

Linear Regression Practical Implementation

```
In [1]: 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  
7 import warnings  
8 warnings.filterwarnings("ignore")
```

```
In [2]: 1 ## Loading Dataset from sklearn Library  
2 from sklearn.datasets import load_boston  
3
```

```
In [3]: 1 boston = load_boston()
```

In [5]: 1 print(boston)

```
{'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, 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]), 'feature_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  :Number of Instances: 506 \n  :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.\n  :Attribute Information (in order):\n    - CRIM      per capita crime rate by town\n    - ZN        proportion of residential land 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 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 property-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      Median value of owner-occupied homes in $1000's\n  :Missing Attribute Values: None\n  :Creator: Harrison, D. and Rubinfeld, D.L.\nThis is a copy of UCI ML housing dataset.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/housing/\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. 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 data has been used in many machine learning papers that address regression\nproblems.\n.. topic:: References\n  - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.\n  - 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.\n",
  'filename': 'C:\\\\Users\\\\Lenovo\\\\anaconda3\\\\lib\\\\site-packages\\\\sklearn\\\\datasets\\\\data\\\\boston_house_prices.csv'}
```

```
In [79]: 1 boston.keys()
```

```
Out[79]: dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

```
In [151]: 1 print(boston.filename)
```

```
C:\\Users\\Lenovo\\anaconda3\\lib\\site-packages\\sklearn\\datasets\\data\\boston_house_prices.csv
```

In [4]:

```
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 [6]:

```
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]
```

```
In [8]: 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 [9]: 1 `print(boston.feature_names)`

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

In []: 1 `## Let prepare the Dataframe`

In [10]: 1 `dataset = pd.DataFrame(boston.data,columns = boston.feature_names)`

In [11]: 1 `dataset`

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
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.
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.
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.
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.
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.
...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.

506 rows × 13 columns

In [12]: 1 `dataset['Price'] = boston.target`

In [13]: 1 dataset

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
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.
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.
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.
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.
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.
...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.

506 rows × 14 columns

In [14]: 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
```

```
In [15]: 1 dataset.describe()
```

```
Out[15]:
```

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

```
In [16]: 1 ## For Checking Missing Value  
2  
3 dataset.isnull().sum()
```

```
Out[16]: 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
```

In [17]:

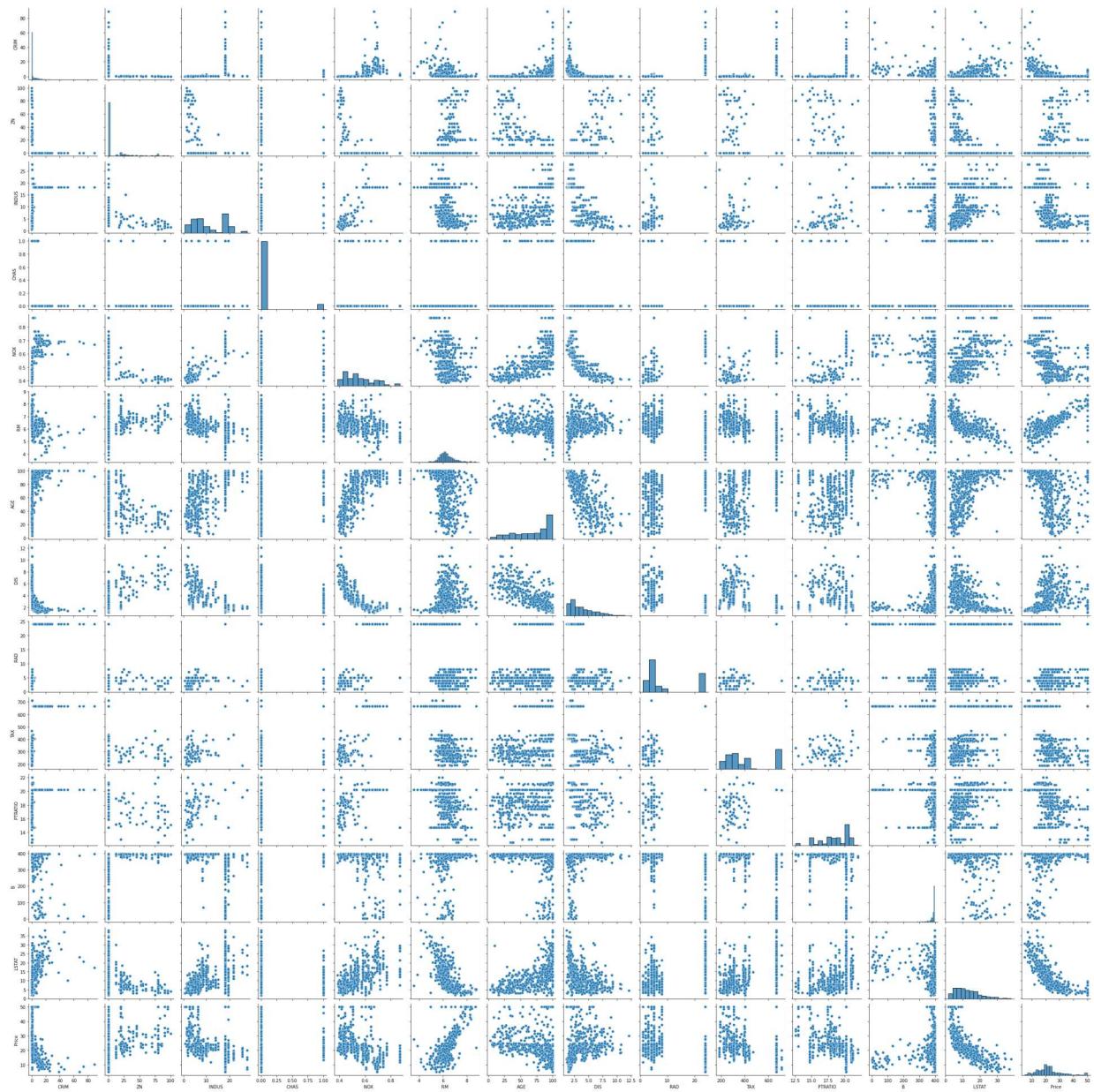
```
1 ## EDA  
2  
3 dataset.corr() ## for Analysis the dataset ,we use Correlation.
```

Out[17]:

	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.

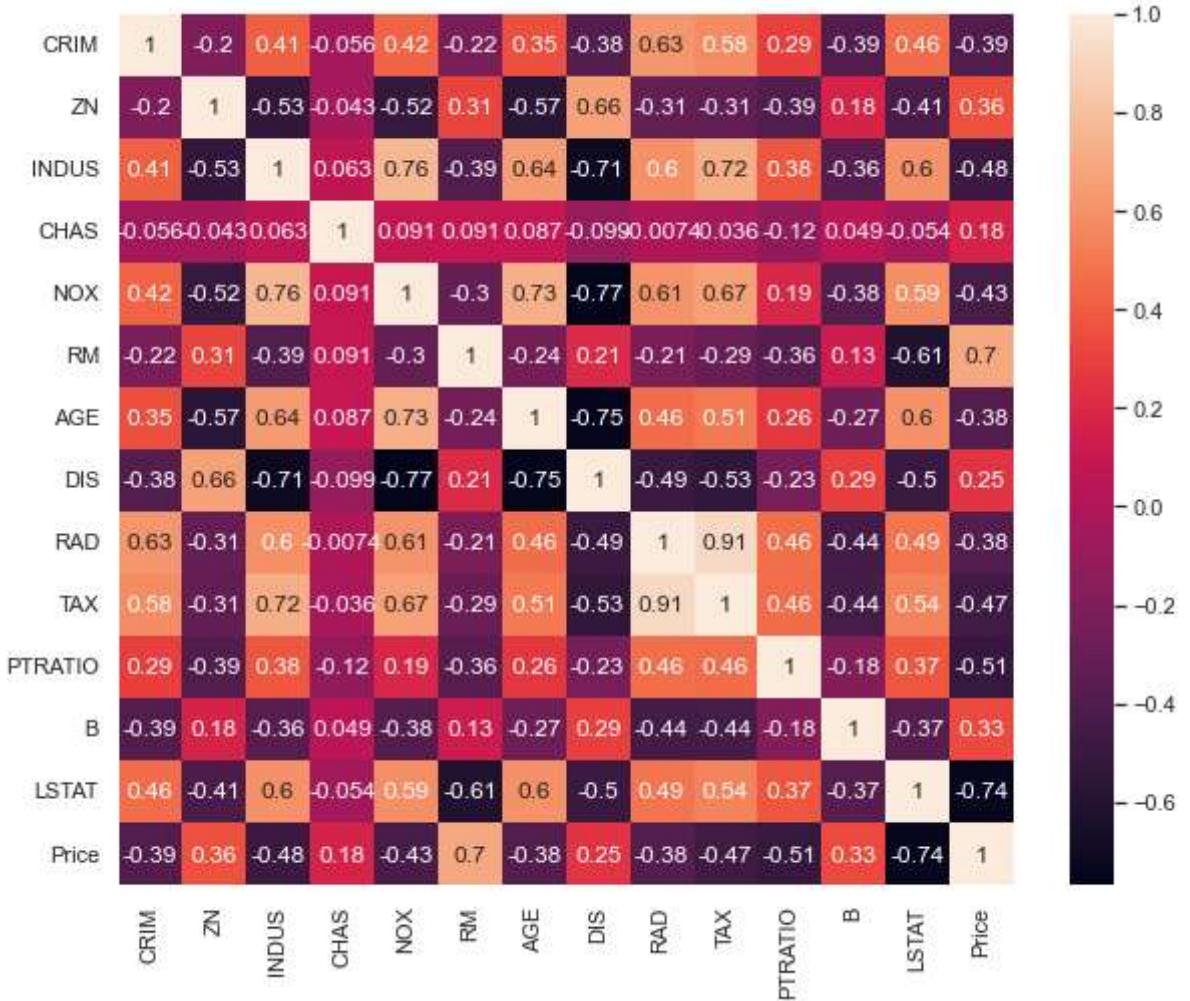
```
In [18]: 1 import seaborn as sns  
2 sns.pairplot(dataset)
```

