Linear Regression on Boston House Price Prediction

Submiitted by : Nilutpal Das

In [1]:
```
# importing necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
# Loading dataset
from sklearn.datasets import load_boston
```

In [120…
```
boston = load_boston()
import warnings
warnings.filterwarnings("ignore")
```

In [4]:
```
# Parameters
boston.keys()
```

Out[4]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename', 'data_module'])
```

In [5]:
```
# Values
boston.values()
```

Out[5]:

```
dict_values([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]]), array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 1
8.9, 15. ,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
        21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
        35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
        19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
        20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
        23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
        33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
        21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
        20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
        23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
        15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
        17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
        25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
        23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
        32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
        34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
        20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
        26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
        31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
        22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
        42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
        36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
        32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
        20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
        20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]), array(['CR
IM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'), ".. _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/categoric
al predictive. Median Value (attribute 14) is usually the target.\n\n    :Attribute I
```

```
nformation (in order):\n        - CRIM     per capita crime rate by town\n        - Z
N       proportion of residential land zoned for lots over 25,000 sq.ft.\n        - I
NDUS    proportion of non-retail business acres per town\n        - CHAS     Charles
River dummy variable (= 1 if tract bounds river; 0 otherwise)\n        - NOX      nit
ric 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 p
roperty-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\n    :Missing Attribute Values: None\n\n    :Creator: Ha
rrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttps://
archive.ics.uci.edu/ml/machine-learning-databases/housing/\n\n\nThis dataset was take
n 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.   Us
ed in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley, 1980.   N.B. Vario
us transformations are used in the table on\npages 244-261 of the latter.\n\nThe Bost
on house-price data has been used in many machine learning papers that address regres
sion\nproblems.   \n     \n.. topic:: References\n\n   - Belsley, Kuh & Welsch, 'Regr
ession diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley,
1980. 244-261.\n   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Lear
ning. In Proceedings on the Tenth International Conference of Machine Learning, 236-2
43, University of Massachusetts, Amherst. Morgan Kaufmann.\n", 'boston_house_prices.c
sv', 'sklearn.datasets.data'])
```

In [6]:
```
# Description
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 otherwi
se)
        - 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 to
wn
        - 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/


This dataset was taken from the StatLib library which is maintained at Carnegie Mello
n 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 addres
s 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 Proceed
ings on the Tenth International Conference of Machine Learning, 236-243, University o
f Massachusetts, Amherst. Morgan Kaufmann.
```

In [7]:
```
# Data matrix
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 [8]:
```
# Target data
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.  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 [9]:
```
# Feature/Column names
print(boston.feature_names)
```

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

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

In [11]:
```
# Showing top 5 rows
dataset.head()
```

Out[11]:

	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 [12]:

```
# target feature = Price
dataset['Price']=boston.target
```

In [13]:

```
# Showing top 5 rows
dataset.head()
```

Out[13]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Pr
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	2
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
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
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	3
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	3

In [14]:

```
# Basic info of the dataset
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]:

```
# Statistical info
dataset.describe()
```

Out[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DI
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.79504
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.10571
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.12960
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.10017
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.20745
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.18842
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.12650

In [16]:

```
# check the missing values
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]:

```
dataset.isnull().sum().sum()
```

Out[17]:

```
0
```

In [18]:

