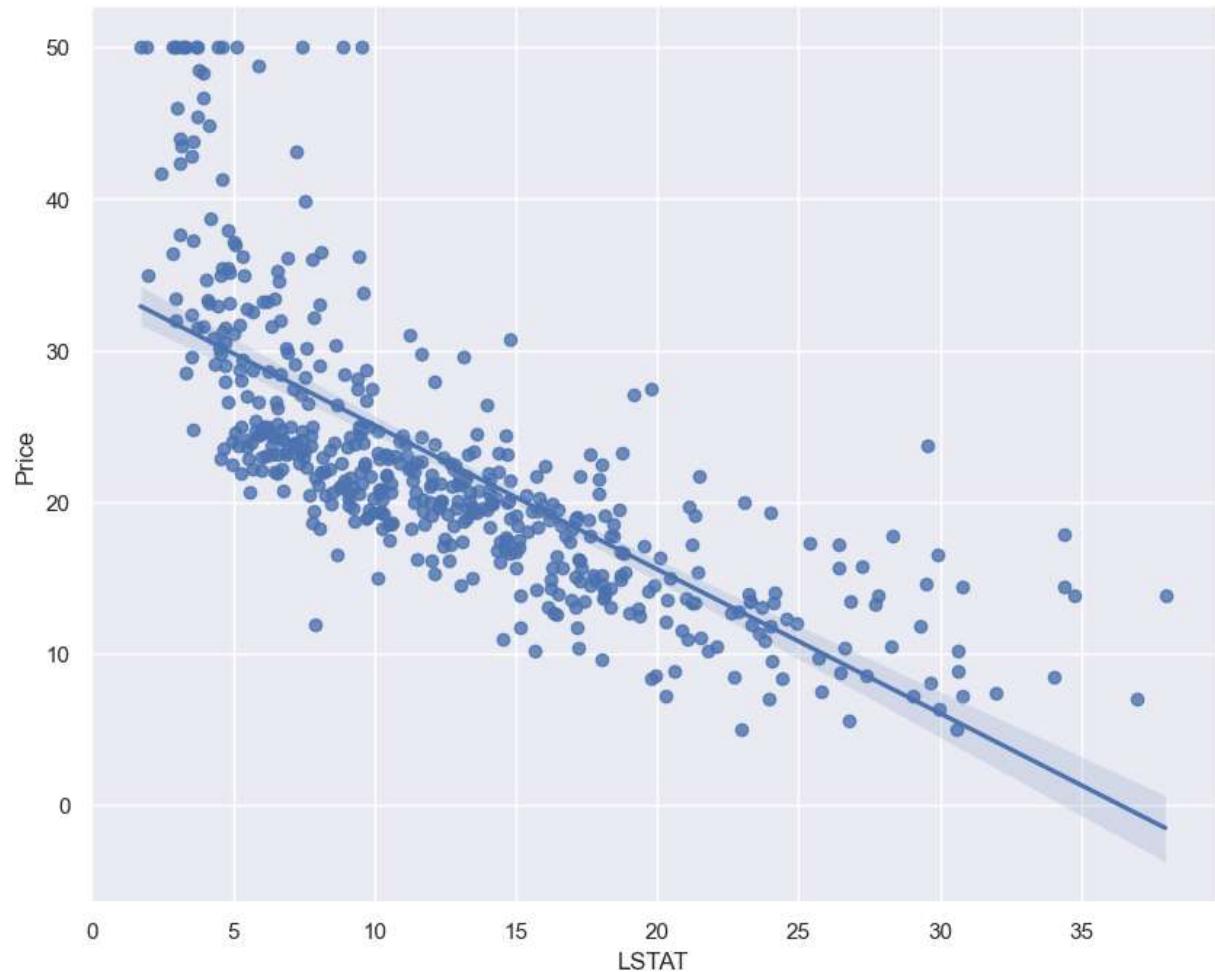
```
In [345]: 1 sns.regplot(x="RM",y="Price",data=dataset)
```

```
Out[345]: <AxesSubplot:xlabel='RM', ylabel='Price'>
```



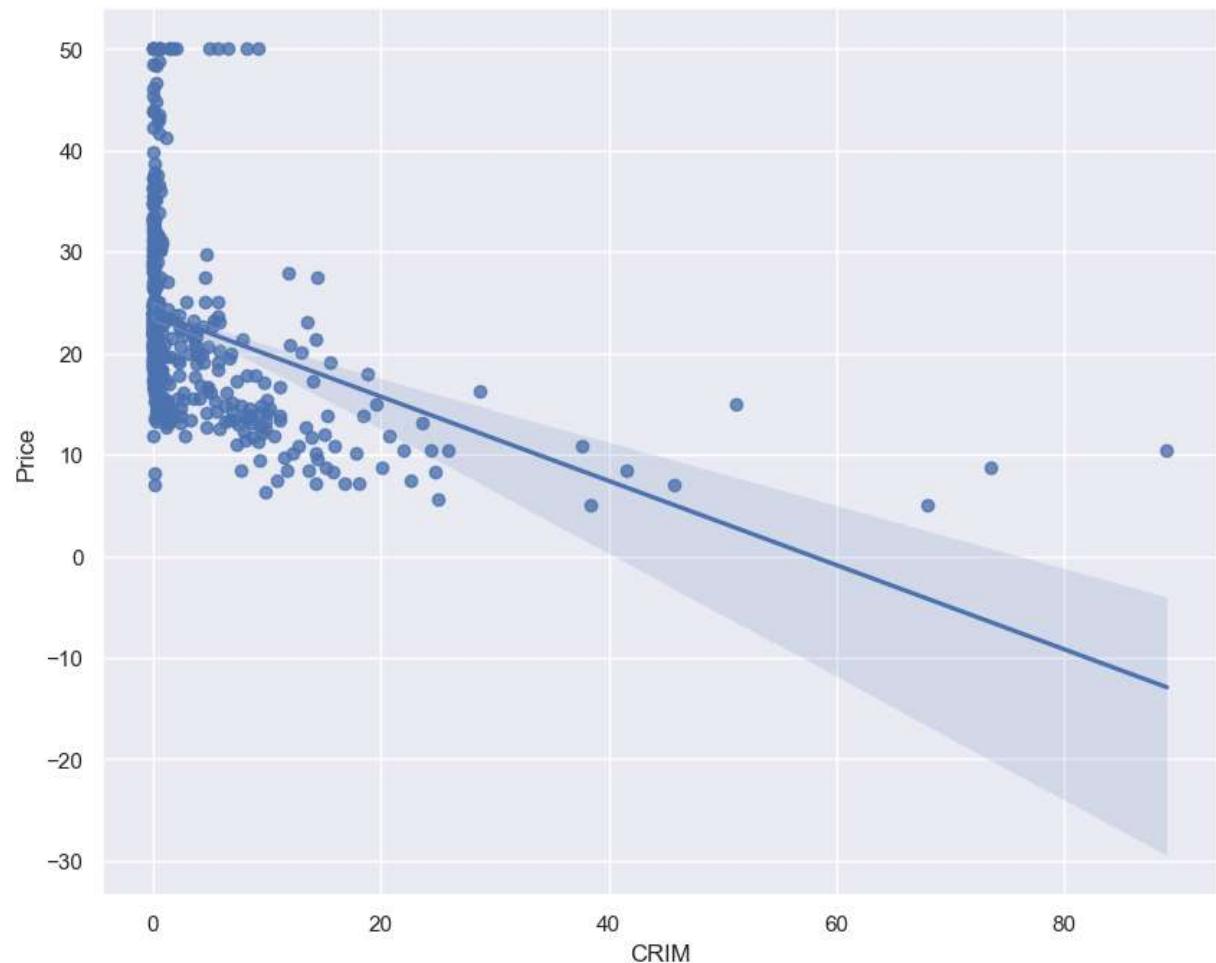
```
In [346]: 1 sns.regplot(x="LSTAT",y="Price",data=dataset)
```

```
Out[346]: <AxesSubplot:xlabel='LSTAT', ylabel='Price'>
```