Out[18]: <seaborn.axisgrid.PairGrid at 0x15832a45160>



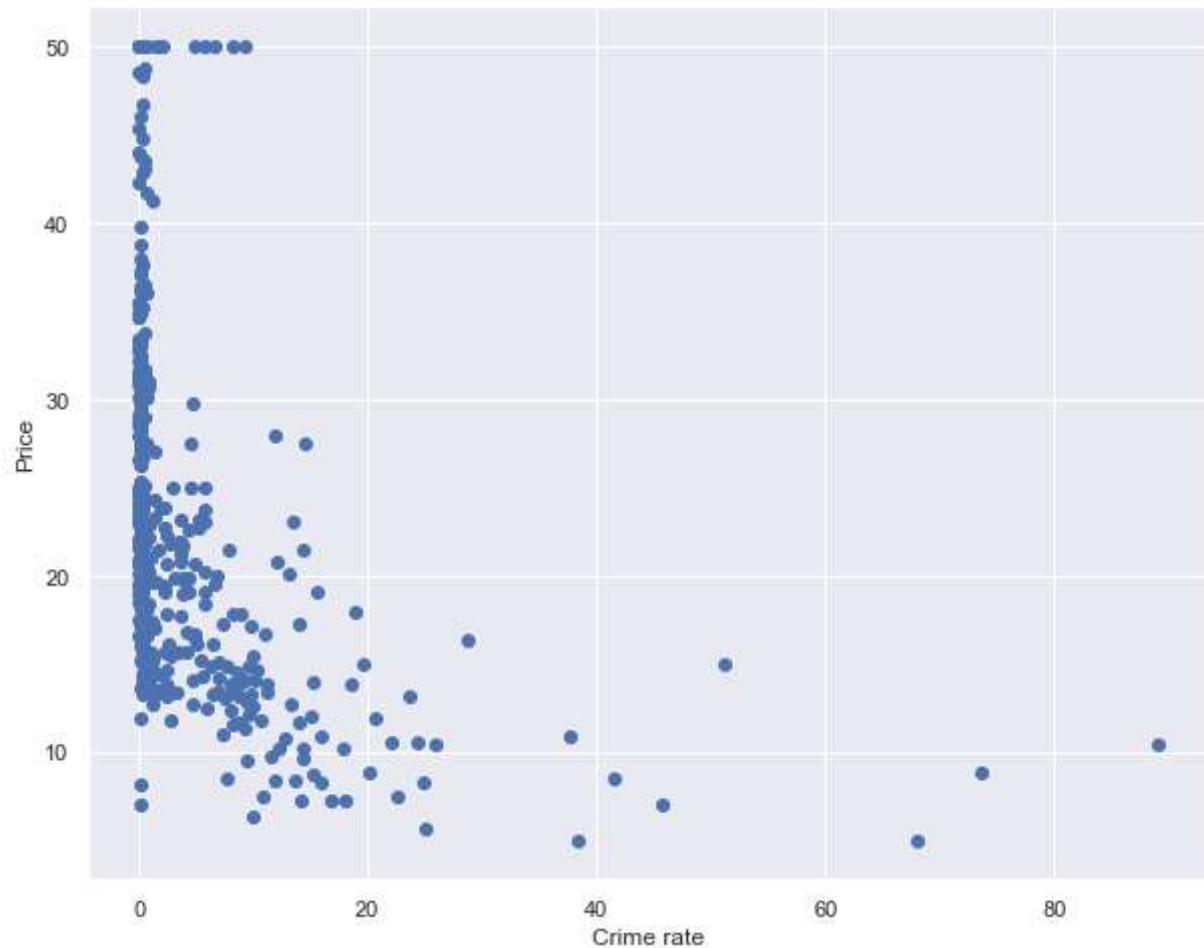
```
In [80]: 1 sns.set(rc = {'figure.figsize' : (10,8)})  
2  
3 sns.heatmap(dataset.corr(), annot = True)
```

Out[80]: <AxesSubplot:>



```
In [81]: 1 plt.scatter(dataset['CRIM'],dataset['Price'])  
2 plt.xlabel("Crime rate")  
3 plt.ylabel("Price")
```

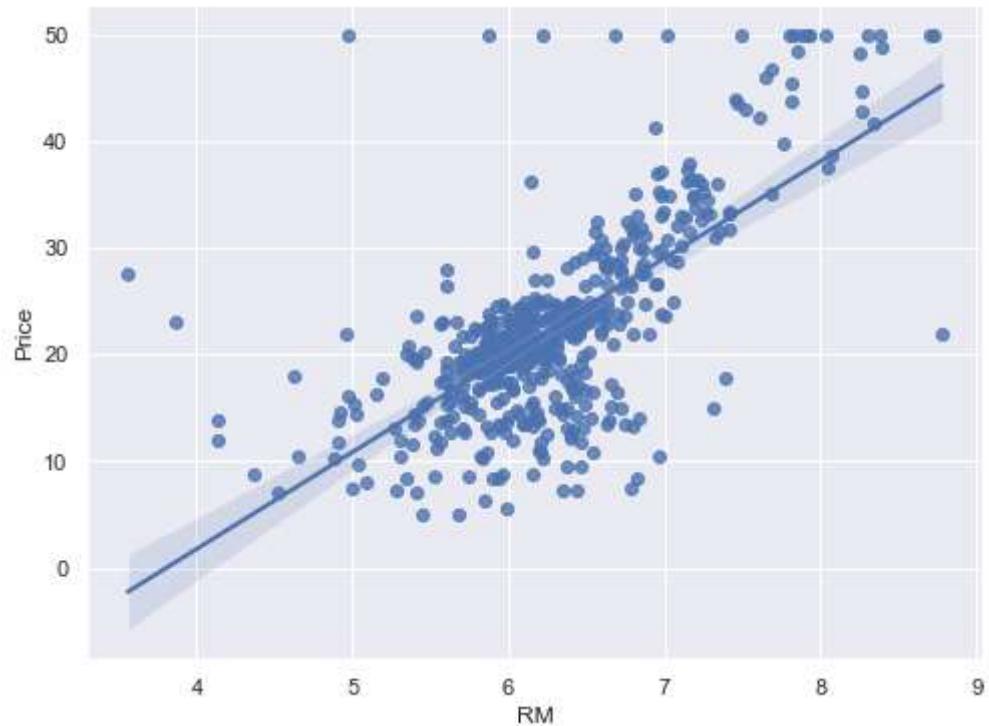
Out[81]: Text(0, 0.5, 'Price')



In [82]:

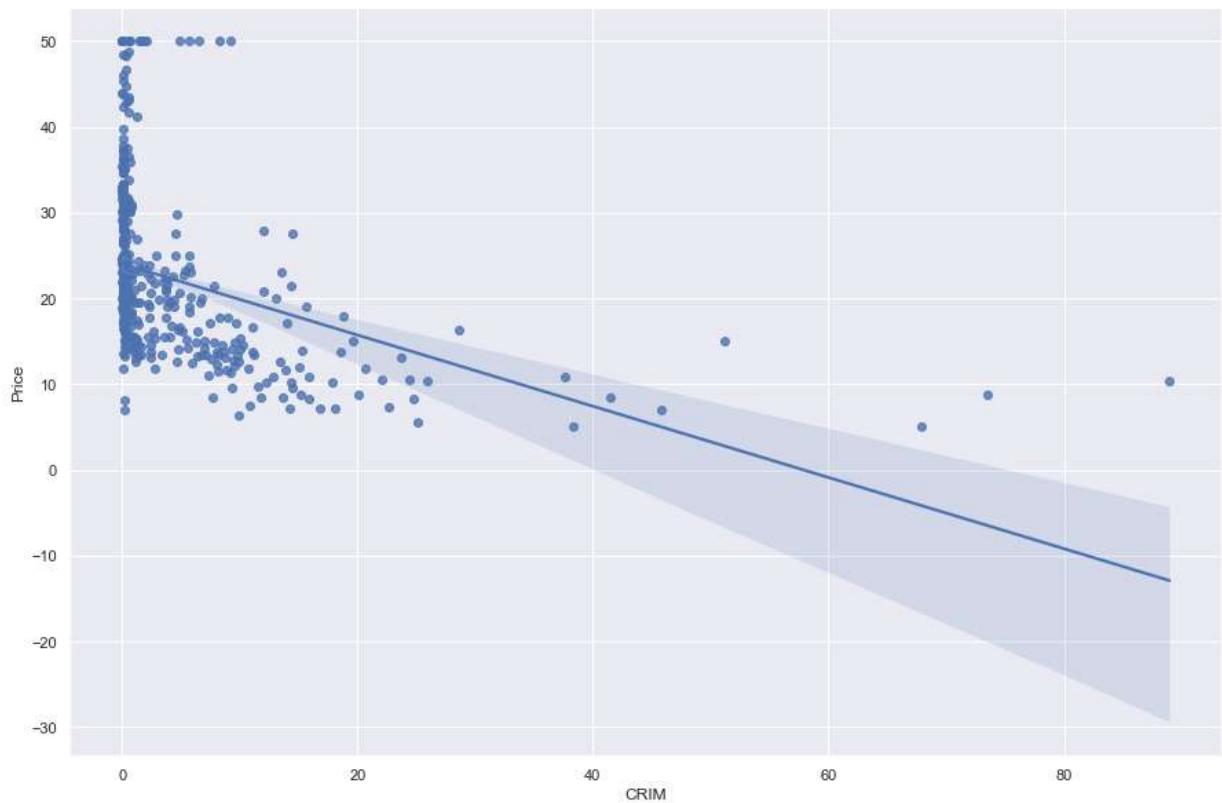
```
1 sns.set(rc={'figure.figsize':(8,6)})  
2 sns.regplot(x = "RM", y = "Price", data = dataset)
```

Out[82]: <AxesSubplot:xlabel='RM', ylabel='Price'>



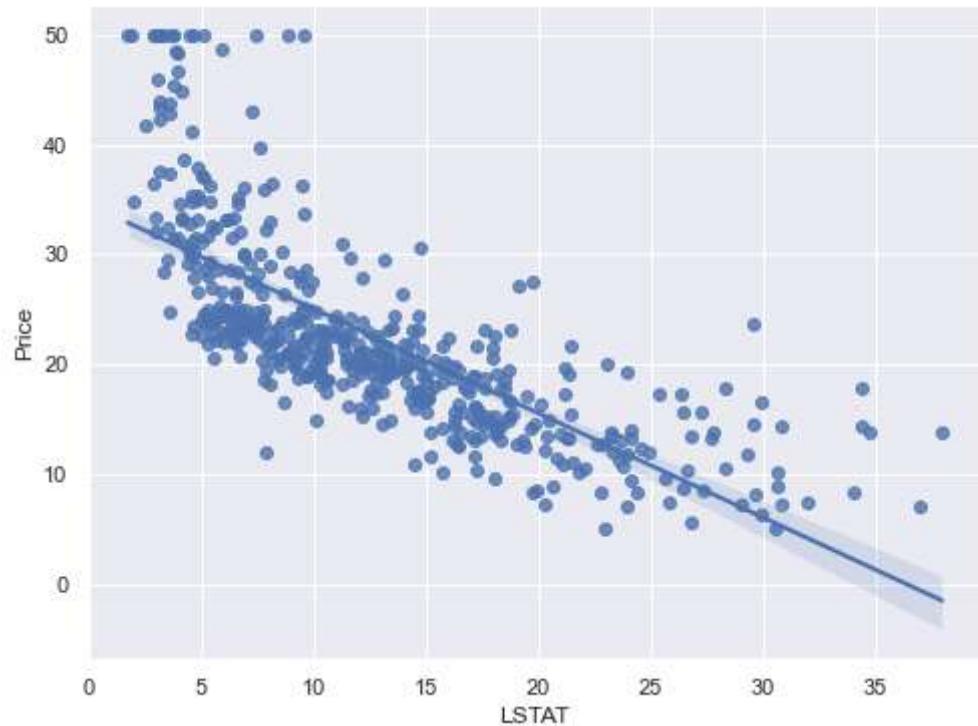
```
In [22]: 1 sns.regplot(x = "CRIM", y = "Price", data = dataset)
```

```
Out[22]: <AxesSubplot:xlabel='CRIM', ylabel='Price'>
```