```
# correlation
dataset.corr()
```

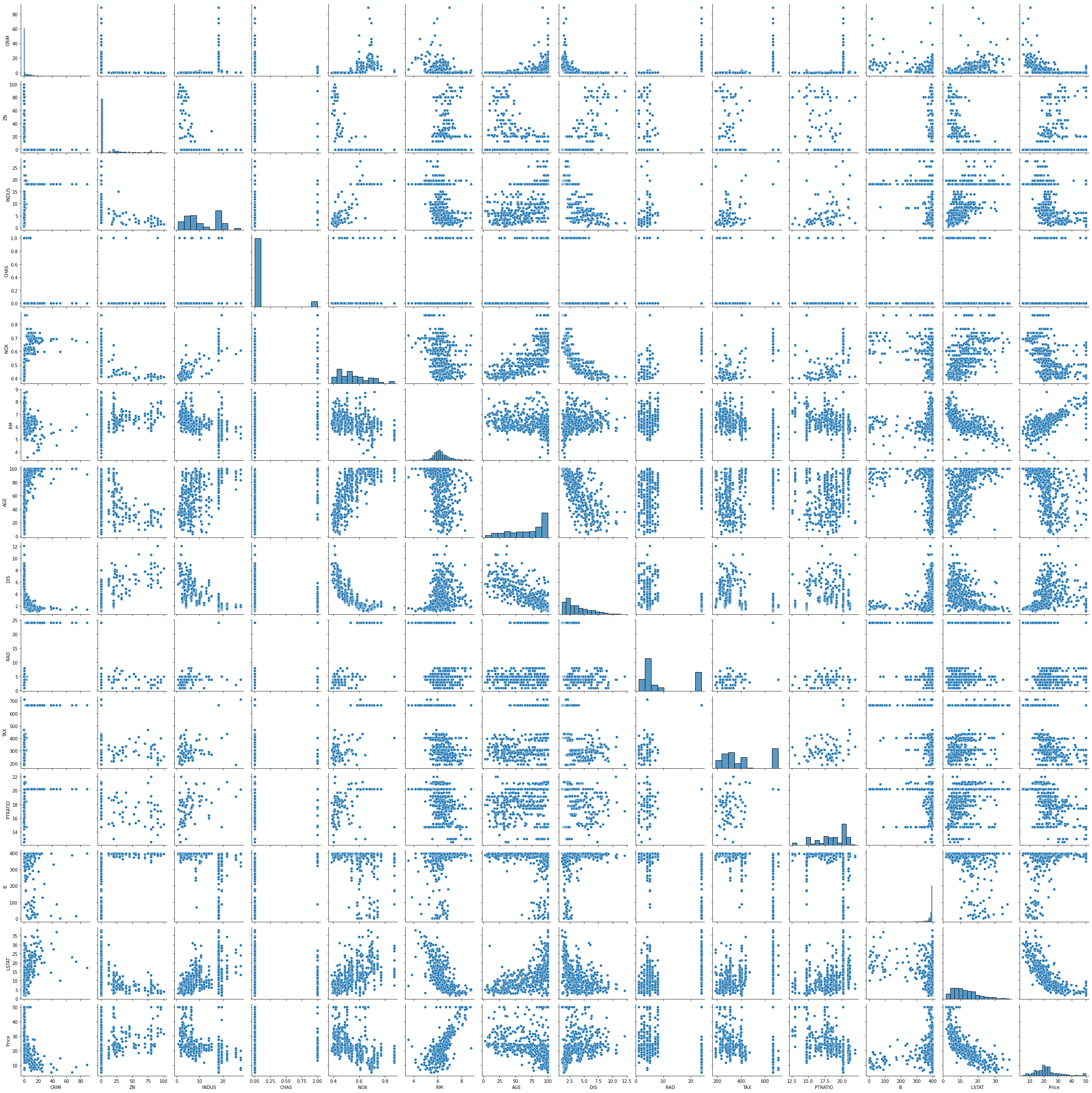
Out[18]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	R
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.62
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.31
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.59
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.00
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.61
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.20
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.45
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.49
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.00
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.91
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.46
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.44
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.48
Price	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.38

In [19]:

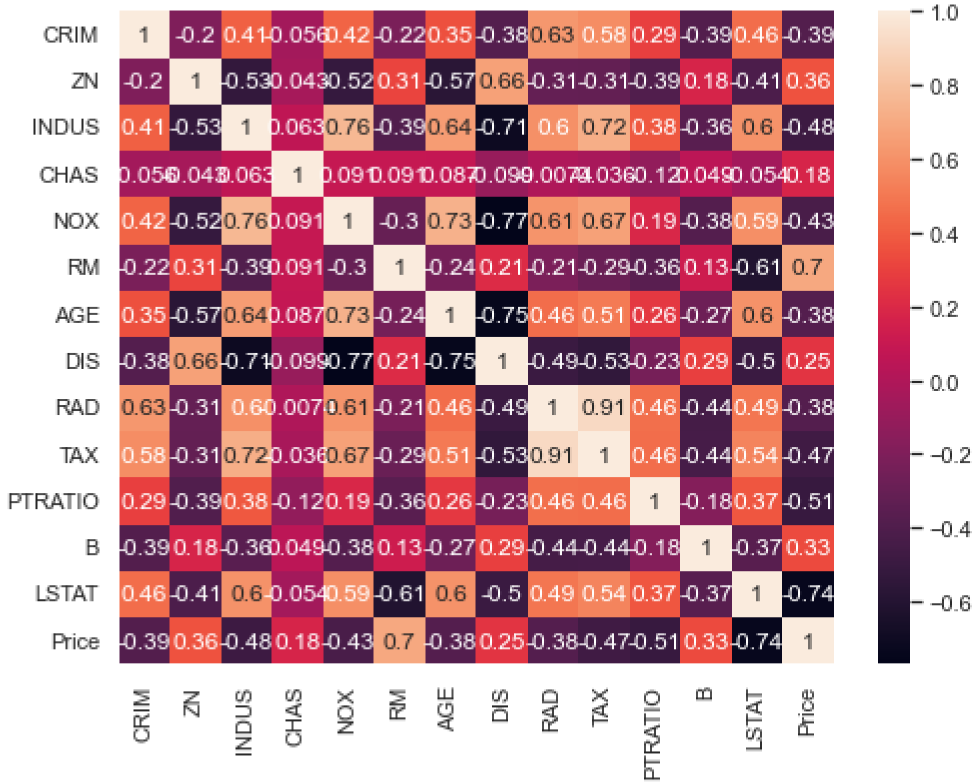
```python
# Checking multicolinearity
import seaborn as sns
sns.pairplot(dataset)
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x22c80065340>