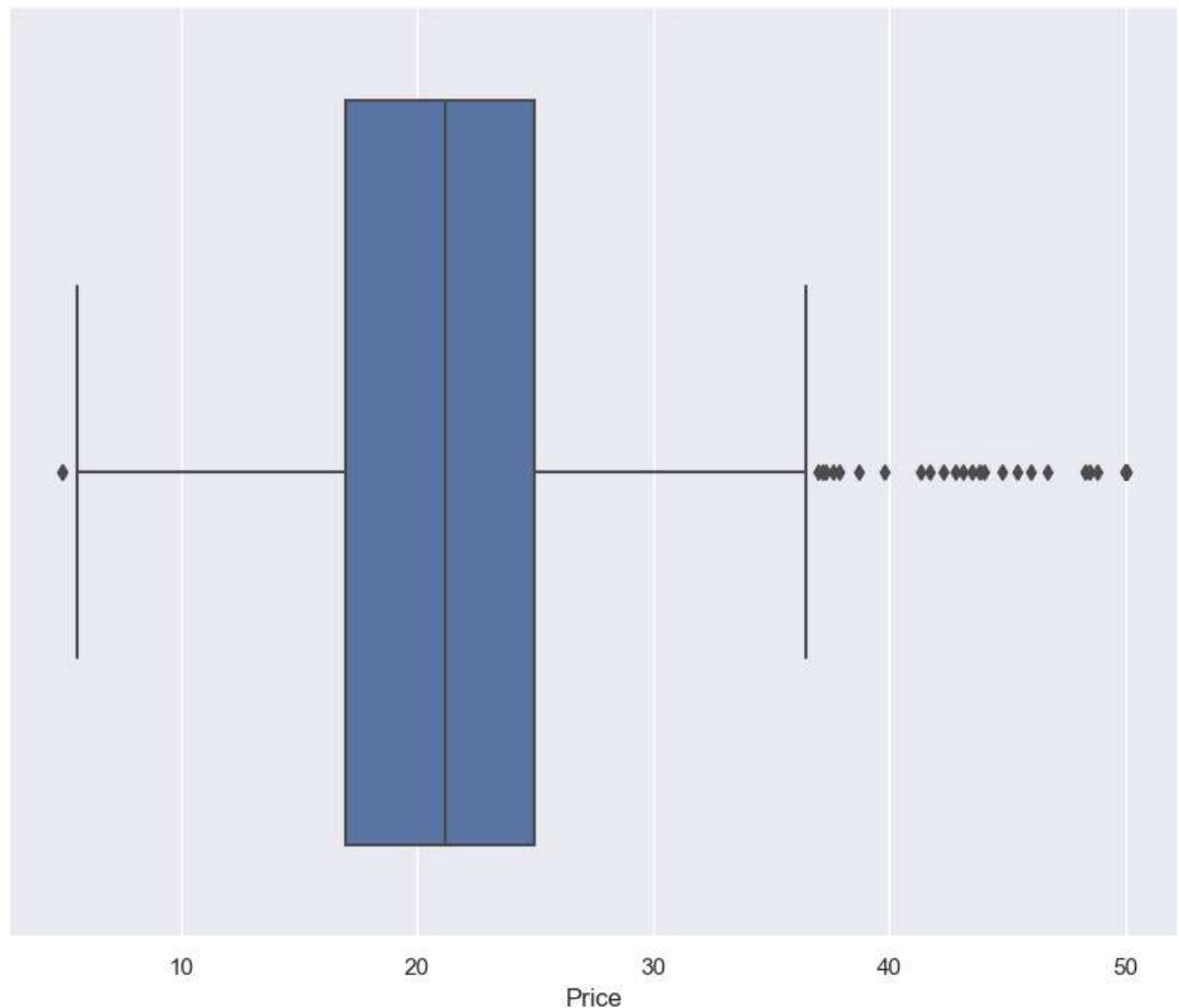
```
In [347]: 1 sns.regplot(x="CRIM",y="Price",data=dataset)
```

```
Out[347]: <AxesSubplot:xlabel='CRIM', ylabel='Price'>
```



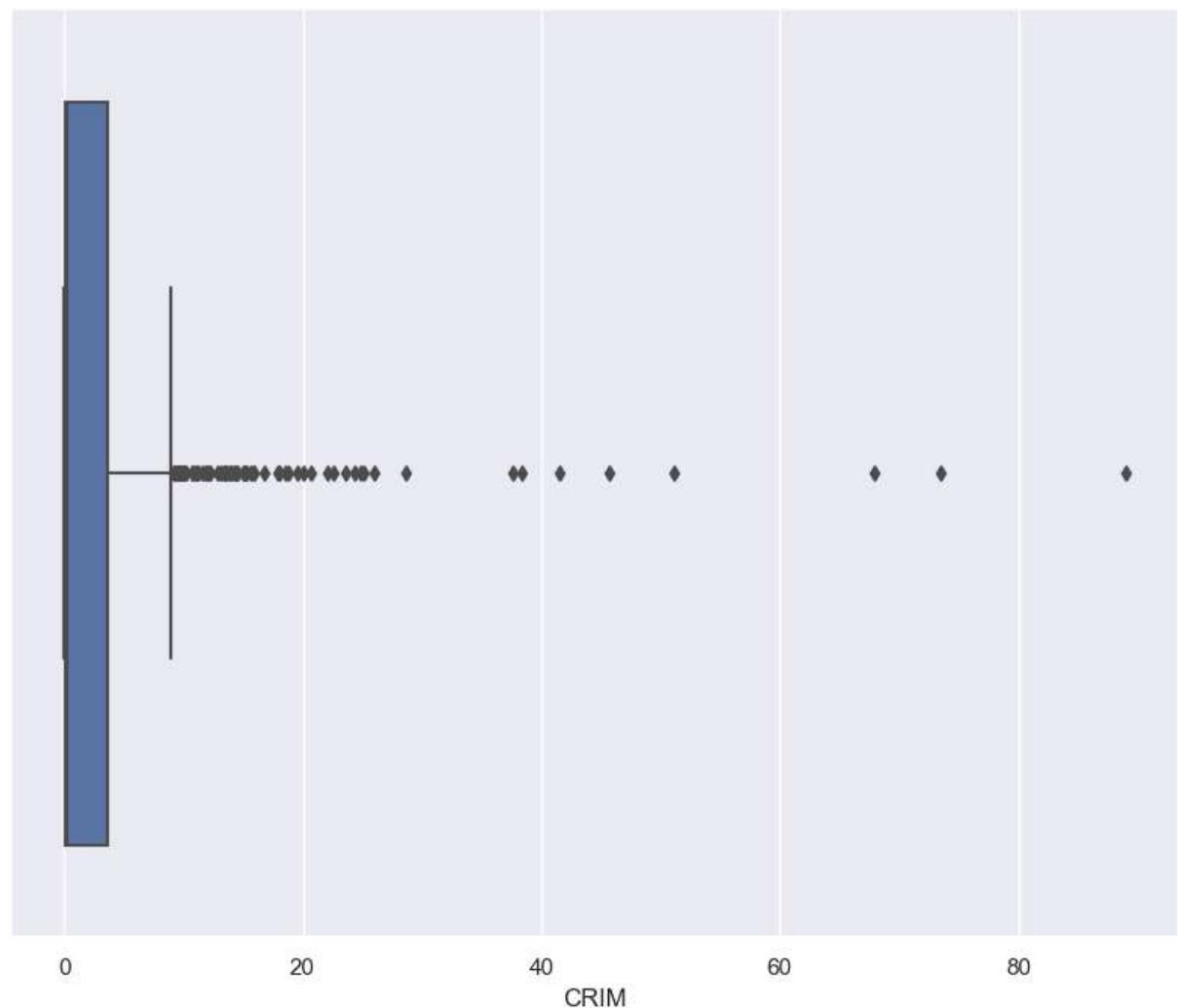
```
In [348]: 1 sns.boxplot(dataset['Price'])  
2 # But it is output we can't do anything to output variable
```

Out[348]: <AxesSubplot:xlabel='Price'>



```
In [349]: 1 sns.boxplot(dataset['CRIM'])
```

```
Out[349]: <AxesSubplot:xlabel='CRIM'>
```