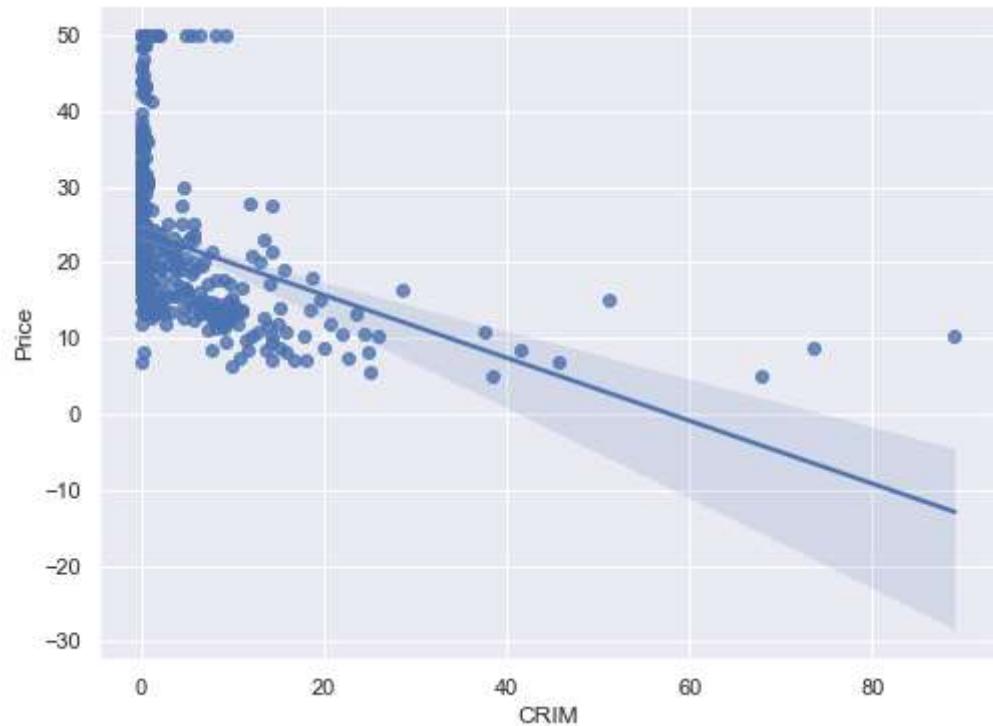
```
In [83]: 1 sns.regplot(x="LSTAT",y="Price",data=dataset)
```

```
Out[83]: <AxesSubplot:xlabel='LSTAT', ylabel='Price'>
```



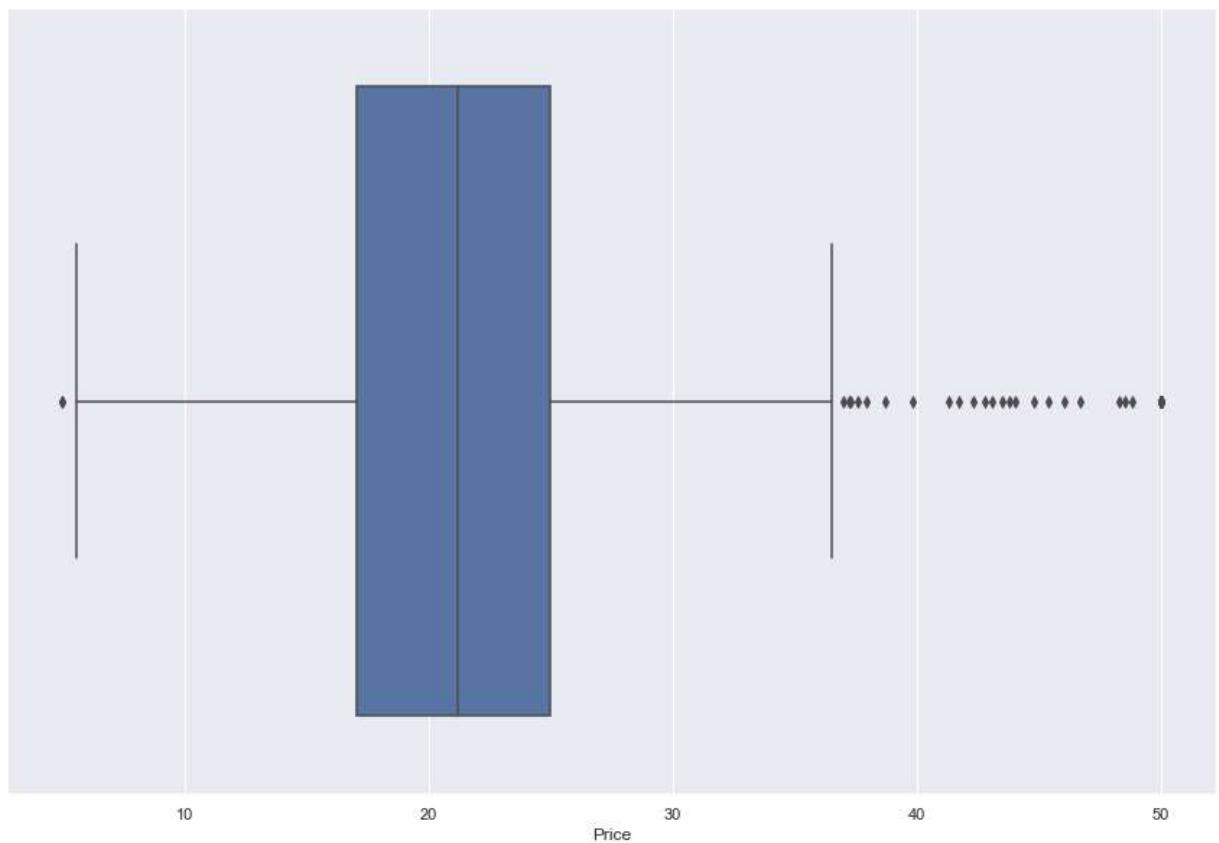
```
In [84]: 1 sns.regplot(x="CRIM",y="Price",data=dataset)
```

```
Out[84]: <AxesSubplot:xlabel='CRIM', ylabel='Price'>
```



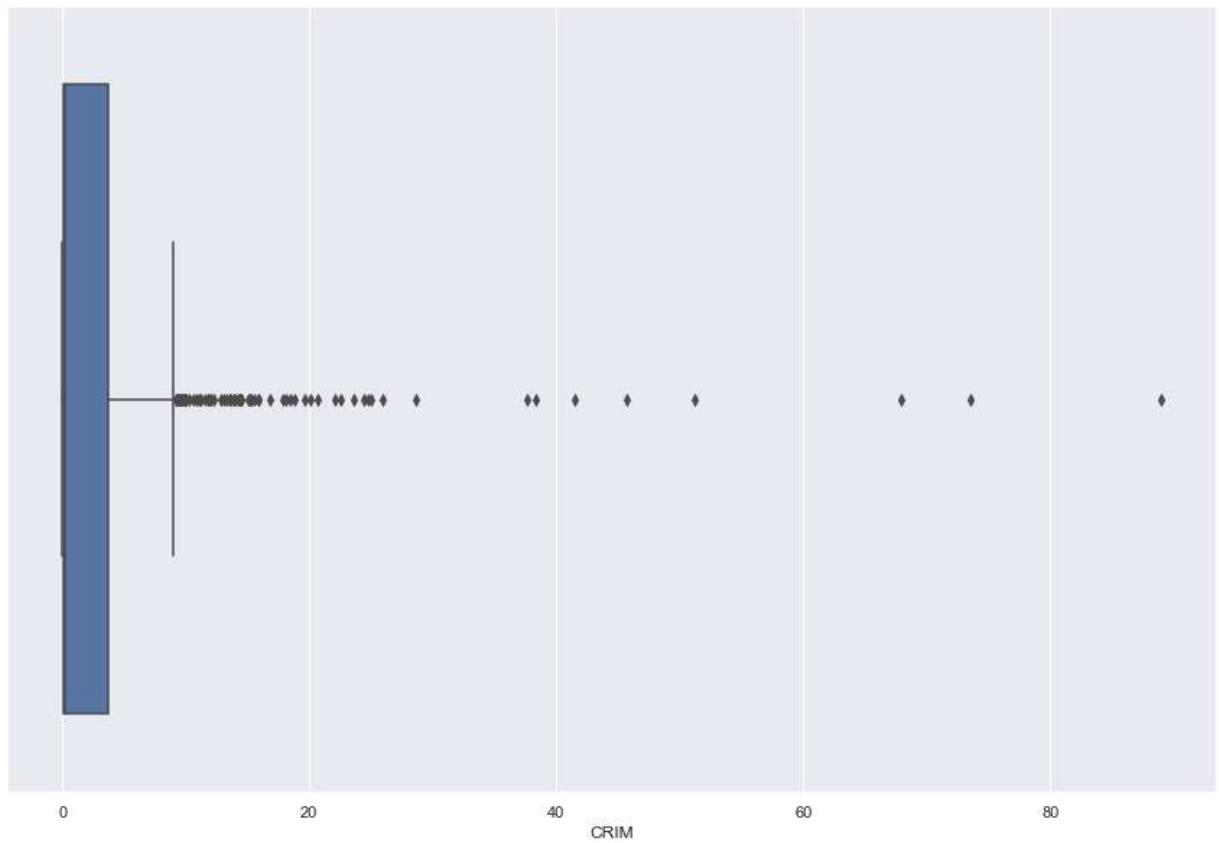
```
In [23]: 1 sns.boxplot(dataset['Price'])
```

```
Out[23]: <AxesSubplot:xlabel='Price'>
```



```
In [24]: 1 sns.boxplot(dataset['CRIM'])
```

```
Out[24]: <AxesSubplot:xlabel='CRIM'>
```



```
In [25]: 1 dataset.head(3)
```

```
Out[25]:    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
```

```
In [26]: 1 ## Independent and Dependent Feature  
2  
3 x = dataset.iloc[:, :-1]  
4 y = dataset.iloc[:, -1]
```

```
In [27]: 1 x
```

Out[27]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST.
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.
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.
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.
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.
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.
...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.