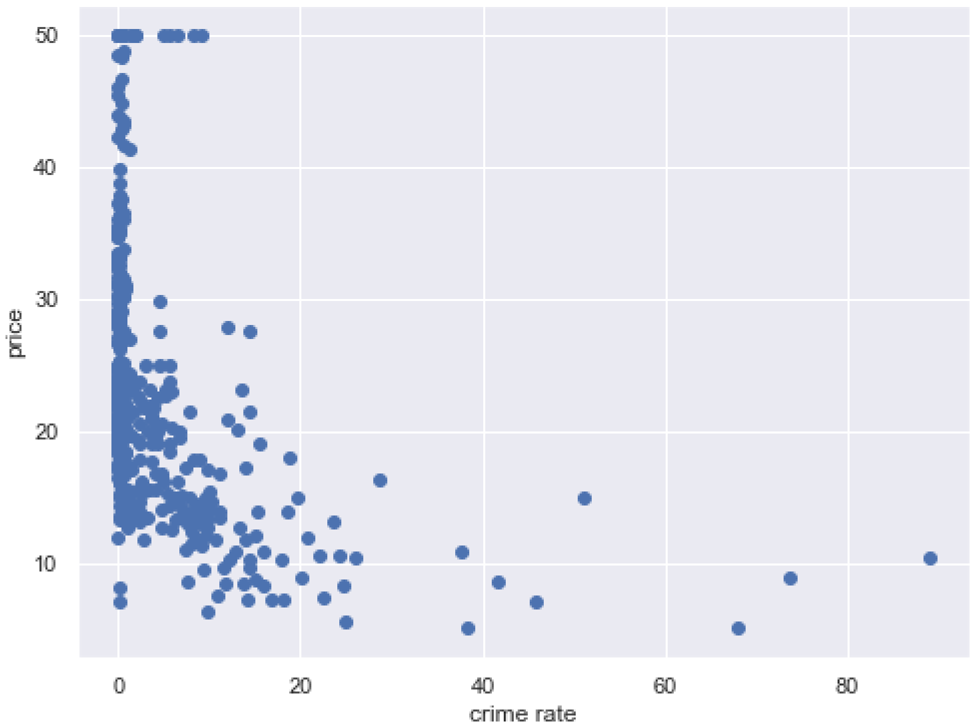
In [20]:
```python
# Checking correlation using heatmap
sns.set(rc={'figure.figsize':(8,6)})
sns.heatmap(dataset.corr(),annot=True)
```

Out[20]: <AxesSubplot:>



In [21]:
```python
# Analysing 'CRIM' feature with target feature 'Price'
plt.scatter(dataset['CRIM'], dataset['Price'])
plt.xlabel("crime rate")
plt.ylabel("price")
```

Out[21]:
```
Text(0, 0.5, 'price')
```

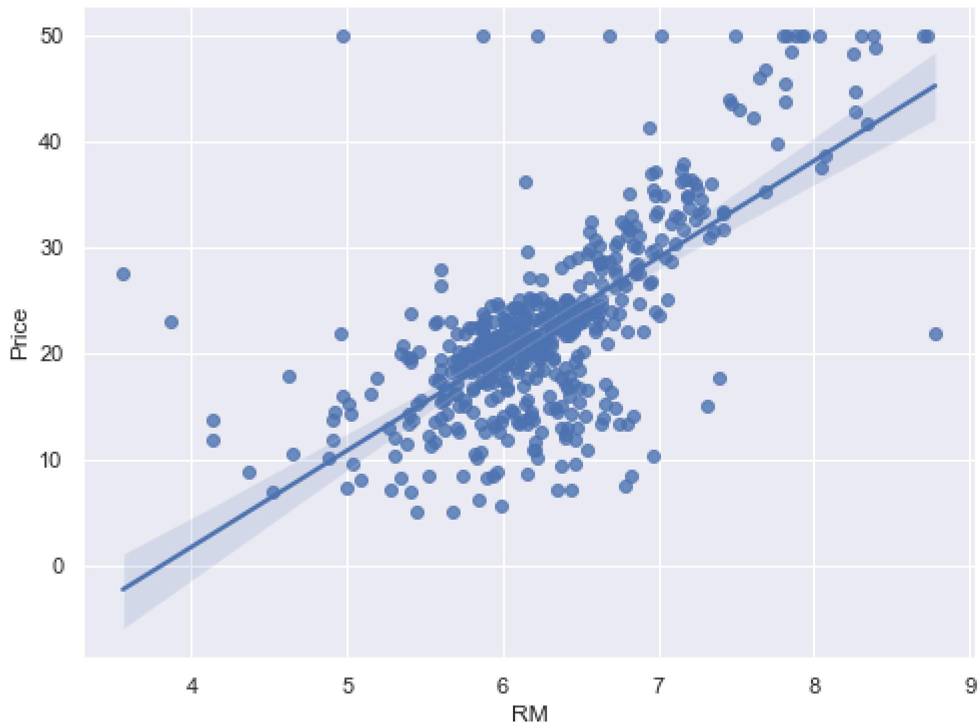


In [22]:
```python
# Analysing the 'RM' feature with target feature 'Price'
plt.scatter(dataset['RM'],dataset['Price'])
```