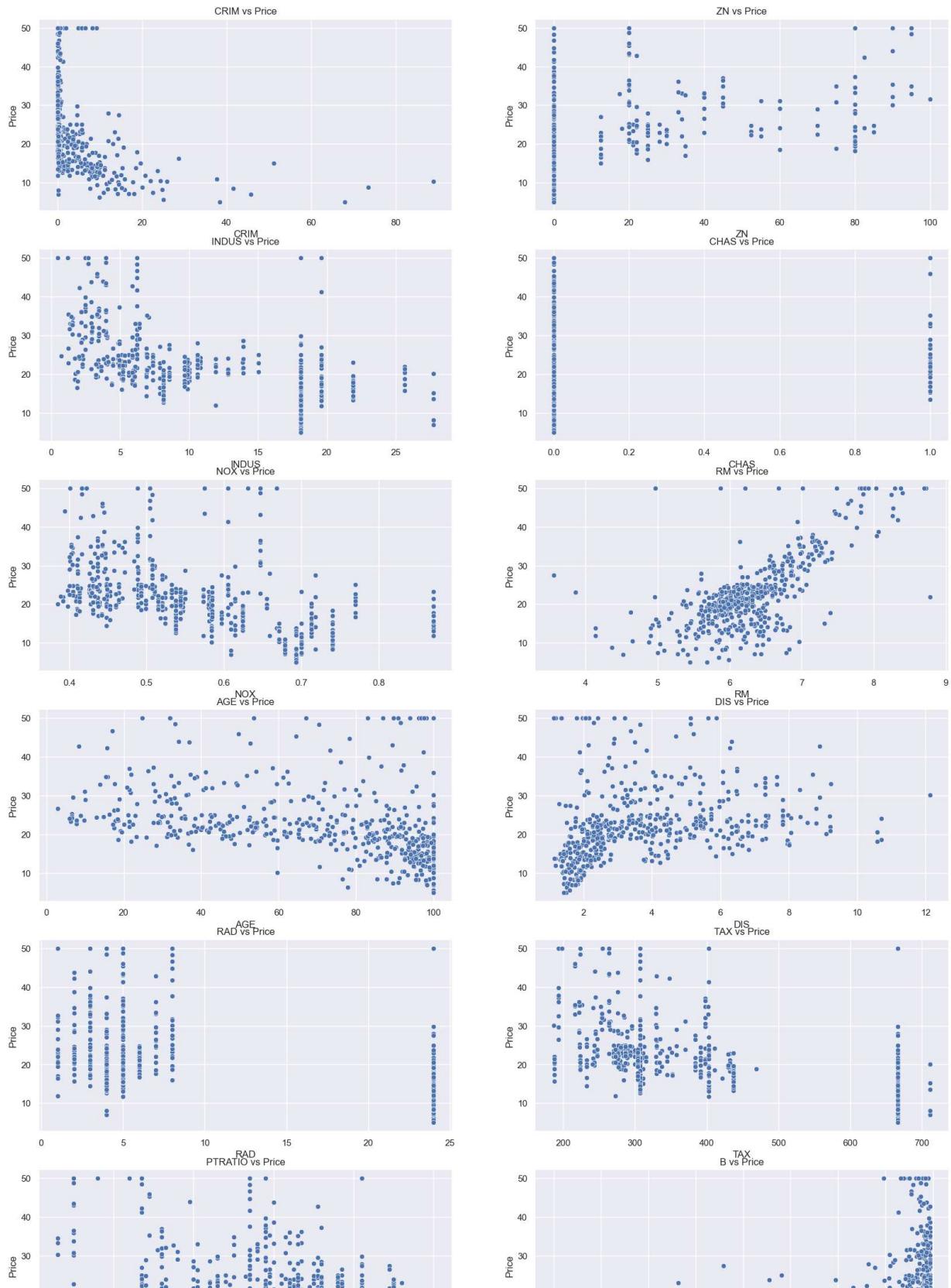
**Visualize the relations between independent variables with Price**

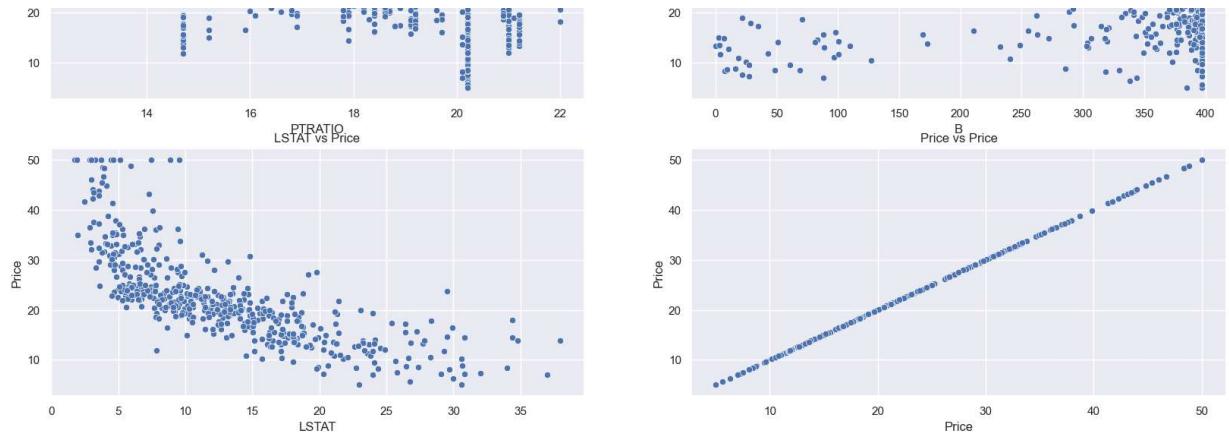
In [400]:

```

1 plt.figure(figsize=(20,50))
2 for feature in enumerate(dataset):
3     plt.subplot(10,2,feature[0]+1)
4     sns.scatterplot(data=dataset, x=dataset[feature[1]], y=dataset['Price'])
5     plt.xlabel(feature[1])
6     plt.ylabel('Price')
7     plt.title("{} vs Price".format(feature[1]))