506 rows × 13 columns

```
In [28]: 1 y
```

Out[28]:

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

```
In [29]: 1 from sklearn.model_selection import train_test_split
```

```
In [30]: 1 X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.33, ra
```

```
In [31]: 1 X_train
```

```
Out[31]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS'
478	10.23300	0.0	18.10	0.0	0.614	6.185	96.7	2.1705	24.0	666.0	20.2	379.70	18
26	0.67191	0.0	8.14	0.0	0.538	5.813	90.3	4.6820	4.0	307.0	21.0	376.88	14
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19
492	0.11132	0.0	27.74	0.0	0.609	5.983	83.5	2.1099	4.0	711.0	20.1	396.90	13
108	0.12802	0.0	8.56	0.0	0.520	6.474	97.1	2.4329	5.0	384.0	20.9	395.24	12
...
106	0.17120	0.0	8.56	0.0	0.520	5.836	91.9	2.2110	5.0	384.0	20.9	395.67	18
270	0.29916	20.0	6.96	0.0	0.464	5.856	42.1	4.4290	3.0	223.0	18.6	388.65	13
348	0.01501	80.0	2.01	0.0	0.435	6.635	29.7	8.3440	4.0	280.0	17.0	390.94	5
435	11.16040	0.0	18.10	0.0	0.740	6.629	94.6	2.1247	24.0	666.0	20.2	109.85	23
102	0.22876	0.0	8.56	0.0	0.520	6.405	85.4	2.7147	5.0	384.0	20.9	70.80	10

339 rows × 13 columns

```
In [33]: 1 X_train.shape
```

```
Out[33]: (339, 13)
```

```
In [34]: 1 Y_train
```

```
Out[34]:
```

478	14.6
26	16.6
7	27.1
492	20.1
108	19.8
...	...
106	19.5
270	21.1
348	24.5
435	13.4
102	18.6

Name: Price, Length: 339, dtype: float64

```
In [35]: 1 Y_train.shape
```

```
Out[35]: (339,)
```

```
In [36]: 1 X_test.shape
```

```
Out[36]: (167, 13)
```

```
In [37]: 1 Y_test.shape
```

```
Out[37]: (167,)
```

Standardize or Feature Scalling the Dataset

```
In [38]: 1 ## Standardize or Feature Scalling the Dataset
2
3 from sklearn.preprocessing import StandardScaler
4 scaler = StandardScaler()
5
```

```
In [39]: 1 scaler ## Scaler is an Object , an Standard Scaler Objects
```

```
Out[39]: StandardScaler()
```

```
In [40]: 1 X_train = scaler.fit_transform(X_train) ## Fit_transport means Applyi
```

```
In [41]: 1 X_test = scaler.transform(X_test)
```

```
In [42]: 1 X_train
```

```
Out[42]: array([[ 0.89624872, -0.51060139,  0.98278223, ...,
   0.24040357,  0.77155612],
 [-0.34895881, -0.51060139, -0.44867555, ...,
  0.20852839,  0.32248963],
 [-0.41764058,  0.03413008, -0.48748013, ...,
  0.43481957,  0.92775316],
 ...,
 [-0.43451148,  2.97567999, -1.32968321, ...,
  0.36745216, -0.90756208],
 [ 1.01703049, -0.51060139,  0.98278223, ...,
 -2.80977992,  1.50233514],
 [-0.40667333, -0.51060139, -0.38831288, ...,
 -3.25117205, -0.26046005]])
```

```
In [43]: 1 X_test
```

```
Out[43]: array([[ -0.42451319, -0.51060139, -1.03649306, ...,
   -0.74102621,
    0.41899501, -0.48220406],
 [-0.42911576,  1.2325393 , -0.6973123 , ...,
  0.43481957, -1.25063772],
 [-0.42269508, -0.51060139,  2.36824941, ...,
  0.35807046,  0.77713459],
 ...,
 [-0.33727525,  0.36096896, -1.04799071, ...,
  0.38395492, -0.28556314],
 [-0.30591027, -0.51060139, -0.44867555, ...,
  0.2463943 , -0.07218683],
 [-0.36872487,  0.36096896, -1.04799071, ...,
  0.32133488, -0.91871901]])
```

Model Training

Linear Regression Model

```
In [44]: 1 ## Linear Regression  
2 from sklearn.linear_model import LinearRegression
```

```
In [45]: 1 regression = LinearRegression()
```

```
In [46]: 1 regression
```

```
Out[46]: LinearRegression()
```

```
In [47]: 1 regression.fit(X_train, Y_train)
```

```
Out[47]: LinearRegression()
```

print the Coefficients and the intercept

```
In [48]: 1 ## print the Coefficients  
2 print(regression.coef_)
```

```
[ -0.98858032  0.86793276  0.40502822  0.86183791 -1.90009974  2.80813518  
 -0.35866856 -3.04553498  2.03276074 -1.36400909 -2.0825356   1.04125684  
 -3.92628626]
```

```
In [49]: 1 ## print the intercept  
2 print(regression.intercept_)
```

```
22.970796460176988
```

```
In [50]: 1 ## Prediction for the Test data  
2 reg_pred = regression.predict(X_test)
```

```
In [53]: 1 reg_pred  
2
```

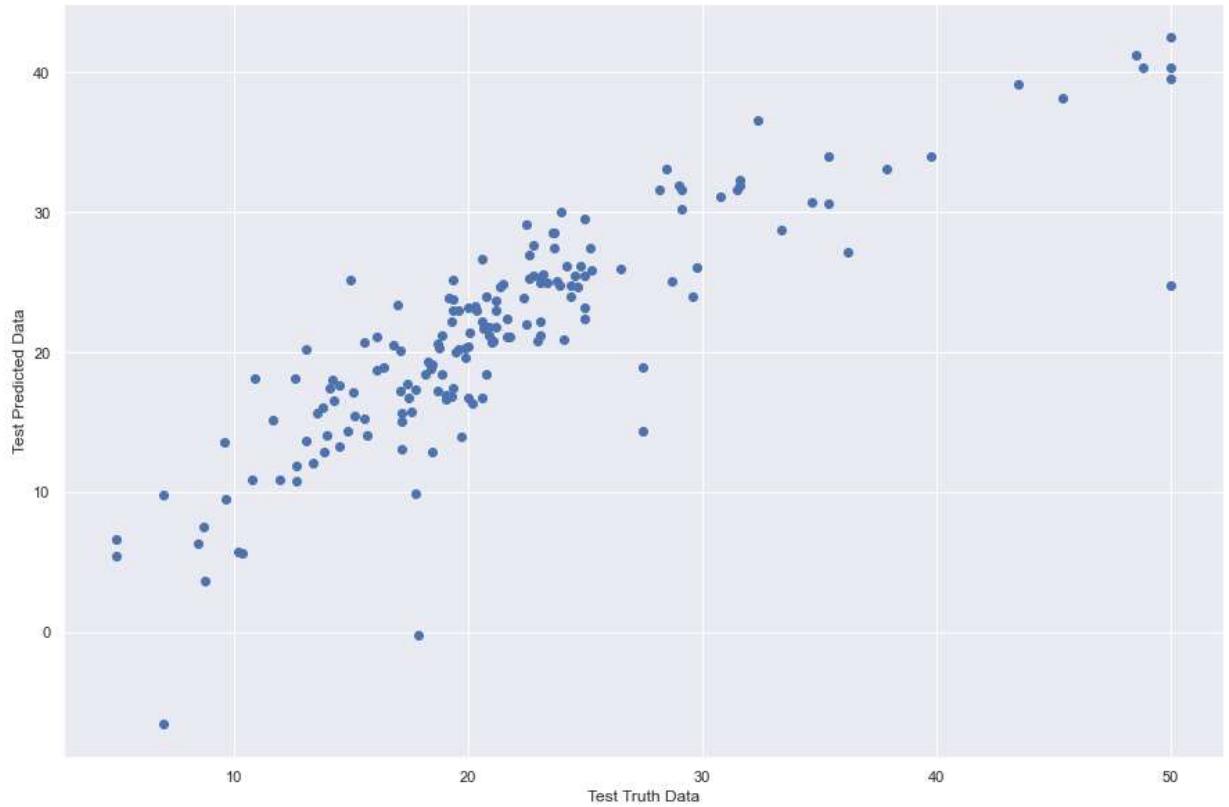
```
Out[53]: array([28.53469469, 36.6187006 , 15.63751079, 25.5014496 , 18.7096734 ,  
 23.16471591, 17.31011035, 14.07736367, 23.01064388, 20.54223482,  
 24.91632351, 18.41098052, -6.52079687, 21.83372604, 19.14903064,  
 26.0587322 , 20.30232625, 5.74943567, 40.33137811, 17.45791446,  
 27.47486665, 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])
```

Assumption of Linear Regression.