Out[22]:
```
<matplotlib.collections.PathCollection at 0x22c8bcbde80>
```

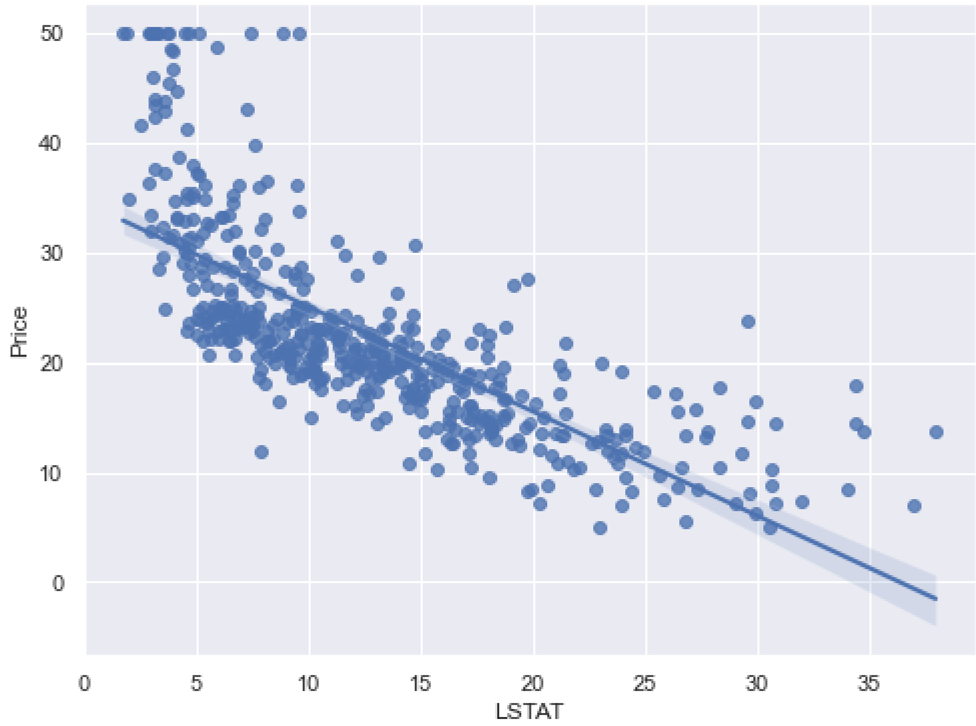
In [23]:
```
# Impleamenting linear regression using regplot
sns.regplot(x="RM",y="Price",data=dataset)
```

Out[23]: <AxesSubplot:xlabel='RM', ylabel='Price'>



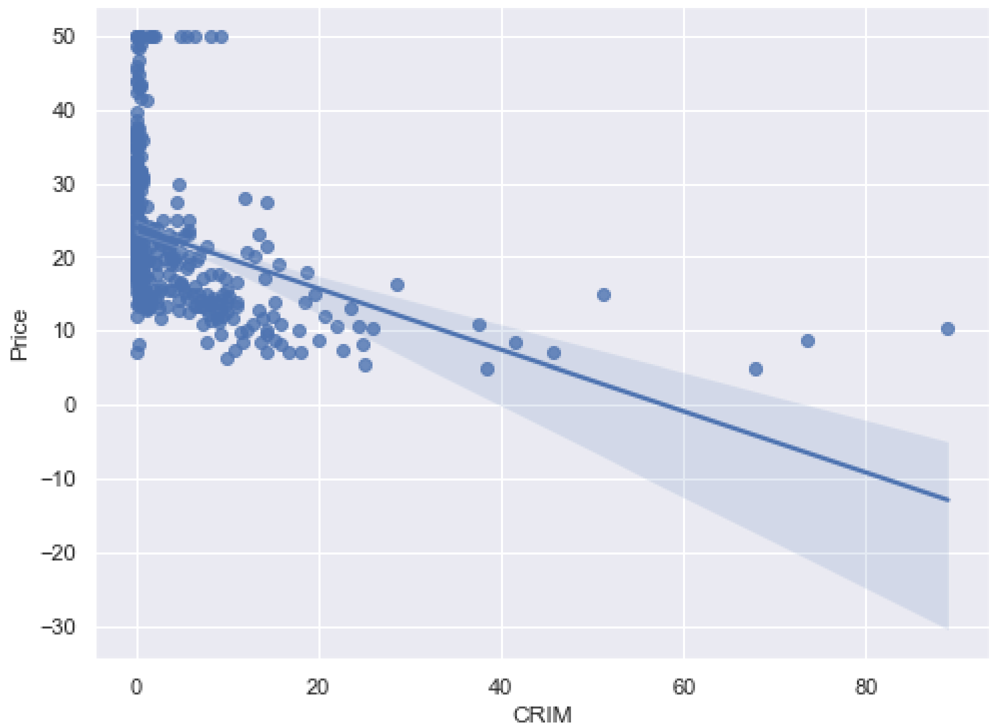
In [24]:
```
# Impleamenting linear regression using regplot
sns.regplot(x="LSTAT",y="Price",data=dataset)
```