```





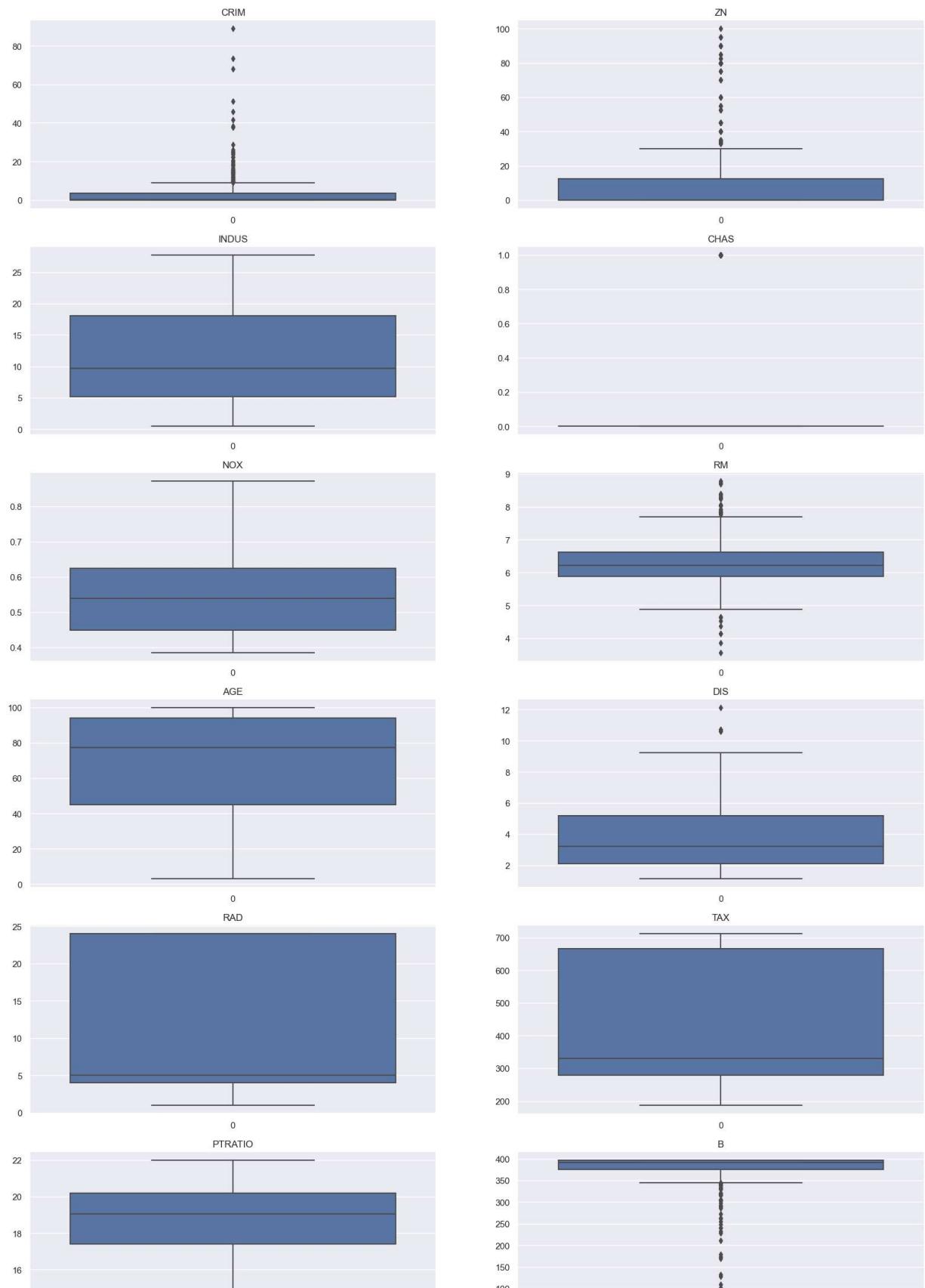
## Boxplot of all variables

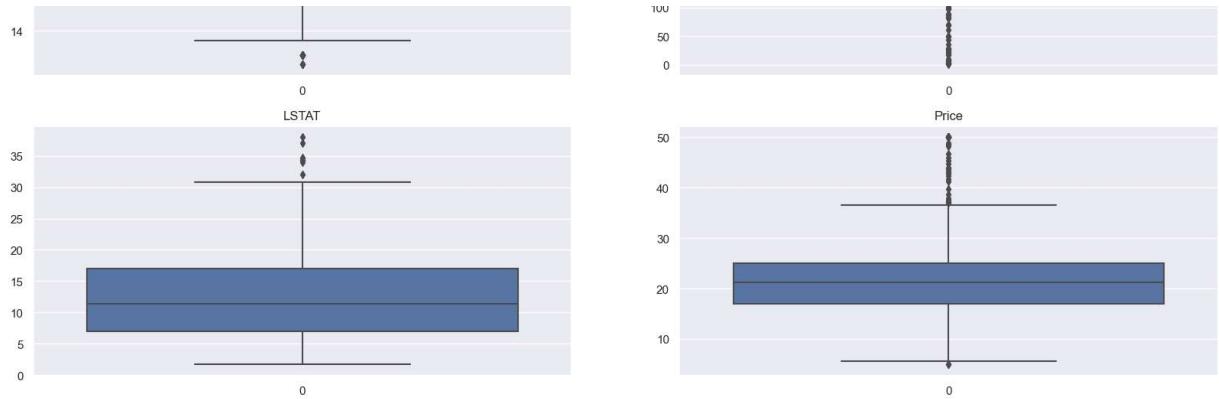
In [399]:

```

1  ### getting outliers
2  plt.figure(figsize=(20,50))
3  for feature in enumerate(dataset):
4      plt.subplot(10,2,feature[0]+1)
5      sns.boxplot(data=dataset[feature[1]])
6      plt.title(feature[1])

```





In [350]: 1 dataset.head()

Out[350]:

	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

## Independent and Dependent features

In [351]: 1  
2 X=dataset.iloc[:, :-1]  
3 y=dataset.iloc[:, -1]

In [352]: 1 X.head()

Out[352]:

	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 [353]: 1 y

Out[353]:

0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
	...
501	22.4
502	20.6
503	23.9
504	22.0
505	11.9

Name: Price, Length: 506, dtype: float64

## Splitting the data

In [354]: 1 from sklearn.model\_selection import train\_test\_split

In [355]:

1	X_train, X_test, y_train, y_test = train_test_split(
2	X, y, test_size=0.33, random_state=10)

In [356]: 1 X\_train

Out[356]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
147	2.36862	0.0	19.58	0.0	0.871	4.926	95.7	1.4608	5.0	403.0	14.7	391.71	2
330	0.04544	0.0	3.24	0.0	0.460	6.144	32.2	5.8736	4.0	430.0	16.9	368.57	
388	14.33370	0.0	18.10	0.0	0.700	4.880	100.0	1.5895	24.0	666.0	20.2	372.92	3
238	0.08244	30.0	4.93	0.0	0.428	6.481	18.5	6.1899	6.0	300.0	16.6	379.41	
113	0.22212	0.0	10.01	0.0	0.547	6.092	95.4	2.5480	6.0	432.0	17.8	396.90	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
320	0.16760	0.0	7.38	0.0	0.493	6.426	52.3	4.5404	5.0	287.0	19.6	396.90	
15	0.62739	0.0	8.14	0.0	0.538	5.834	56.5	4.4986	4.0	307.0	21.0	395.62	
484	2.37857	0.0	18.10	0.0	0.583	5.871	41.9	3.7240	24.0	666.0	20.2	370.73	1
125	0.16902	0.0	25.65	0.0	0.581	5.986	88.4	1.9929	2.0	188.0	19.1	385.02	1
265	0.76162	20.0	3.97	0.0	0.647	5.560	62.8	1.9865	5.0	264.0	13.0	392.40	1

339 rows × 13 columns



In [357]: 1 y\_train

Out[357]:

147	14.6
330	19.8
388	10.2
238	23.7
113	18.7
...	
320	23.8
15	19.9
484	20.6
125	21.4
265	22.8

Name: Price, Length: 339, dtype: float64

In [358]: 1 X\_test  
2

Out[358]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSI
305	0.05479	33.0	2.18	0.0	0.472	6.616	58.1	3.3700	7.0	222.0	18.4	393.36	8
193	0.02187	60.0	2.93	0.0	0.401	6.800	9.9	6.2196	1.0	265.0	15.6	393.37	5
65	0.03584	80.0	3.37	0.0	0.398	6.290	17.8	6.6115	4.0	337.0	16.1	396.90	4
349	0.02899	40.0	1.25	0.0	0.429	6.939	34.5	8.7921	1.0	335.0	19.7	389.85	5
151	1.49632	0.0	19.58	0.0	0.871	5.404	100.0	1.5916	5.0	403.0	14.7	341.60	13
...	...	...	...	...	...	...	...	...	...	...	...	...	...
442	5.66637	0.0	18.10	0.0	0.740	6.219	100.0	2.0048	24.0	666.0	20.2	395.69	16
451	5.44114	0.0	18.10	0.0	0.713	6.655	98.2	2.3552	24.0	666.0	20.2	355.29	17
188	0.12579	45.0	3.44	0.0	0.437	6.556	29.1	4.5667	5.0	398.0	15.2	382.84	4
76	0.10153	0.0	12.83	0.0	0.437	6.279	74.5	4.0522	5.0	398.0	18.7	373.66	11
314	0.36920	0.0	9.90	0.0	0.544	6.567	87.3	3.6023	4.0	304.0	18.4	395.69	9

167 rows × 13 columns



```
In [359]: 1 y_test
```

```
Out[359]: 305    28.4
193    31.1
65     23.5
349    26.6
151    19.6
...
442    18.4
451    15.2
188    29.8
76     20.0
314    23.8
Name: Price, Length: 167, dtype: float64
```

```
In [360]: 1 X_train.shape
```

```
Out[360]: (339, 13)
```

```
In [361]: 1 y_train.shape
```

```
Out[361]: (339, )
```

```
In [362]: 1 X_test.shape
```

```
Out[362]: (167, 13)
```

```
In [363]: 1 y_test.shape
```

```
Out[363]: (167, )
```

## Why do we need to perform feature scaling?

Scaling the features makes the flow of gradient descent smooth and helps algorithms quickly reach the minima of the cost function. Without scaling features, the algorithm may be biased toward the feature which has values higher in magnitude.

## Feature Scaling

```
In [370]: 1
2 from sklearn.preprocessing import StandardScaler
3 scaler=StandardScaler()
```

```
In [371]: 1 scaler
```

```
Out[371]: StandardScaler()
```

```
In [372]: 1 X_train_scaled=scaler.fit_transform(X_train)
           2 X_train_scaled
```

```
Out[372]: array([[-0.13641471, -0.47928013,  1.16787606, ..., -1.77731527,
                   0.39261401,  2.36597873],
                  [-0.41777807, -0.47928013, -1.18043314, ..., -0.75987458,
                   0.14721899, -0.54115799],
                  [ 1.31269177, -0.47928013,  0.95517731, ...,  0.76628645,
                   0.19334986,  2.52100705],
                  ...,
                  [-0.13520965, -0.47928013,  0.95517731, ...,  0.76628645,
                   0.17012536,  0.06331026],
                  [-0.40281114, -0.47928013,  2.04022838, ...,  0.25756611,
                   0.32166792,  0.27238516],
                  [-0.33104058,  0.34161649, -1.07552092, ..., -2.56351944,
                   0.39993132, -0.34772815]])
```

```
In [373]: 1 X_test_scaled=scaler.transform(X_test)
           2 X_test_scaled
```

```
Out[373]: array([[-0.41664568,  0.87519929, -1.33277144, ..., -0.06616502,
                   0.41011193, -0.56391444],
                  [-0.42063267,  1.98340973, -1.22498491, ..., -1.36108953,
                   0.41021798, -1.11860295],
                  [-0.41894074,  2.80430634, -1.16175014, ..., -1.12985301,
                   0.44765291, -1.16980497],
                  ...,
                  [-0.40804678,  1.36773726, -1.15169007, ..., -1.54607875,
                   0.29854946, -1.18545003],
                  [-0.41098494, -0.47928013,  0.19779729, ...,  0.07257689,
                   0.20119741, -0.13154186],
                  [-0.37856708, -0.47928013, -0.22328875, ..., -0.06616502,
                   0.43482111, -0.5141347 ]])
```

## Model Training

```
In [374]: 1 from sklearn.linear_model import LinearRegression
```

```
In [375]: 1 regression=LinearRegression()
```

```
In [376]: 1 regression
```

```
Out[376]: LinearRegression()
```

```
In [377]: 1 regression.fit(X_train_scaled,y_train)
```

```
Out[377]: LinearRegression()
```