```
In [54]: 1 ## Assumption of Linear Regression.  
2  
3 import seaborn as sns  
4 import matplotlib.pyplot as plt
```

```
In [56]: 1 ## Relationship Between Real Data & Predicted Data
          2 plt.scatter(Y_test,reg_pred)           ## IF we get Linear Manner , it is goo
          3 plt.xlabel("Test Truth Data")        ## When we take Test data & Predictio
          4 plt.ylabel("Test Predicted Data")      5
```

Out[56]: Text(0, 0.5, 'Test Predicted Data')



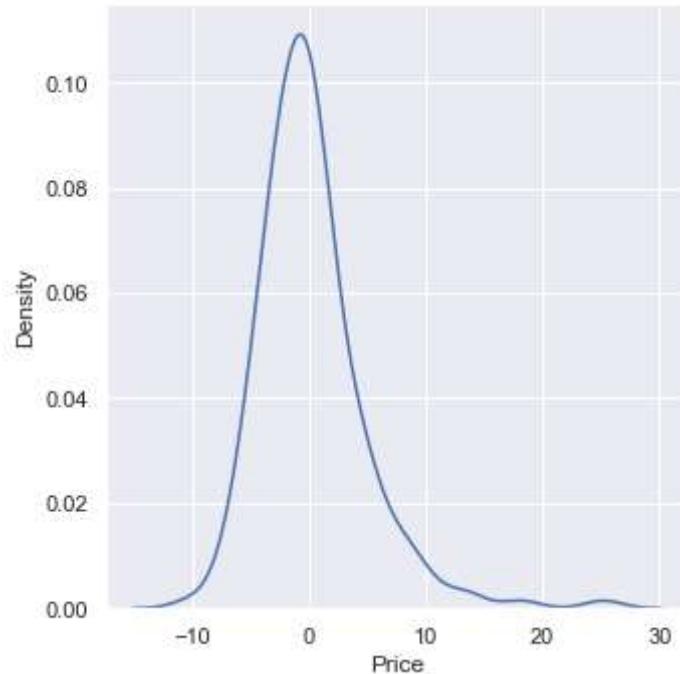
```
In [72]: 1 ## Calculating Residual
          2 residuals = Y_test - reg_pred
          3
```

```
In [73]: 1 residuals  
2
```

```
Out[73]: 173 -4.934695  
274 -4.218701  
491 -2.037511  
72 -2.701450  
452 -2.609673  
...  
110 0.642557  
321 -1.917346  
265 -4.854619  
29 0.297942  
262 8.417851  
Name: Price, Length: 167, dtype: float64
```

```
In [74]: 1 ## Distribution of residual are Approximately Normal Distribution  
2 sns.displot(residuals, kind ="kde")  
3
```

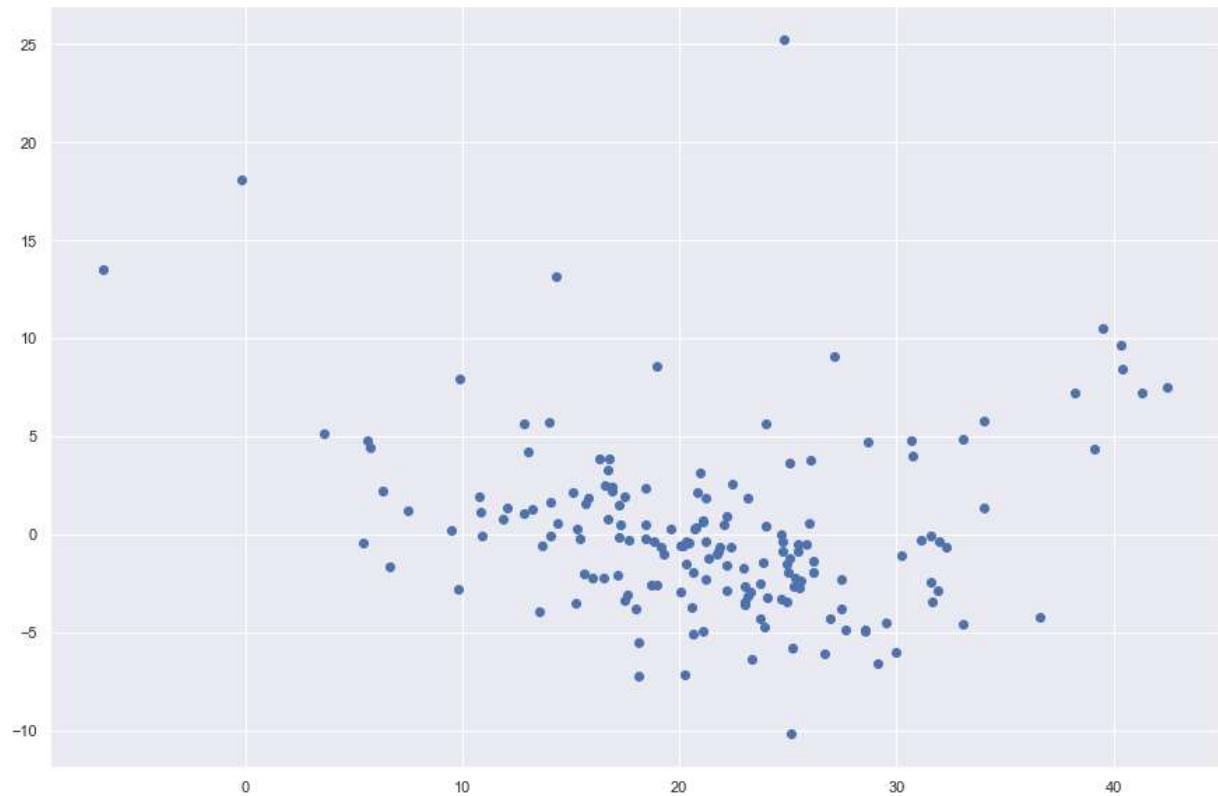
```
Out[74]: <seaborn.axisgrid.FacetGrid at 0x1583f53a130>
```



In [75]:

```
1 ## Scatter Plot with predictions and residual  
2 ### Uniform Distributions  
3 plt.scatter(reg_pred, residuals)      ## Uniform Distributions :- Model is in  
4
```

Out[75]: <matplotlib.collections.PathCollection at 0x15843ee3b50>



Performance Metrics

```
In [76]: 1 ## Performance Metrics  
2  
3 from sklearn.metrics import mean_squared_error  
4 from sklearn.metrics import mean_absolute_error  
5  
6 print(mean_squared_error(Y_test, reg_pred))  
7 print(mean_absolute_error(Y_test, reg_pred))  
8 print(np.sqrt(mean_squared_error(Y_test, reg_pred)))
```

```
20.724023437339753  
3.148255754816832  
4.552364598463062
```

R Squared and Adjusted R-Squared

```
In [77]: 1 ## R Squared  
2 from sklearn.metrics import r2_score  
3 score=r2_score(Y_test,reg_pred)  
4 print(score)  
5
```

```
0.7261570836552476
```

```
In [78]: 1 ## Adjusted R square  
2 #display adjusted R-squared  
3 1 - (1-score)*(len(Y_test)-1)/(len(Y_test)-X_test.shape[1]-1)  
4
```

```
Out[78]: 0.7028893848808568
```

Ridge Regression Model

```
In [93]: 1 ## Ridge Regression  
2 from sklearn.linear_model import Ridge
```

```
In [94]: 1 ridge=Ridge()
```

```
In [95]: 1 ridge
```

```
Out[95]: Ridge()
```

```
In [96]: 1 ridge.fit(X_train,Y_train)
```

```
Out[96]: Ridge()
```

print the coefficients and the intercept

```
In [98]: 1 ## print the coefficients  
2 print(ridge.coef_)  
3  
[-0.97541551  0.84608896  0.37564928  0.86738391 -1.86077739  2.81535042  
-0.36108635 -3.00177053  1.95063015 -1.29462251 -2.06972563  1.03867858  
-3.91121554]
```

```
In [99]: 1 ## print the intercept  
2 print(ridge.intercept_)  
3
```

```
22.970796460176988
```

```
In [101]: 1 ## Prediction for the Test data  
2 ridge_pred = ridge.predict(X_test)
```

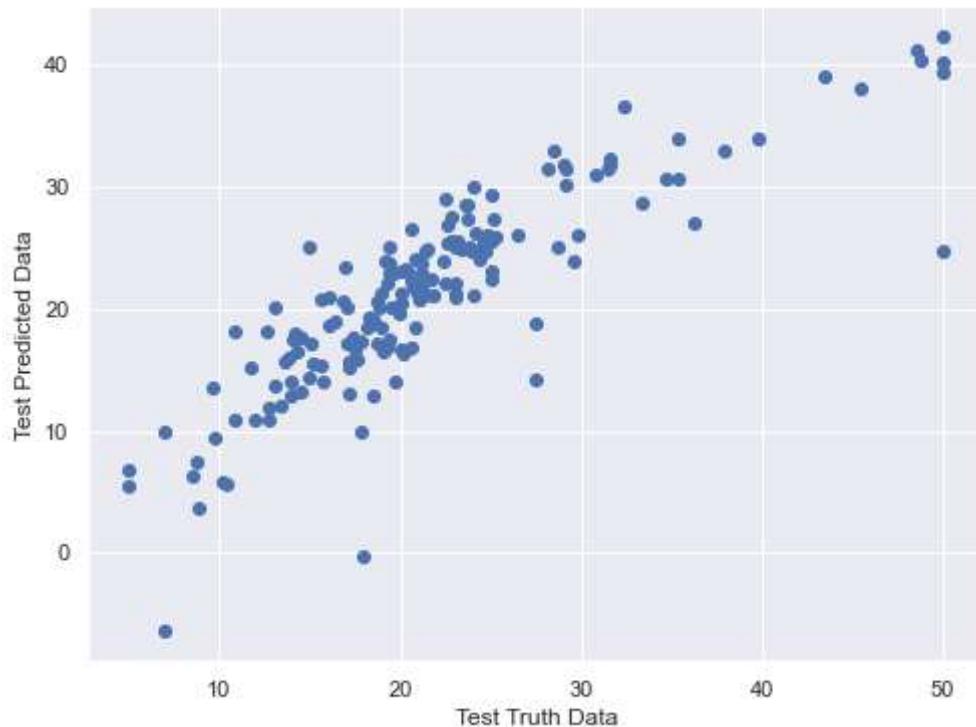
```
In [102]: 1 ridge_pred
```