Out[24]: <AxesSubplot:xlabel='LSTAT', ylabel='Price'>



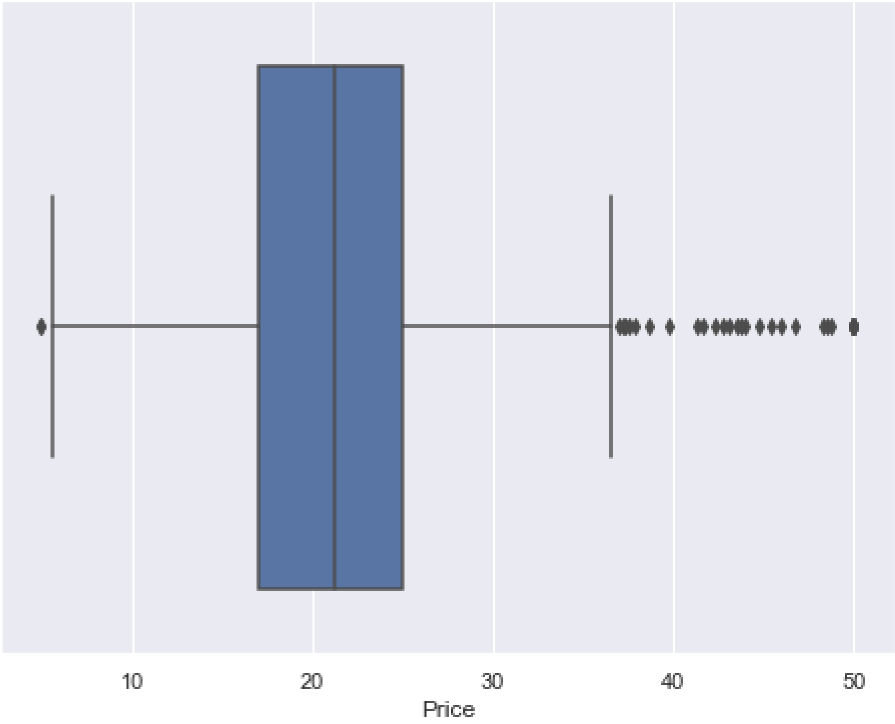
In [25]:
```
# Impleamenting linear regression using regplot
sns.regplot(x="CRIM",y="Price",data=dataset)
```

Out[25]: <AxesSubplot:xlabel='CRIM', ylabel='Price'>



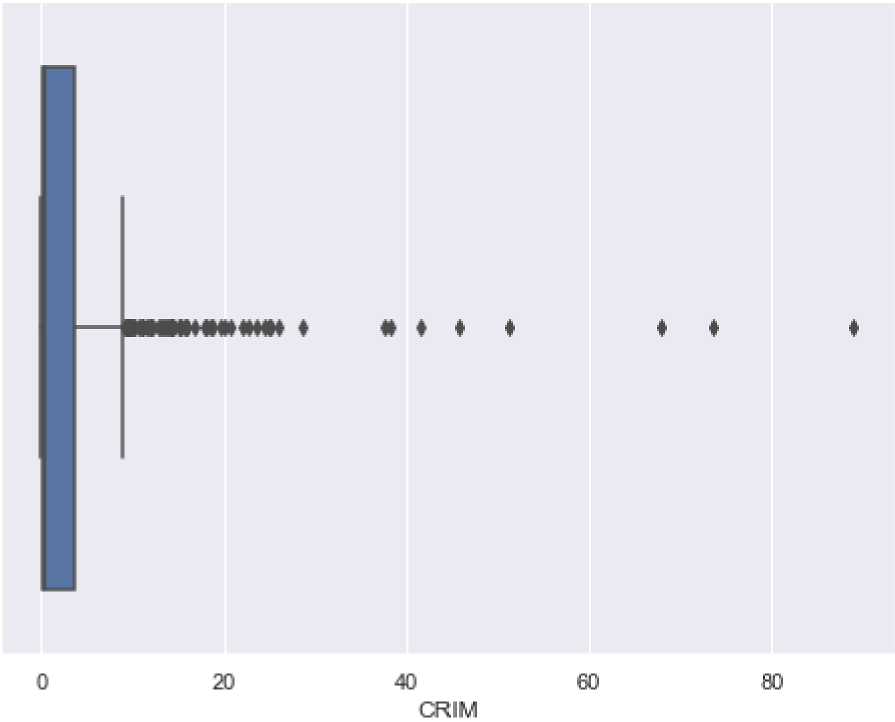
In [26]:
```
# Checking the outlier using boxplot on 'Price'
sns.boxplot(dataset['Price'])
```