## Print the coefficients and intercept

In [378]: 1 print(regression.coef\_)

```
[ -1.29099218  1.60949999 -0.14031574  0.37201867 -1.76205329  2.22752218
 0.32268871 -3.31184248  2.70288107 -2.09005699 -1.7609799  1.25191514
-3.83392028]
```

In [379]: 1 print(regression.intercept\_)

```
22.077286135693214
```

In [380]: 1 ## Prediction for the test data
2 y\_pred=regression.predict(X\_test\_scaled)
3 y\_pred

Out[380]: array([31.43849583, 31.98794389, 30.99895559, 22.31396689, 18.89492791,
 16.21371128, 35.9881236 , 14.81264582, 25.04500847, 37.12806894,
 21.49110158, 30.88757187, 28.05752881, 34.05600093, 33.75791114,
 40.63880011, 24.24023412, 23.41351375, 25.54158122, 21.34135664,
 32.71699711, 17.88341061, 25.49549436, 25.01006418, 32.54102925,
 20.48979076, 19.48816948, 16.92733183, 38.38530857, 0.36265208,
 32.42715816, 32.15306983, 26.10323665, 23.79611814, 20.67497128,
 19.69393973, 3.50784614, 35.26259797, 27.04725425, 27.66164435,
 34.35132103, 29.83057837, 18.40939436, 31.56953795, 17.91877807,
 28.50042742, 19.49382421, 21.69553078, 38.0954563 , 16.44490081,
 24.58507284, 19.67889486, 24.53954813, 34.30610423, 26.74699088,
 34.87803562, 21.06219662, 19.87980936, 18.68725139, 24.71786624,
 19.96344041, 23.56002479, 39.57630226, 42.81994338, 30.37060855,
 17.03737245, 23.83719412, 3.2425022 , 31.5046382 , 28.63779884,
 18.49288659, 27.14115768, 19.67125483, 25.34222917, 25.05430467,
 10.29463949, 38.96369453, 8.26774249, 18.52214761, 30.34082002,
 22.87681099, 20.96680268, 20.04604103, 28.73415756, 30.81726786,
 28.23002473, 26.28588806, 31.59181918, 22.13093608, -6.48201197,
 21.53000756, 19.90826887, 24.96686716, 23.44746617, 19.28521216,
 18.75729874, 27.40013804, 22.17867402, 26.82972 , 23.39779064,
 23.9260607 , 19.16632572, 21.09732823, 11.01452286, 13.7692535 ,
 20.74596484, 23.54892211, 14.04445469, 28.88171403, 15.77611741,
 15.25195598, 22.429474 , 26.60737213, 28.88742175, 24.29797261,
 18.26839956, 16.26943281, 17.40100292, 15.53131616, 21.27868825,
 33.78464602, 30.00899396, 21.16115702, 13.95560661, 16.18475215,
 29.30998858, 13.1866784 , 22.08393725, 24.34499386, 31.86829501,
 33.45923602, 5.90671516, 35.20153265, 24.17614831, 17.54200544,
 24.25032915, 28.44671354, 34.50123773, 6.33164665, 1.93565618,
 28.40727267, 12.56461105, 18.31045646, 19.71015745, 5.50105857,
 14.51366874, 37.193992 , 25.81821367, 23.31632083, 26.43254504,
 11.38255141, 20.46224115, 35.27645709, 20.57841598, 11.48799917,
 16.23913171, 24.56511742, 10.53131603, 15.07115005, 25.98488217,
 11.2136222 , 11.695686 , 19.40437966, 19.58768384, 32.43800883,
 22.66170871, 25.68576052])

## Assumption of linear regression

- Linearity: The relationship between X and the mean of Y is linear.
- Homoscedasticity: The variance of residual is the same for any value of X.

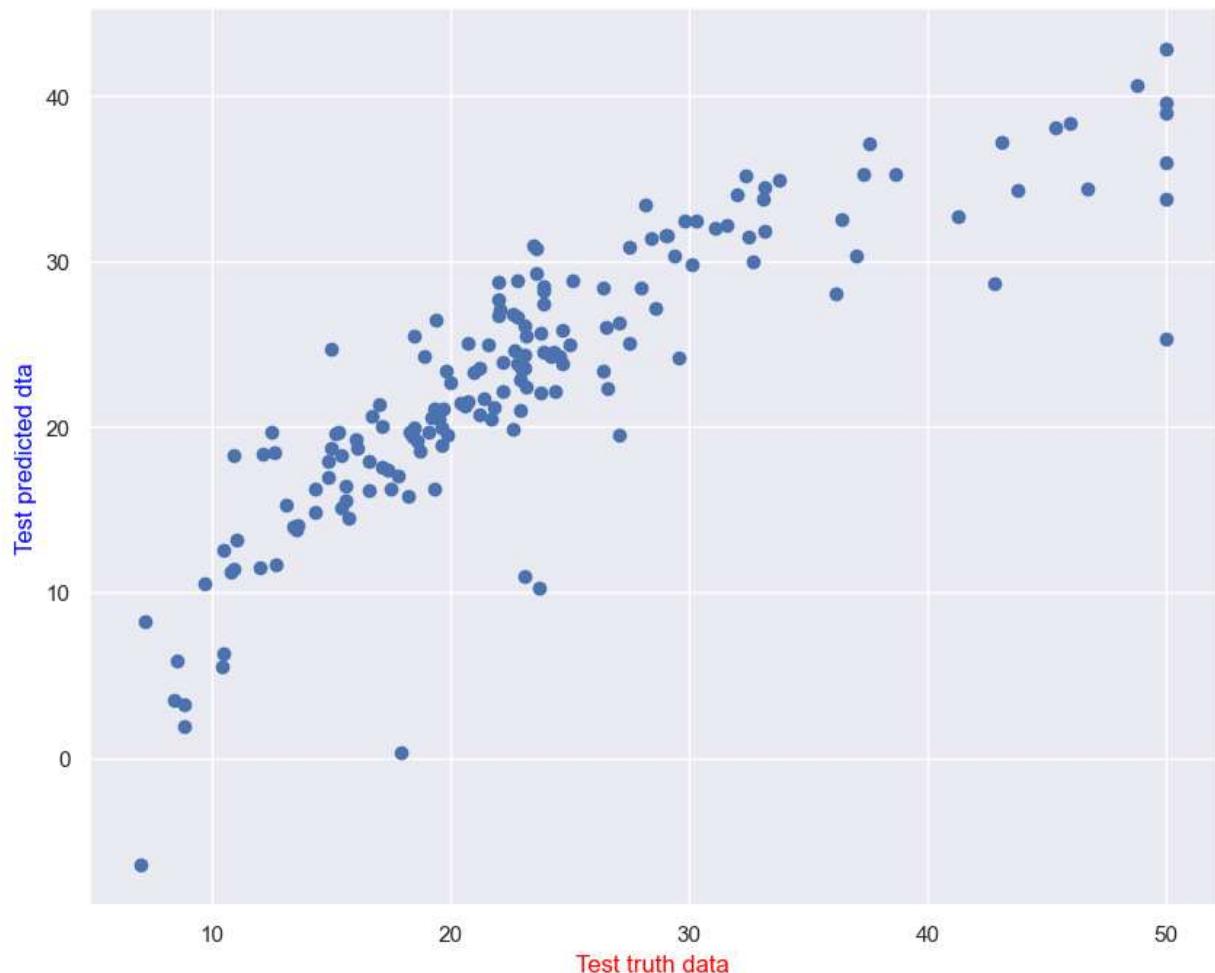
- Independence: Observations are independent of each other.
- Normality: For any fixed value of X, Y is normally distributed

## Showing assumption of linear regression

In [382]:

```
1 plt.scatter(y_test,y_pred)
2 plt.xlabel("Test truth data",color='red')
3 plt.ylabel("Test predicted dta",color='blue')
```

Out[382]: Text(0, 0.5, 'Test predicted dta')