```
Out[102]: array([28.50742044, 36.56623172, 15.74385228, 25.49107605, 18.70449805,  
23.16555674, 17.3353633 , 14.11553585, 22.94790581, 20.56605905,  
24.89034838, 18.45231813, -6.44592912, 21.85588536, 19.15767075,  
26.02049072, 20.20418625, 5.75334034, 40.26109346, 17.45757103,  
27.43625 , 30.15008183, 10.86542778, 23.8922577 , 17.98002216,  
15.97783016, 23.27084129, 14.39145132, 22.39046559, 19.33913789,  
22.14761798, 25.0633962 , 25.12323651, 18.44886131, 16.56498494,  
17.51980787, 30.71926279, 20.06321173, 24.0051703 , 24.93961773,  
14.01804472, 31.55716108, 42.36411402, 17.74514996, 26.92815913,  
17.13671991, 13.73088718, 26.14522758, 20.23784138, 30.01688815,  
21.23834632, 33.98097943, 15.46847103, 25.98238451, 39.0872846 ,  
22.93270116, 18.79729577, 32.98900997, 24.75121546, 12.88534635,  
22.44620063, 30.6177806 , 31.5496446 , 16.3728183 , 21.07398052,  
16.71642532, 20.21399742, 26.13275151, 31.06390949, 11.90776462,  
20.45341375, 27.424314 , 10.91082555, 16.86277875, 24.02466788,  
5.49260171, 21.36935091, 41.1562018 , 18.12590293, 9.9134628 ,  
21.27916165, 12.98165074, 21.84047835, 9.47562845, 22.9796062 ,  
31.89358365, 18.96118579, 25.49195924, 29.40639573, 20.09806745,  
25.55440575, 5.61779356, 20.21319857, 15.20694421, 14.27823198,  
20.87913507, 24.73423572, -0.20608345, 13.60245079, 15.61960778,  
22.04117361, 24.70915804, 10.84401433, 19.6404862 , 23.76546818,  
12.0612365 , 18.43781737, 25.45763751, 20.79157763, 24.72828941,  
7.53954069, 18.89986796, 21.74415493, 27.09734457, 31.89508098,  
15.20451115, 33.98082344, 12.90997237, 21.08195942, 28.5634719 ,  
15.81062962, 24.79975721, 3.74945983, 23.92684052, 25.82363831,  
23.03894509, 25.37633896, 33.01041733, 20.74845835, 38.12699454,  
13.96268847, 25.34137677, 17.65545489, 20.63383355, 9.92384983,  
21.01822914, 22.20503183, 32.23576441, 31.55073046, 15.34316201,  
16.75621922, 29.10097721, 25.14507288, 16.89567228, 6.37702315,  
26.61598967, 23.43020988, 17.25262089, 13.2787411 , 39.46160064,  
16.5377085 , 18.15230812, 25.09351869, 23.72728145, 22.19137133,  
21.26128221, 16.8779622 , 23.15293077, 28.72634311, 6.73699279,  
23.9593386 , 17.21668437, 21.08725762, 25.0201436 , 27.60453343,  
20.74915073, 40.35440842])
```

Assumptions of Ridge Regression:-

```
In [103]: 1 ## Relationship Between Real Data & Predicted Data.  
2 plt.scatter(Y_test,ridge_pred)      ## IF we get Linear Manner , it is good Li  
3 plt.xlabel("Test Truth Data")  
4 plt.ylabel("Test Predicted Data")  
5
```

Out[103]: Text(0, 0.5, 'Test Predicted Data')



```
In [104]: 1 ## Calculating residuals:-  
2 residuals=Y_test - ridge_pred  
3
```

```
In [105]: 1 residuals  
2
```

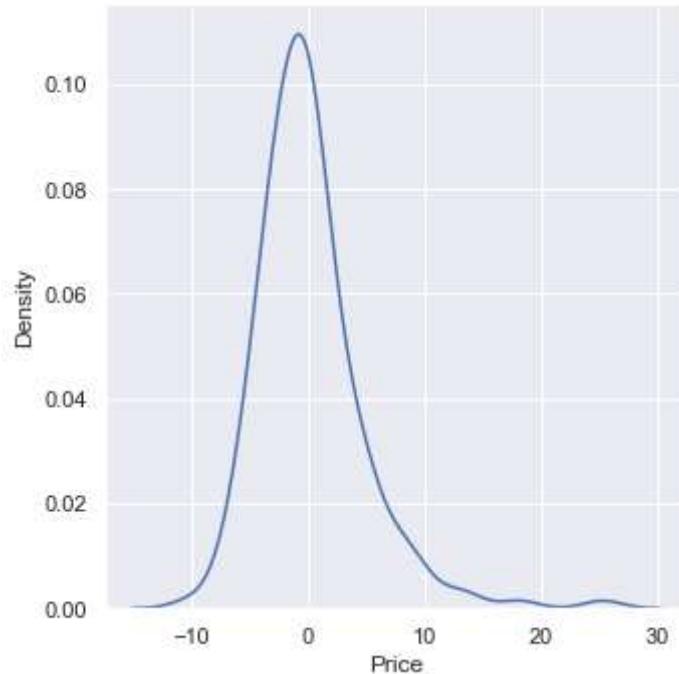
Out[105]:

173	-4.907420
274	-4.166232
491	-2.143852
72	-2.691076
452	-2.604498
	...
110	0.612742
321	-1.920144
265	-4.804533
29	0.250849
262	8.445592

Name: Price, Length: 167, dtype: float64

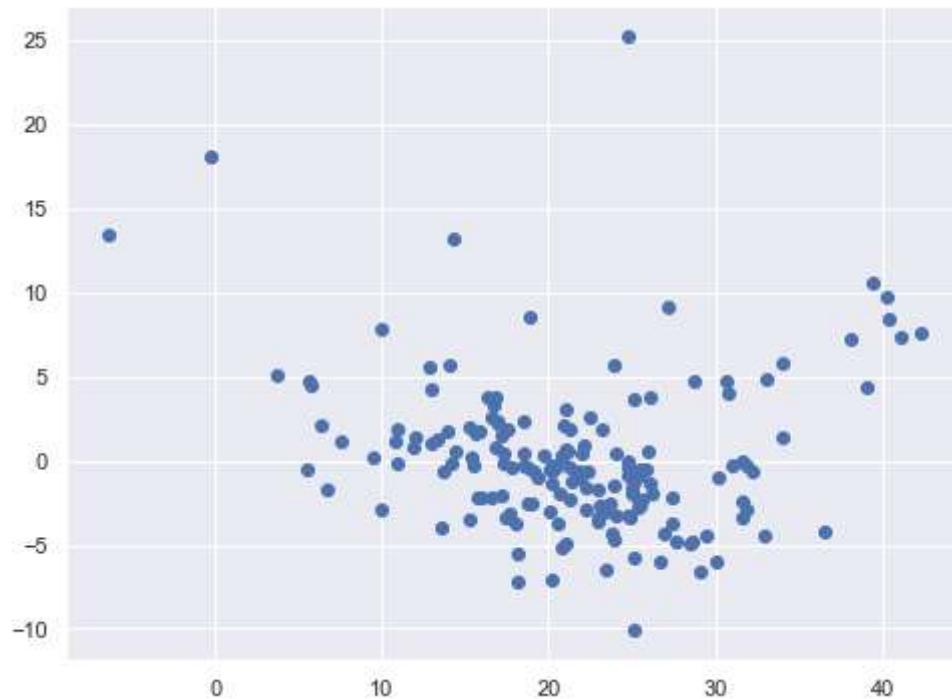
```
In [106]: 1 ## Distribution of residual are Approximately Normal Distribution
           2 sns.displot(residuals,kind="kde")
           3
```

Out[106]: <seaborn.axisgrid.FacetGrid at 0x1584436a730>



```
In [107]:  
1 ## Scatter plot with predictions and residual  
2 ##uniform distribution  
3 plt.scatter(ridge_pred,residuals)  
4
```

```
Out[107]: <matplotlib.collections.PathCollection at 0x15845d207c0>
```



Performance Metrics

```
In [109]:  
1 ## Performance Metrics  
2 from sklearn.metrics import mean_squared_error  
3 from sklearn.metrics import mean_absolute_error  
4 print(mean_squared_error(Y_test,ridge_pred))  
5 print(mean_absolute_error(Y_test,ridge_pred))  
6 print(np.sqrt(mean_squared_error(Y_test,ridge_pred)))  
7
```

```
20.752416320800283
```

```
3.146011462661694
```

```
4.555482007515811
```

R Squared and Adjusted R Squared

```
In [110]:  
1 ## R Squared  
2 from sklearn.metrics import r2_score  
3 score=r2_score(Y_test,ridge_pred)  
4 print(score)  
5
```

```
0.7257819060246212
```

```
In [111]: 1 ## Adjusted R square  
2 #display adjusted R-squared  
3 1 - (1-score)*(len(Y_test)-1)/(len(Y_test)-X_test.shape[1]-1)  
4
```

```
Out[111]: 0.7024823294123341
```

Lasso Regression Model

```
In [112]: 1 ## Lasso Regression  
2 from sklearn.linear_model import Lasso
```

```
In [115]: 1 lasso = Lasso()
```

```
In [116]: 1 lasso
```

```
Out[116]: Lasso()
```

```
In [118]: 1 lasso.fit(X_train,Y_train)
```

```
Out[118]: Lasso()
```

print the coefficients and the intercept

```
In [119]: 1 ## print the coefficients  
2 print(lasso.coef_)  
3
```

```
[-0.          0.          -0.          0.27140271 -0.          2.62932147  
-0.          -0.          -0.          -0.          -1.21106809  0.29872625  
-3.81788375]
```

```
In [120]: 1 ## print the Intercept  
2 print(lasso.intercept_)  
3
```

```
22.970796460176988
```

```
In [121]: 1 ## Prediction for the Test data  
2 lasso_pred = lasso.predict(X_test)  
3
```

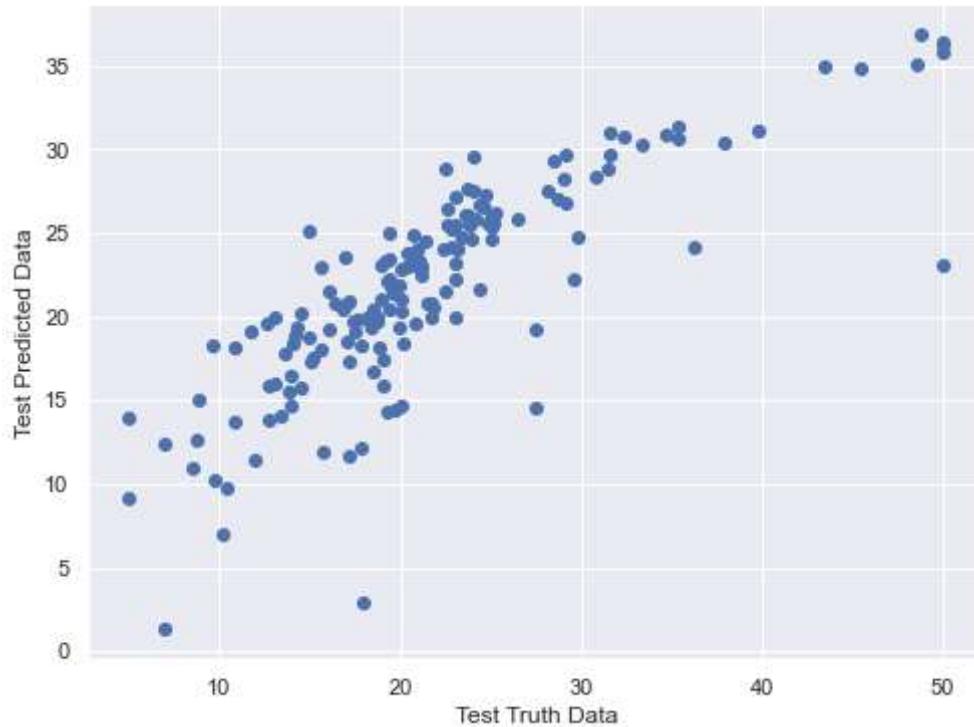
```
In [122]: 1 lasso_pred
```