Out[26]: <AxesSubplot:xlabel='Price'>




```
In [27]: # Checking outlier on 'CRIM'
         sns.boxplot(dataset['CRIM'])
```

Out[27]: <AxesSubplot:xlabel='CRIM'>




```
In [28]: # Columns
         dataset.columns
```

```
Out[28]: Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
                'PTRATIO', 'B', 'LSTAT', 'Price'],
               dtype='object')
```

```
In [29]: # Independent Features and Dependent Features
         X=dataset.iloc[:,:-1]
         y=dataset.iloc[:,-1]
```

In [30]:
```python
X.head()
```

Out[30]:

	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 [31]:
```python
# Target Feature
y
```

Out[31]:
```
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 [32]:
```python
# Splitting the Dataset
from sklearn.model_selection import train_test_split
```

In [33]:
```python
# Splitting the dataset into training and testing data.
# test_size = 33% of the data is selected for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state
```

In [34]:
```python
# Training data i.e Independent variables
X_train
```

Out[34]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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	29.53
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	9.09
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	30.62
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	6.36
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	17.09
...	...	...	...	...	...	...	...	...	...	...	...	...	...
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	7.20
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	8.47
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	13.34
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	14.81
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	10.45

339 rows × 13 columns

In [35]:

```
# Testing data i.e. Independent variables
X_test
```

Out[35]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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	8.93
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	5.03
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	4.67
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	5.89
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	13.28
...	...	...	...	...	...	...	...	...	...	...	...	...	...
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	16.59
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	17.73
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	4.56
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	11.97
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	9.28

167 rows × 13 columns

In [36]:

```
# Training data i.e. Dependent variable
y_train
```

Out[36]:
```
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 [37]:
```
# Testing data i.e. Dependent variable
y_test
```

Out[37]:
```
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 [38]:
```
# Checking the shape of training data i.e. independent variable
X_train.shape
```

Out[38]:
```
(339, 13)
```

In [39]:
```
# Checking the shape of training data i.e. dependent variable
y_train.shape
```

Out[39]:
```
(339,)
```

In [40]:
```
y_test.shape
```

Out[40]:
```
(167,)
```

In [41]:
```
X_test.shape
```

Out[41]:
```
(167, 13)
```

In [42]:
```
# feature engineering
```

In [43]:
```
# standardize or feature scaling the dataset
```

In [44]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [45]:
```
scaler
```

Out[45]:
```
▾ StandardScaler
StandardScaler()
```

In [46]:
```
X_train=scaler.fit_transform(X_train)
```

In [47]:
```
X_test=scaler.transform(X_test)
```

In [48]:
```
X_train
```

Out[48]:
```
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 [49]:
```
X_test
```

Out[49]:
```
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 ]])
```

In [50]:
```
# model training
```

In [51]:
```
from sklearn.linear_model import LinearRegression
```

In [52]:
```
regression=LinearRegression()
```

In [53]:
```
regression
```

Out[53]:
```
▾ LinearRegression
LinearRegression()
```

In [54]:
```
regression.fit(X_train,y_train)
```

Out[54]:
```
▾ LinearRegression
LinearRegression()
```

In [55]:
```
# print the coefficients
```

In [56]:
```
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 [57]:
```python
print(regression.intercept_)
```