## Residuals

In [384]:

```
1 residuals=y_test-y_pred
2 residuals
```

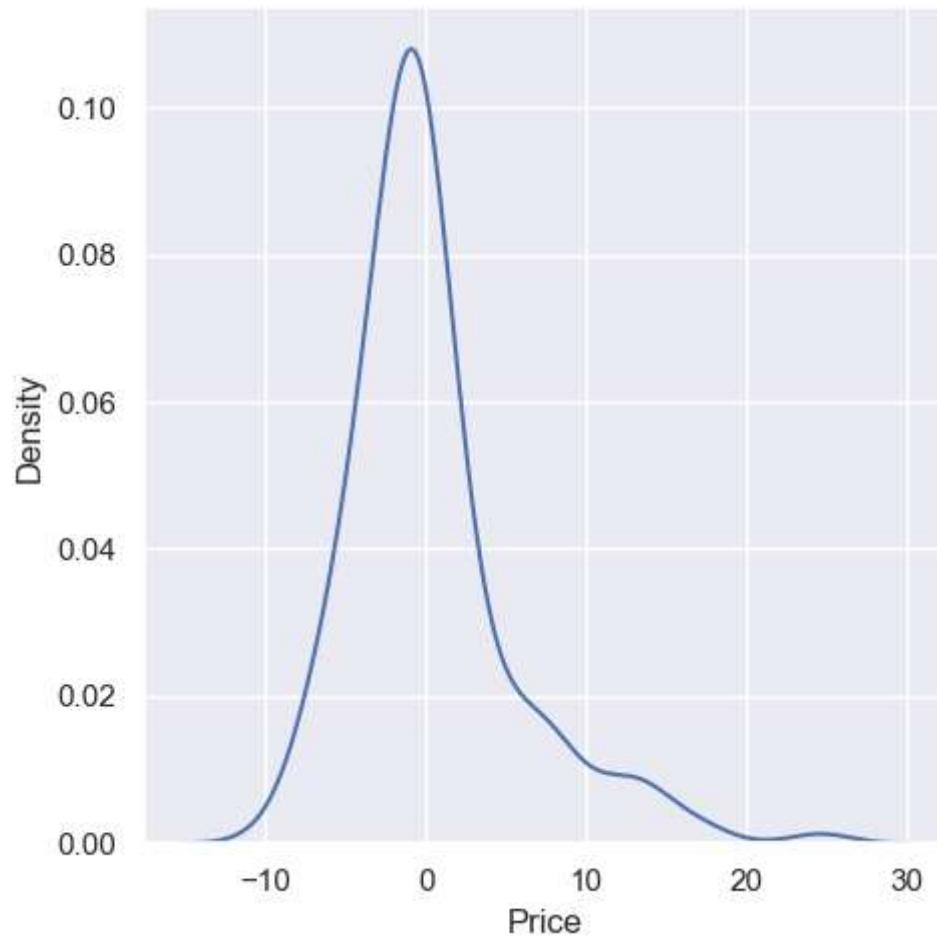
Out[384]:

```
305    -3.038496
193    -0.887944
65     -7.498956
349     4.286033
151     0.705072
...
442    -1.004380
451    -4.387684
188    -2.638009
76     -2.661709
314    -1.885761
Name: Price, Length: 167, dtype: float64
```

In [385]:

```
1 sns.displot(residuals,kind="kde")
```

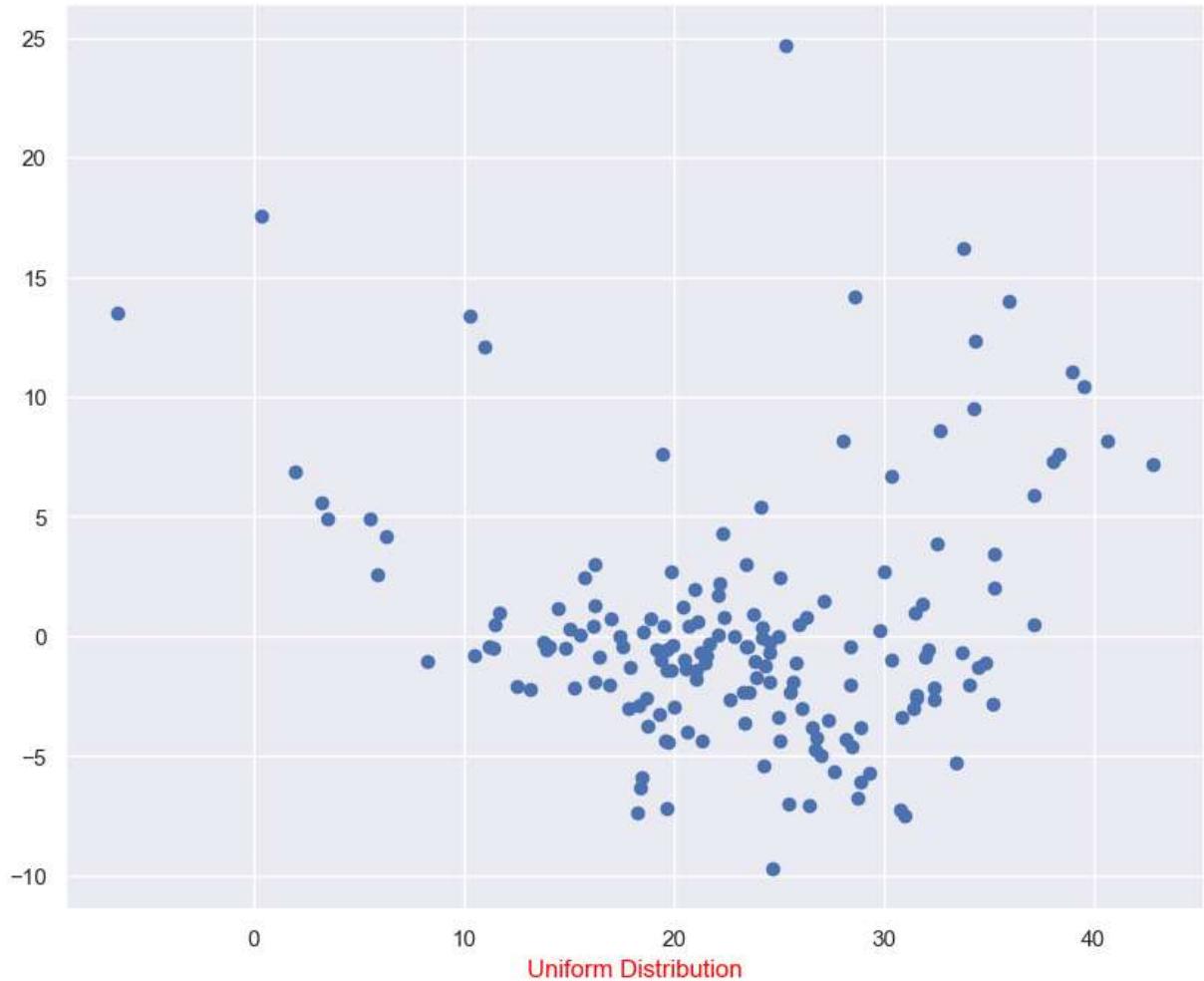
Out[385]:



## Scatter plot with predictions and residual

```
In [386]: 1 ##uniform distribution
2 plt.scatter(y_pred,residuals)
3 plt.xlabel("Uniform Distribution",color="red")
```

Out[386]: Text(0.5, 0, 'Uniform Distribution')



## Performance Metrics for Linear Regression

```
In [387]: 1
2 from sklearn.metrics import mean_squared_error
3 from sklearn.metrics import mean_absolute_error
4 print(mean_squared_error(y_test,y_pred))
5 print(mean_absolute_error(y_test,y_pred))
6 print(np.sqrt(mean_squared_error(y_test,y_pred)))
```

27.100991709962464  
3.5206585298797903  
5.205861284164462

## R2-score for Linear Regression

```
In [388]: 1 from sklearn.metrics import r2_score
2 score=r2_score(y_test,y_pred)
3 print(score)
```

0.7165219393967557

## Adjusted R2-score for Linear Regression

```
In [389]: 1
2 1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[389]: 0.6924355682343886

## Ridge Regression Algorithm

```
In [282]: 1 from sklearn.linear_model import Ridge
2 from sklearn.model_selection import GridSearchCV
3 ridge=Ridge()
```

```
In [283]: 1 parameters={'alpha':[1,2,5,10,20,30,40]}
2 ridge_cv=GridSearchCV(ridge,parameters,scoring='neg_mean_squared_error',cv=5
3 ridge_cv.fit(X_train,y_train)
```

Out[283]: GridSearchCV(cv=5, estimator=Ridge(),  
                   param\_grid={'alpha': [1, 2, 5, 10, 20, 30, 40]},  
                   scoring='neg\_mean\_squared\_error')

```
In [284]: 1 print(ridge_cv.best_params_)