```
Out[122]: array([26.08015466, 30.7480057 , 17.78164882, 25.25224684, 19.28387274,
 22.81161765, 18.31125182, 14.6359243 , 21.41277818, 20.44276659,
 20.7857368 , 21.00978479, 1.29101416, 22.48591111, 20.4207989 ,
 24.73115299, 18.16643043, 6.95747132, 35.82658816, 18.45664358,
 25.66618031, 26.77096265, 13.79601995, 24.00317031, 18.83677575,
 15.53225538, 22.93567982, 18.81410882, 19.96419904, 19.71394554,
 19.9929271 , 25.48086778, 25.07506471, 19.62299031, 15.87164442,
 20.47826644, 30.90020658, 21.73740698, 21.69357896, 24.78795141,
 14.48946282, 27.49872616, 36.28097645, 19.68302782, 25.54695918,
 17.26691093, 16.01035524, 25.87512519, 19.3705841 , 29.52965183,
 23.10173719, 31.37342903, 17.55332715, 25.82107048, 34.98857199,
 22.91267519, 19.3967501 , 29.34678421, 24.65125376, 16.72971658,
 25.42537393, 30.6751849 , 28.90511192, 18.42571639, 27.56426639,
 14.62706882, 20.02272756, 25.60745002, 28.32959623, 15.91971307,
 20.36020491, 26.04012236, 13.70562148, 23.19186499, 23.2538407 ,
 9.14791655, 21.08680468, 35.13203126, 18.20120981, 12.40579126,
 23.03574753, 11.70030485, 24.10234373, 10.23869501, 22.24788446,
 28.20852115, 20.77401763, 26.01572261, 25.97666619, 20.77471688,
 24.05595237, 9.79658092, 21.55718522, 20.96232324, 14.58941397,
 22.29462592, 23.04513053, 2.87810564, 18.26028545, 17.31405284,
 21.55660947, 24.48282506, 11.46772233, 21.88129799, 25.04349207,
 14.07796126, 19.97841644, 26.61705358, 23.30429098, 27.32736035,
 12.59741065, 19.28050072, 24.94727892, 24.22470232, 29.72519314,
 19.11634391, 31.14895846, 16.43050137, 20.50890111, 27.69026978,
 19.80307948, 26.66386801, 15.01321139, 23.31466084, 26.15446018,
 23.80801526, 27.15999771, 30.37432077, 22.93935948, 34.91159865,
 11.97264266, 26.45153342, 20.25377754, 19.96681079, 12.21677635,
 21.57200937, 23.11587937, 31.05309711, 29.72484228, 18.03669387,
 19.12012649, 28.8679226 , 23.41788443, 14.36679948, 10.91433849,
 23.78530314, 23.58885901, 18.48704182, 15.72569049, 36.3867166 ,
 19.38880373, 19.56932184, 27.03174387, 22.95041998, 22.07807906,
 23.22702436, 17.41528186, 24.66493926, 30.31735718, 13.9998893 ,
 22.25446895, 19.75004517, 20.8350658 , 25.47389672, 24.13577633,
 23.02944605, 36.90324597])
```

Assumptions of Lasso Regression:-

```
In [123]: 1 ## Relationship Between Real Data & Predicted Data.  
2 plt.scatter(Y_test,lasso_pred)           ## IF we get Linear Manner , it is goo  
3 plt.xlabel("Test Truth Data")  
4 plt.ylabel("Test Predicted Data")  
5
```

Out[123]: Text(0, 0.5, 'Test Predicted Data')



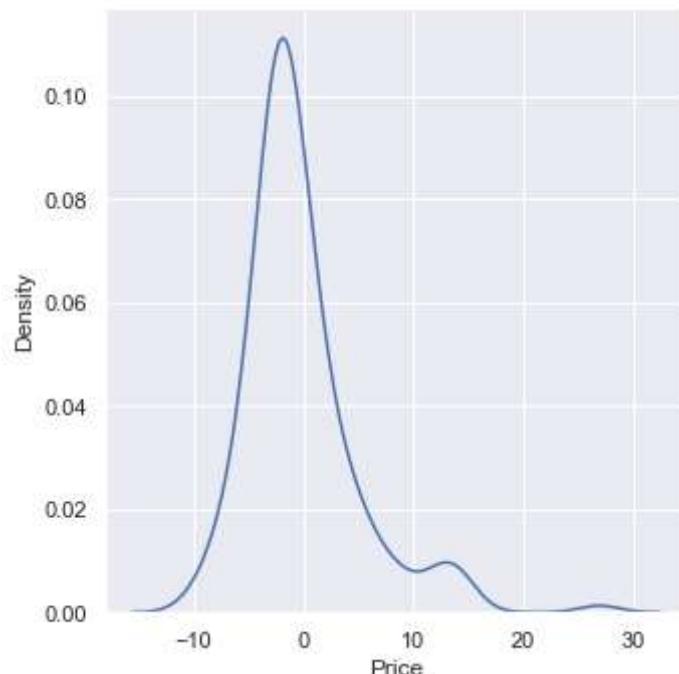
```
In [124]: 1 ## Calculating residuals  
2 residuals=Y_test - lasso_pred  
3
```

```
In [125]: 1 residuals
```

```
Out[125]: 173    -2.480155
274     1.651994
491     -4.181649
72      -2.452247
452     -3.183873
...
110     0.864934
321    -2.373897
265    -1.335776
29     -2.029446
262    11.896754
Name: Price, Length: 167, dtype: float64
```

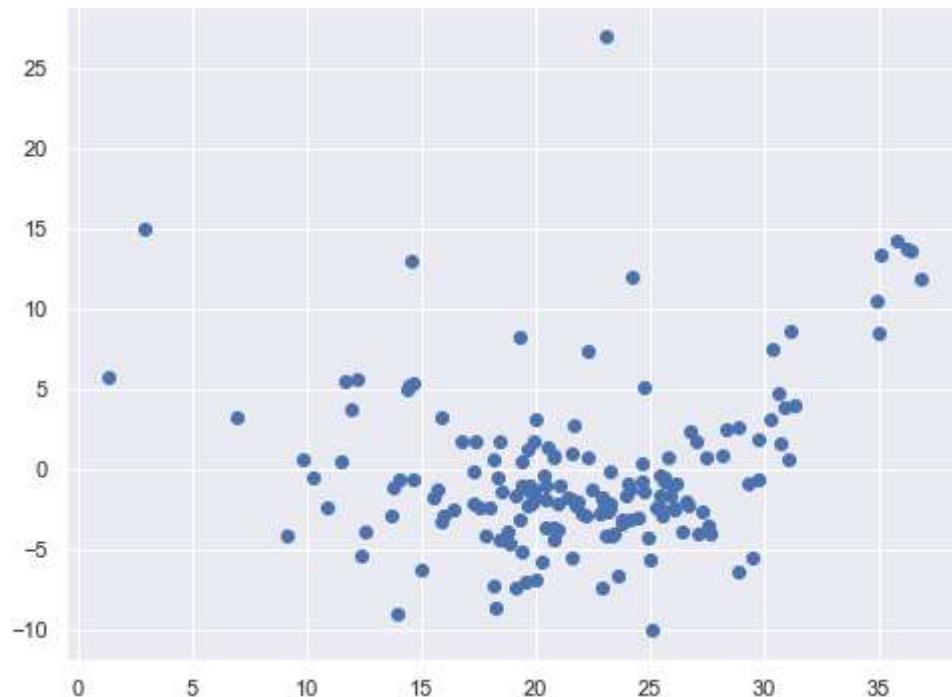
```
In [126]: 1 ## Distribution of residual are Approximately Normal Distribution
2 sns.displot(residuals,kind="kde")
3
```

```
Out[126]: <seaborn.axisgrid.FacetGrid at 0x15845da6730>
```



```
In [127]:  
1 ## Scatter plot with predictions and residual  
2 ##uniform distribution  
3  
4 plt.scatter(lasso_pred,residuals)
```

```
Out[127]: <matplotlib.collections.PathCollection at 0x15845e50130>
```



Performance Metrics

```
In [128]:  
1 ## Performance Metrics  
2 from sklearn.metrics import mean_squared_error  
3 from sklearn.metrics import mean_absolute_error  
4 print(mean_squared_error(Y_test, lasso_pred))  
5 print(mean_absolute_error(Y_test, lasso_pred))  
6 print(np.sqrt(mean_squared_error(Y_test, lasso_pred)))
```

```
26.16637721498099
```

```
3.6464026430077423
```

```
5.11530812512609
```

R Squared and Adjusted R Squared

```
In [129]: 1 ## R Squared  
2 from sklearn.metrics import r2_score  
3 score=r2_score(Y_test, lasso_pred)  
4 print(score)  
5
```

0.6542429577734992

```
In [130]: 1 ## Adjusted R square  
2 #display adjusted R-squared  
3 1 - (1-score)*(len(Y_test)-1)/(len(Y_test)-X_test.shape[1]-1)
```

Out[130]: 0.6248649084339926

Elastic Net Regression Model

```
In [132]: 1 ## Elastic Net Regression  
2 from sklearn.linear_model import ElasticNet
```

```
In [133]: 1 elastic = ElasticNet()
```

```
In [134]: 1 elastic
```

Out[134]: ElasticNet()

```
In [137]: 1 elastic.fit(X_train, Y_train)
```

Out[137]: ElasticNet()

print the coefficients and the intercept

```
In [138]: 1 ## ## print the coefficients  
2 print(elastic.coef_)
```

```
[-0.36520114  0.          -0.14336748  0.63145824 -0.25193148  2.34999448  
-0.          -0.          -0.          -0.25649969 -1.23951556  0.56384945  
-2.56053213]
```

```
In [139]: 1 ## print the Intercepts  
2 print(elastic.intercept_)
```

22.970796460176988