```
22.077286135693214
```

In [58]:
```python
# prediction
reg_pred = regression.predict(X_test)
```

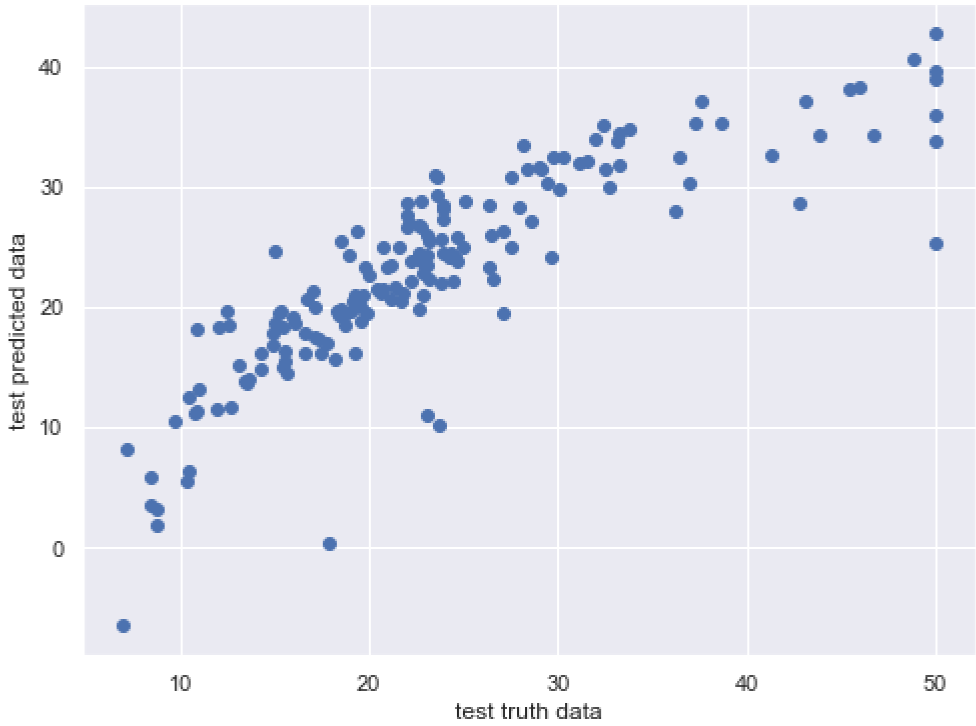
In [59]:
```python
reg_pred
```

Out[59]:
```
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])
```

In [60]:
```python
# assumption linear regression
```

In [61]:
```python
plt.scatter(y_test,reg_pred) #actual and predicted data are similar
plt.xlabel("test truth data")
plt.ylabel("test predicted data")
```

Out[61]:
```
Text(0, 0.5, 'test predicted data')
```



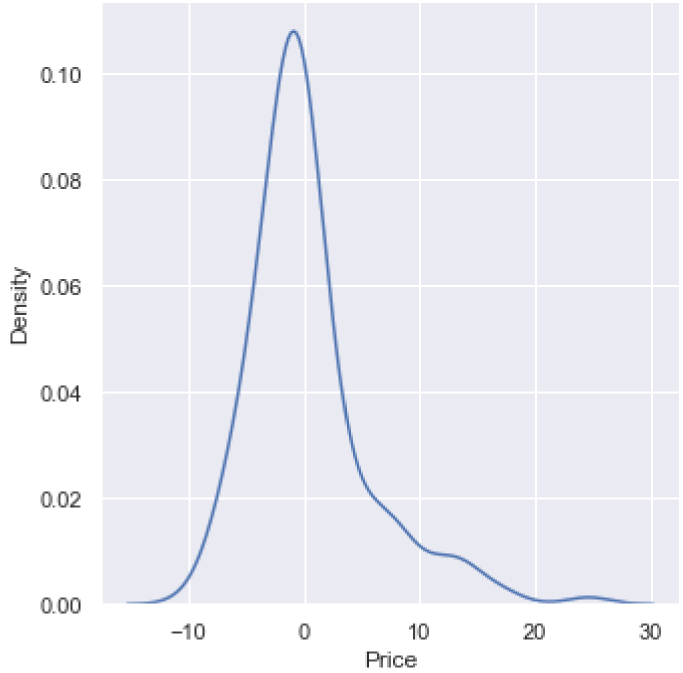
In [62]:
```
# Residual = difference between actual data and predicted data
residuals=y_test-reg_pred
```

In [63]:
```
residuals
```

Out[63]:
```
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
```

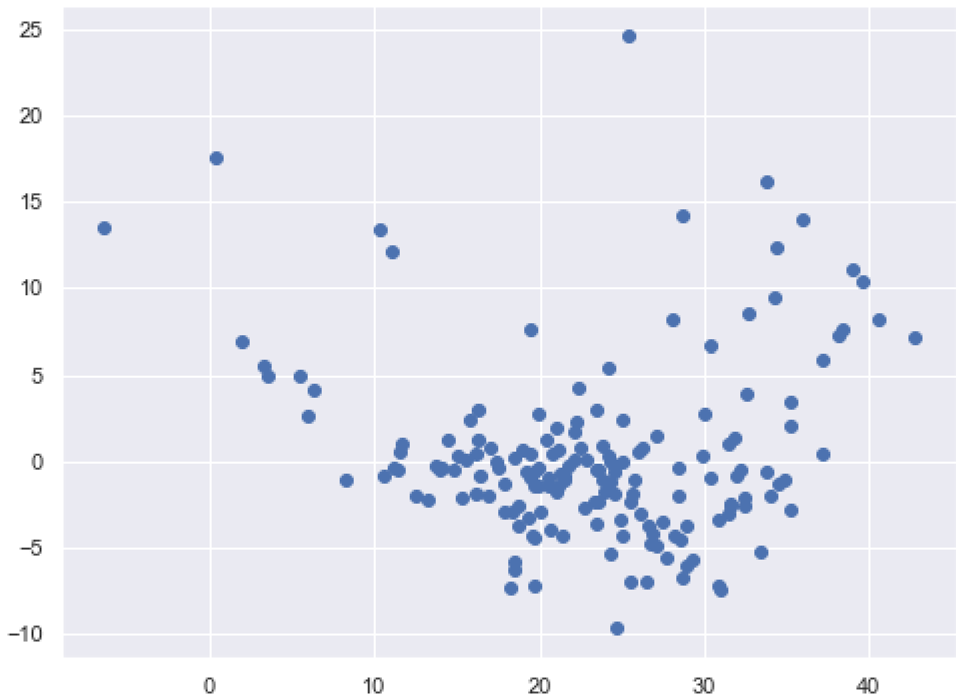
In [64]:
```
# Plotting displot for residual
sns.displot(residuals,kind='kde')
```

Out[64]:
```
<seaborn.axisgrid.FacetGrid at 0x22c8ed95cd0>
```



In [65]:
```
# Scatter plot prediction and residual
# uniform distribution
plt.scatter(reg_pred, residuals) #model is good
```

Out[65]: <matplotlib.collections.PathCollection at 0x22c8fe4f2b0>



In [66]:
```
# Performance Metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,reg_pred))
print(mean_absolute_error(y_test,reg_pred))
print(np.sqrt(mean_squared_error(y_test,reg_pred)))
```