{'alpha': 20}
```

```
In [285]: 1 print(ridge_cv.best_score_)
```

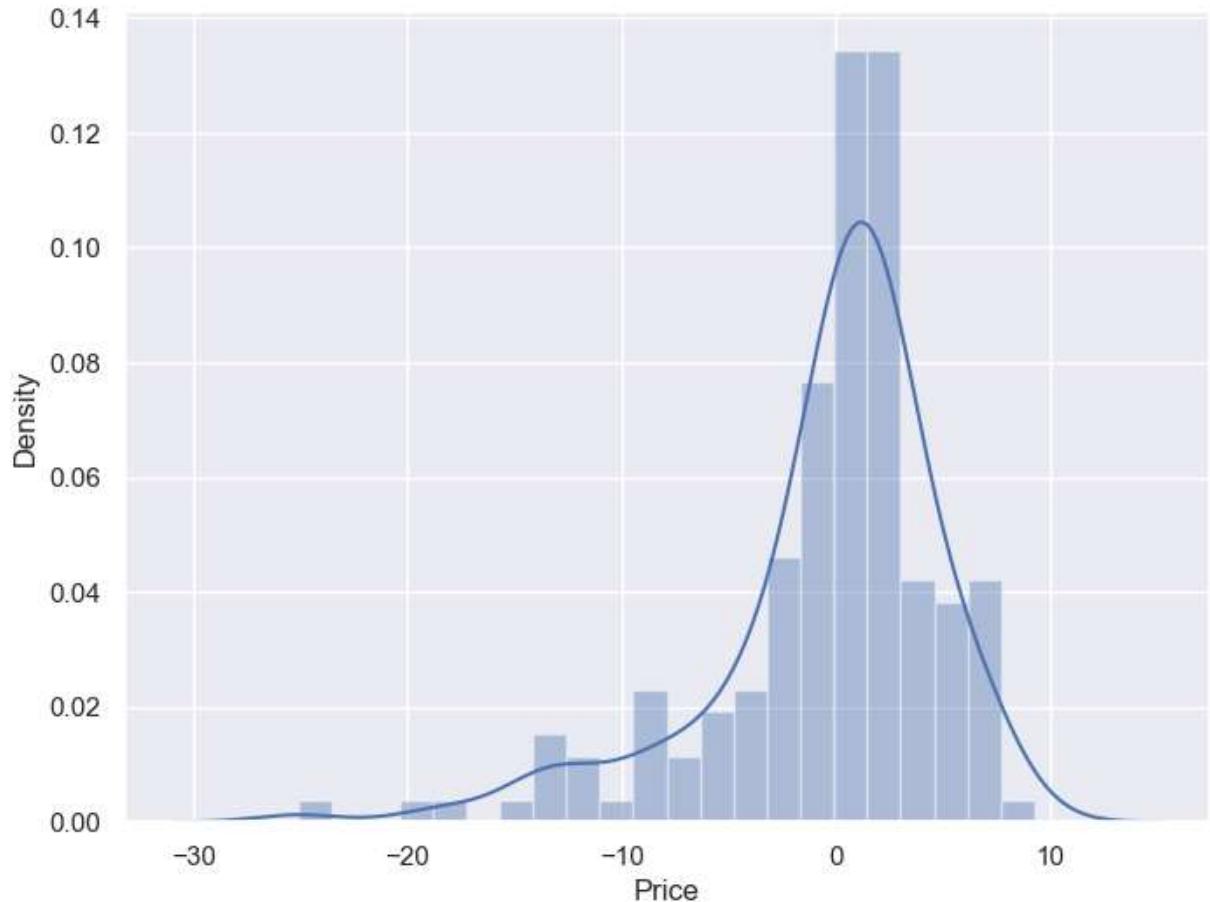
-23.021620343340334

```
In [286]: 1 ridge_pred=ridge_cv.predict(X_test)
           2 ridge_pred
```

```
Out[286]: array([31.26250761, 31.72290483, 30.9482677 , 23.15867751, 22.25207719,
       16.82744046, 35.26664846, 14.98348017, 23.87927736, 36.11118446,
       21.00392778, 29.36139443, 27.95902936, 33.76278925, 32.85296518,
       40.90685235, 25.12697028, 23.20141137, 25.67103585, 23.49079297,
       32.02212489, 18.78042937, 25.0004098 , 25.48350707, 32.20705998,
       21.72626817, 19.09311882, 17.80473397, 36.76605279, -1.05689757,
       31.56113713, 31.42806064, 26.87630221, 24.0774368 , 19.70088155,
       20.30755411, 3.69997843, 35.09603395, 27.51365246, 27.37312046,
      34.03932594, 29.35684059, 18.3537049 , 31.36143401, 17.03462275,
      28.82567767, 21.15770121, 21.17060424, 36.93350532, 16.87224671,
      24.21333027, 19.69519647, 24.2950541 , 33.37874195, 27.26857497,
      35.55289346, 20.91394843, 21.83460211, 19.42815235, 24.29315415,
      20.18730891, 23.22065682, 37.54489308, 42.90441818, 30.51917427,
      16.8520877 , 23.47720241, 3.76799959, 31.21212223, 28.73003775,
      19.57699737, 26.22502542, 19.03771218, 24.93132161, 24.73703876,
      9.58294029, 37.48384988, 8.29250257, 18.95827242, 30.02736951,
      23.06646751, 21.94730537, 20.19533964, 27.92047708, 30.37141261,
      28.64444688, 25.21311313, 31.33937825, 23.73728289, -6.37473794,
      23.07678073, 20.14259342, 24.59220624, 24.22293716, 19.22773183,
      18.30036538, 26.87402786, 20.93395572, 26.69003644, 22.82756535,
      23.50112988, 18.80464904, 20.98837545, 11.60180284, 13.68670403,
      19.49331889, 23.08934096, 12.82208464, 29.10950427, 17.82376362,
      14.76640174, 21.4984877 , 26.02906139, 27.72086331, 24.38576877,
      17.80562663, 14.46362868, 17.064275 , 18.54066971, 21.04748615,
      32.56607519, 29.89584462, 21.93850748, 15.70375227, 16.84090273,
      28.80223608, 13.1241004 , 21.73109155, 24.54284098, 30.79138065,
      33.10901161, 6.18732059, 34.1349616 , 24.90402178, 18.34260117,
      23.66214585, 27.89403419, 33.09345682, 6.84251541, 2.25281317,
      28.22428819, 13.57766042, 19.06915977, 22.59569727, 5.55831457,
      13.51697127, 37.79130918, 26.71604798, 23.84084432, 26.21222254,
      11.86307697, 20.71775583, 33.88953051, 20.66784136, 10.71590985,
      16.61177676, 24.28684107, 11.01241617, 17.91805152, 25.8243196 ,
      11.99455383, 12.4134269 , 20.39613222, 19.99959372, 31.8272145 ,
      21.01079405, 25.71396763])
```

```
In [287]: 1 import seaborn as sns  
2 sns.distplot(ridge_pred_y_test)
```

Out[287]: <AxesSubplot:xlabel='Price', ylabel='Density'>



## Performance Metrics for Ridge Regression

```
In [288]: 1 from sklearn.metrics import mean_squared_error
2 from sklearn.metrics import mean_absolute_error
3 print(mean_squared_error(y_test,ridge_pred))
4 print(mean_absolute_error(y_test,ridge_pred))
5 print(np.sqrt(mean_squared_error(y_test,ridge_pred)))
```

29.28721653718942  
 3.6971299961099455  
 5.411766489529035

## R2 Score of Ridge Rgression

```
In [289]: 1 from sklearn.metrics import r2_score
2 ridge_score=r2_score(y_test,ridge_pred)
3 print(score)
```

0.7165219393967557

## Adjusted R2-Score of Ridge Rgression

```
In [290]: 1 1-(1-ridge_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[290]: 0.6676244836846394

## Lasso Rgression

```
In [291]: 1 from sklearn.linear_model import Lasso
```

```
In [292]: 1 lasso=Lasso()
```

```
In [293]: 1 parameters={'alpha':[1,2,5,10,20,30,40]}
2 lasso_cv=GridSearchCV(lasso,parameters,scoring='neg_mean_squared_error',cv=5)
3 lasso_cv.fit(X_train,y_train)
```

Out[293]: GridSearchCV(cv=5, estimator=Lasso(),  
 param\_grid={'alpha': [1, 2, 5, 10, 20, 30, 40]},  
 scoring='neg\_mean\_squared\_error')

```
In [294]: 1 print(lasso_cv.best_params_)
```