```
In [140]: 1 ## Prediction for the Test data  
2 elastic_pred = elastic.predict(X_test)
```

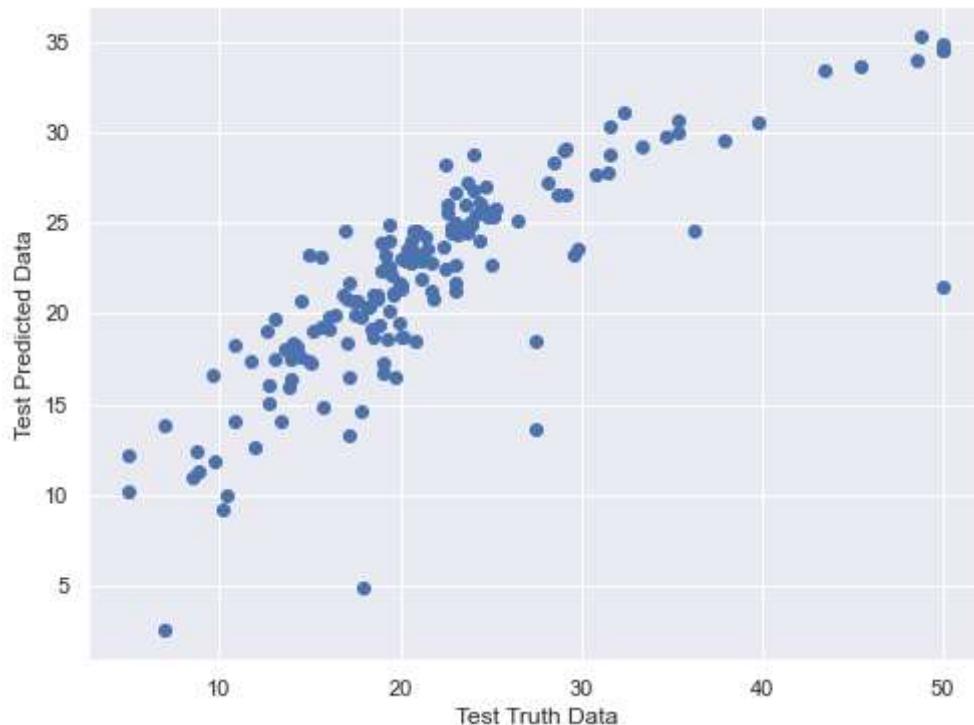
```
In [141]: 1 elastic_pred
```

```
Out[141]: array([26.04802695, 31.11448131, 18.09845158, 24.74715491, 19.13029713,
 23.07195028, 19.8492127 , 16.42921582, 20.98280883, 21.03040905,
 23.59247585, 22.4067143 , 2.50342106, 22.86968897, 21.05836477,
 23.53088819, 19.32942155, 9.24659633, 34.51755093, 18.33111982,
 25.39963891, 26.53220506, 16.04212388, 23.68595117, 18.22309609,
 15.9070075 , 22.91791506, 17.40135861, 22.80881602, 20.34960072,
 21.28107265, 25.0664737 , 23.29041734, 18.52289666, 16.68946719,
 20.17099878, 29.78000437, 22.08911412, 24.00624402, 24.52109601,
 16.51539744, 27.25142517, 34.8940966 , 20.75229792, 25.54944362,
 17.27877681, 17.51067948, 25.422475 , 19.45141801, 28.72445431,
 23.85816391, 30.64335445, 19.05778782, 25.10137208, 33.43673587,
 21.9368327 , 19.10068361, 28.38705767, 24.91075492, 18.68821158,
 25.41735754, 29.96236233, 27.77368373, 18.66077461, 26.83456776,
 18.72984267, 19.66634919, 25.37569386, 27.64862833, 15.09326887,
 21.6230625 , 24.42218348, 14.00935207, 22.80340884, 23.31057037,
 10.15388121, 21.41270277, 33.98445117, 18.23197774, 13.83630127,
 23.23840504, 13.33915878, 24.55297788, 11.80396525, 22.6039322 ,
 29.04777925, 19.93864988, 25.40796591, 25.4253774 , 20.79626634,
 24.35937771, 9.98031645, 21.1883148 , 21.7010251 , 13.64158366,
 21.70329066, 21.48780024, 4.89921219, 16.60651403, 16.48691364,
 22.52662793, 24.23810699, 12.67234464, 21.73288769, 24.94869622,
 14.02785155, 20.34560161, 25.81816846, 23.27665603, 26.98968083,
 12.4461064 , 18.53919342, 24.57036836, 24.5991159 , 28.78917289,
 17.35695925, 30.55890904, 17.49669771, 20.81635985, 27.26668735,
 20.67056474, 26.10145269, 11.29182557, 23.22360335, 25.76790464,
 23.6220014 , 26.68354582, 29.53505955, 23.09099441, 33.69060315,
 14.86234562, 26.00700282, 20.72314768, 21.04261631, 14.58583453,
 19.80159048, 23.16541552, 30.31837307, 29.06727633, 19.21664987,
 19.96431131, 28.26947089, 24.04394758, 18.60022435, 10.97392042,
 23.98834146, 24.56611841, 18.41001674, 17.5657926 , 34.56534242,
 18.12003269, 19.01050052, 26.55906015, 23.10161055, 22.69122071,
 22.66142713, 17.29393495, 22.68213782, 29.22756893, 12.14001806,
 23.24525465, 20.77586134, 21.27148575, 25.02037796, 24.44501814,
 23.10652629, 35.30973171])
```

Assumptions of ElasticNet Regression:-

```
In [142]: 1 ## Relation Between Real Data and Predicted Data
2 plt.scatter(Y_test,elastic_pred)      ## IF we get Linear Manner , it is good
3 plt.xlabel("Test Truth Data")
4 plt.ylabel("Test Predicted Data")
```

Out[142]: Text(0, 0.5, 'Test Predicted Data')



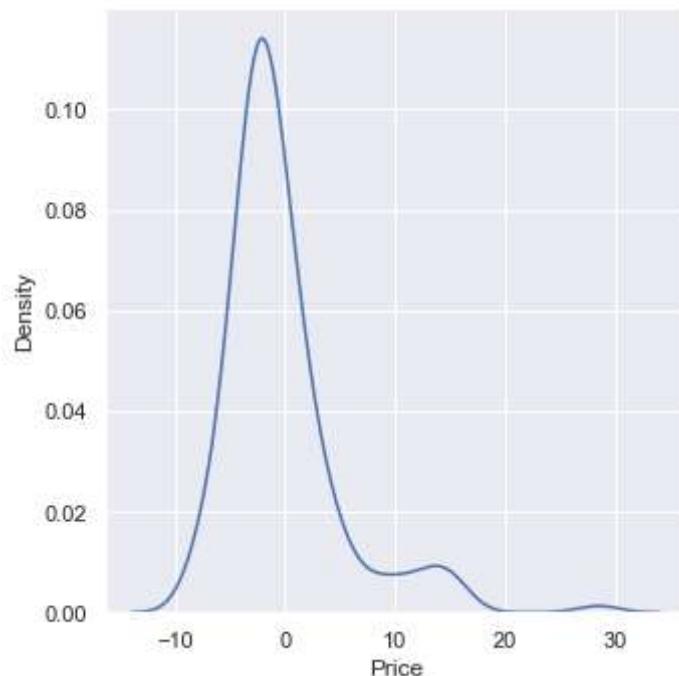
```
In [143]: 1 ## Calculating residuals
2 residuals=Y_test- elastic_pred
3
```

```
In [144]: 1 residuals
```

```
Out[144]: 173    -2.448027
274     1.285519
491     -4.498452
72      -1.947155
452     -3.030297
...
110     0.428514
321    -1.920378
265    -1.645018
29     -2.106526
262    13.490268
Name: Price, Length: 167, dtype: float64
```

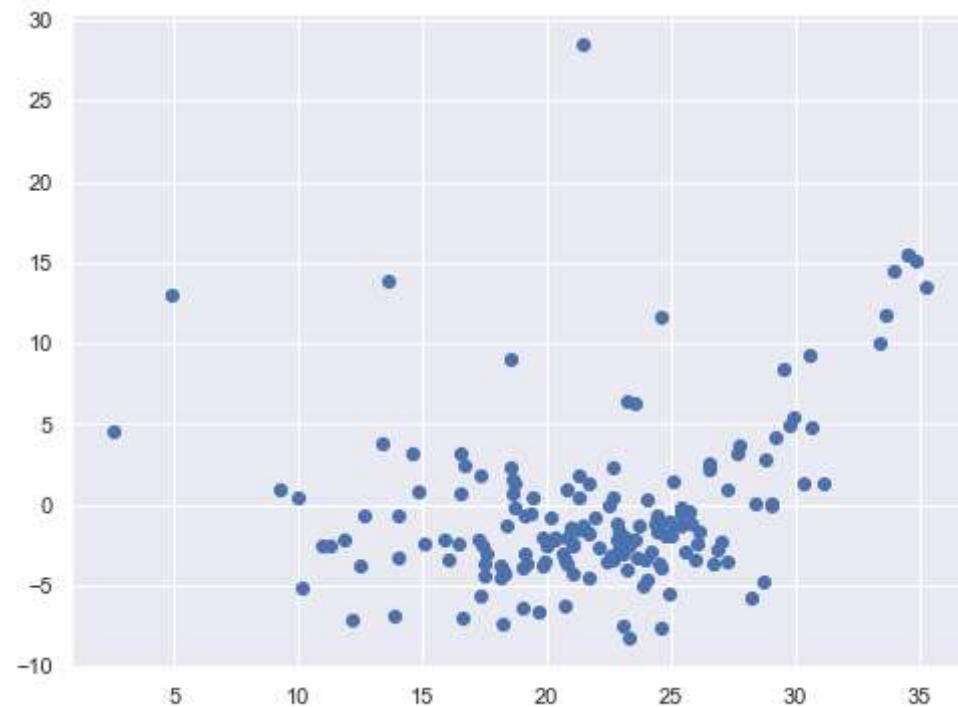
```
In [145]: 1 ## Distribution of residual are Approximately Normal Distribution
2 sns.displot(residuals,kind="kde")
3
```

```
Out[145]: <seaborn.axisgrid.FacetGrid at 0x15845e0a700>
```



```
In [146]: 1 ## Scatter plot with predictions and residual  
2 ##uniform distribution  
3 plt.scatter(elastic_pred,residuals)  
4
```

Out[146]: <matplotlib.collections.PathCollection at 0x15845e78c40>



Performance Metrics

In [147]:

```
1 ## Performance Metrics
2 from sklearn.metrics import mean_squared_error
3 from sklearn.metrics import mean_absolute_error
4 print(mean_squared_error(Y_test,elastic_pred))
5 print(mean_absolute_error(Y_test,elastic_pred))
6 print(np.sqrt(mean_squared_error(Y_test,elastic_pred)))
7
```

```
27.140175406489988
3.627745135070299
5.209623345932985
```

R Squared and Adjusted R Squared

In [148]:

```
1 ## R Squared
2 from sklearn.metrics import r2_score
3 score=r2_score(Y_test,elastic_pred)
4 print(score)
```

```
0.641375391902405
```

In [149]:

```
1 ## Adjusted R square
2 #display adjusted R-squared
3 1 - (1-score)*(len(Y_test)-1)/(len(Y_test)-X_test.shape[1]-1)
```

Out[149]: 0.610904019972544

Thank you