{'alpha': 1}

```
In [295]: 1 lasso_pred=lasso_cv.predict(X_test)
           2 lasso_pred
```

```
Out[295]: array([30.05062227, 30.75863386, 31.79136406, 23.81132585, 24.46450905,
       16.51502359, 33.55687901, 16.18590265, 23.63422119, 32.67378583,
       20.62340113, 26.88616937, 27.50950406, 33.13800373, 31.26216912,
       35.51177715, 25.1322652 , 24.37702007, 25.3426104 , 22.79399868,
       31.42535013, 17.87518719, 24.62140298, 25.67592917, 30.8570178 ,
       23.19651817, 19.54532952, 17.48554848, 33.51934661, 1.0746952 ,
       30.2290915 , 29.6311516 , 28.3746209 , 24.09051453, 17.8996287 ,
       20.68626471, 2.03311885, 33.92194066, 27.84246939, 26.98859872,
       29.66201241, 28.76027267, 16.66430744, 31.3070688 , 17.76180501,
       28.8142132 , 20.84880796, 21.08094697, 33.95869027, 18.19713582,
       25.3790762 , 17.83141295, 24.62557761, 29.23048683, 27.09040407,
       32.30973738, 21.12865331, 21.8682558 , 21.45083035, 21.4927721 ,
       20.82460129, 22.70209723, 32.64112207, 36.46204926, 29.01296777,
       16.60412861, 24.21223336, 6.15819233, 31.29938975, 25.66665157,
       19.12096902, 26.07635104, 19.09894122, 25.20532246, 23.72293349,
       7.46519487, 31.78184104, 7.93417039, 21.66650902, 28.82264119,
       24.04887629, 23.99760225, 21.73647125, 27.1081459 , 29.34325776,
       28.04599659, 25.75769079, 30.4155572 , 25.14780952, -3.2483763 ,
       24.33980236, 22.19039081, 25.32648911, 24.73588935, 21.71166789,
       18.17949212, 27.40772395, 20.30800437, 26.99320873, 22.2873327 ,
       24.23833743, 19.64886888, 22.15222613, 18.57341744, 14.18514138,
       19.70755856, 23.78098739, 13.03503894, 29.45659639, 22.47550945,
       12.90394483, 20.25757185, 24.61575068, 26.37500077, 25.18758983,
       15.73313472, 14.49559824, 18.87614435, 20.19492705, 21.29726476,
       30.81860198, 28.94638256, 22.17757046, 14.71215667, 18.79736922,
       28.12286666, 12.19429838, 22.97031693, 24.30773588, 27.93884477,
       31.73300601, 6.81523194, 32.55092938, 25.22113859, 16.92807435,
       23.27550401, 26.92401426, 30.59006878, 8.93970768, 5.84737618,
       27.80566228, 13.58229434, 17.59584967, 24.93011863, 6.99103487,
       13.70103998, 34.3311626 , 29.01410928, 24.59277709, 27.81462485,
       13.65501713, 20.71519689, 30.01636162, 20.82701755, 11.58795903,
       18.72787631, 23.99187497, 12.02982796, 18.1481676 , 24.76157021,
       11.85369437, 13.24565577, 19.77471126, 18.29296243, 30.94733281,
       21.26773868, 25.73858979])
```

## Performance Metrics for Lasso Regression

```
In [296]: 1 from sklearn.metrics import mean_squared_error
           2 from sklearn.metrics import mean_absolute_error
           3 print(mean_squared_error(y_test,ridge_pred))
           4 print(mean_absolute_error(y_test,ridge_pred))
           5 print(np.sqrt(mean_squared_error(y_test,ridge_pred)))
```

29.28721653718942  
 3.6971299961099455  
 5.411766489529035

## R2-Score of Lasso Regression

```
In [297]: 1 from sklearn.metrics import r2_score  
2 lasso_score=r2_score(y_test,lasso_pred)  
3 print(score)
```

0.7165219393967557

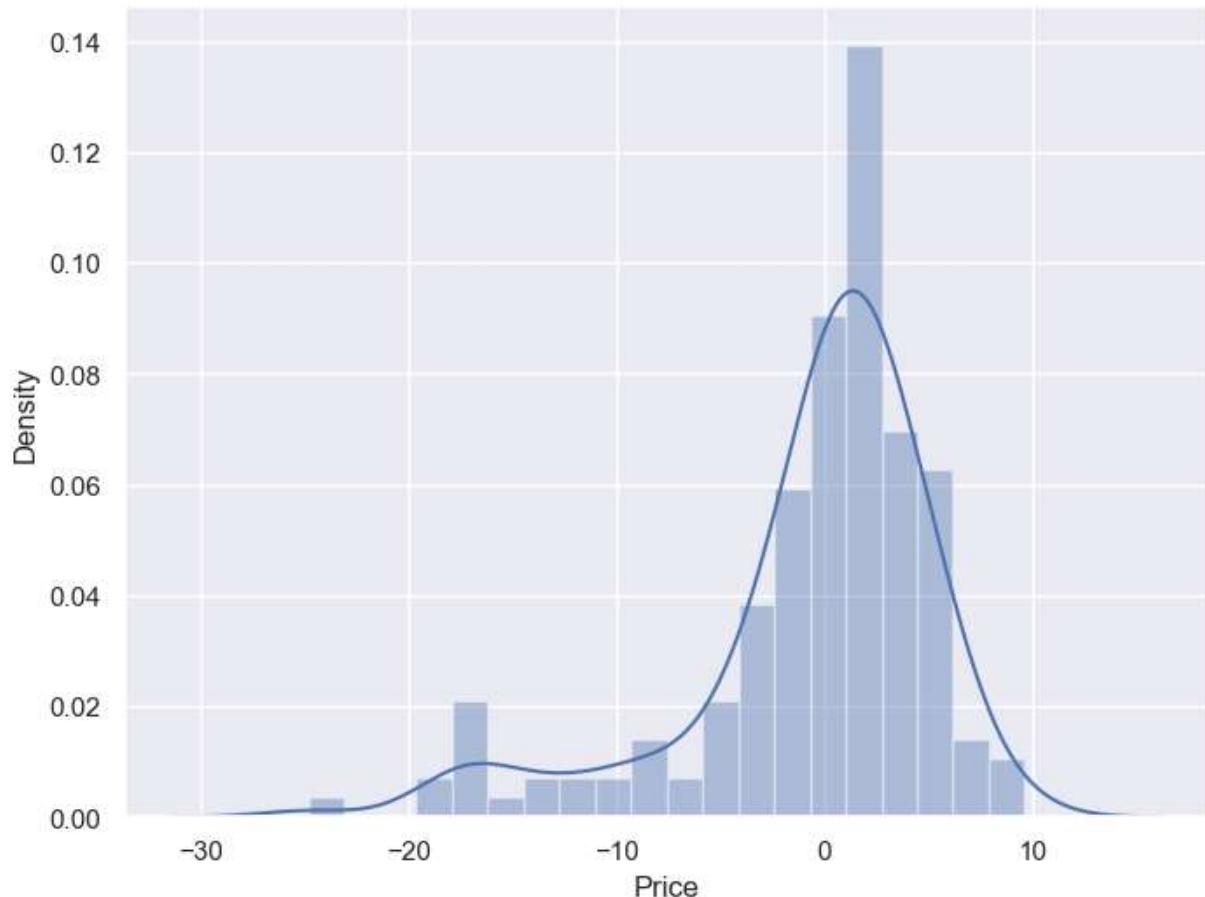
## Adjusted R2-Score Lasso Regression

```
In [298]: 1 1-(1-lasso_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[298]: 0.5901561369378114

```
In [299]: 1 import seaborn as sns  
2 sns.distplot(lasso_pred-y_test)
```

Out[299]: <AxesSubplot:xlabel='Price', ylabel='Density'>



## ElasticNet Regression

```
In [300]: 1 from sklearn.linear_model import ElasticNet
```

```
In [301]: 1 elastic_net_reg=ElasticNet(l1_ratio=0.5)
```

```
In [302]: 1 elastic_net_reg.fit(X_train,y_train)
```

```
Out[302]: ElasticNet()
```

```
In [303]: 1 parameters={'alpha':[1,2,5,10,20,30]}
2 elastic_net_cv=GridSearchCV(lasso,parameters,scoring='neg_mean_squared_error'
3 elastic_net_cv.fit(X_train,y_train)
```

```
Out[303]: GridSearchCV(cv=5, estimator=Lasso(),
param_grid={'alpha': [1, 2, 5, 10, 20, 30]},
scoring='neg_mean_squared_error')
```

```
In [304]: 1 print(elastic_net_cv.best_params_)

{'alpha': 1}
```

```
In [305]: 1 print(elastic_net_cv.best_score_)

-26.640086954213285
```

```
In [306]: 1 elastic_net_pred=elastic_net_cv.predict(X_test)
```

## Performance metrics of ElasticNet Regression

```
In [307]: 1 from sklearn.metrics import mean_squared_error
2 from sklearn.metrics import mean_absolute_error
3 print(mean_squared_error(y_test,elastic_net_pred))
4 print(mean_absolute_error(y_test,elastic_net_pred))
5 print(np.sqrt(mean_squared_error(y_test,elastic_net_pred)))
```

```
36.11332777156727
4.067062767559683
6.0094365602415065
```

## R2-Score of ElasticNet Regression

```
In [308]: 1 from sklearn.metrics import r2_score
2 elastic_net_score=r2_score(y_test,elastic_net_pred)
3 print(score)
```

```
0.7165219393967557
```

## Adjusted R2-score of ElasticNet Regression

```
In [309]: 1 -(1-elastic_net_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

```
Out[309]: 0.5901561369378114
```

```
In [ ]: 1
```