```
27.1009917099625
3.520658529879791
5.205861284164466
```

In [67]:
```
# R squared
```

```
from sklearn.metrics import r2_score
score = r2_score(y_test,reg_pred)
print(score)
```

```
0.7165219393967553
```

In [68]:
```
# Adjusted R squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[68]:
```
0.6924355682343881
```

In [69]:
```
# Ridge regression
from sklearn.linear_model import Ridge
ridge=Ridge()
```

In [70]:
```
# fitting the data
ridge.fit(X_train,y_train)
```

Out[70]:
```
▾ Ridge
Ridge()
```

In [81]:
```
print(ridge.coef_)
```

```
[-1.27565151  1.5819406  -0.1614208   0.37673024 -1.72386872  2.24434183
  0.30956702 -3.26398836  2.60274628 -1.99924081 -1.75198618  1.25002916
 -3.81456087]
```

In [82]:
```
print(ridge.intercept_)
```

```
22.077286135693214
```

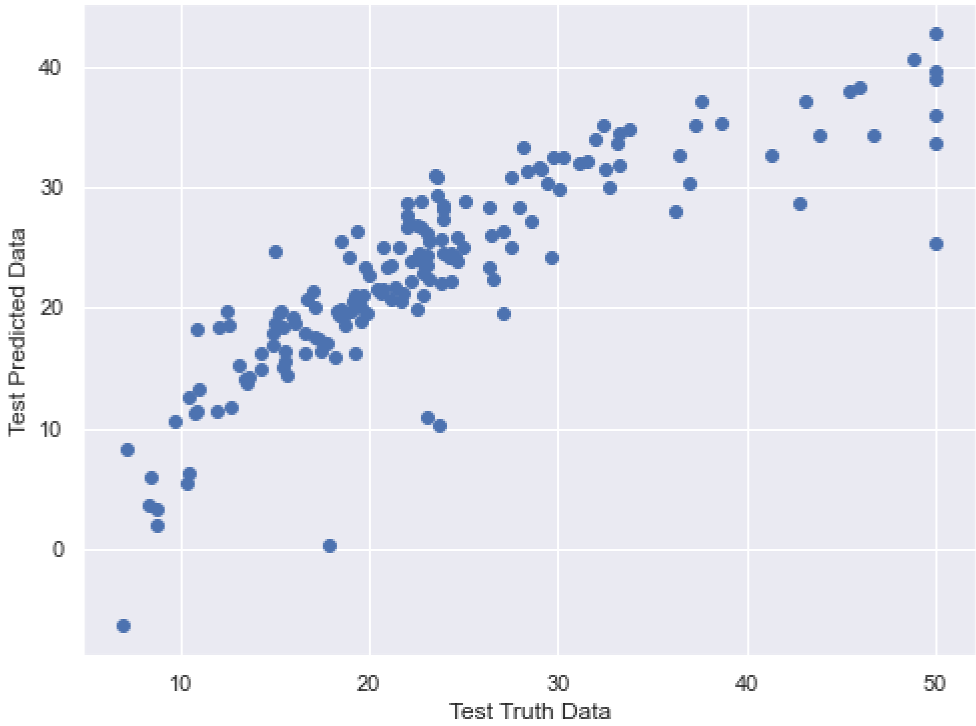
In [78]:
```
# predicting the data
ridge_pred=ridge.predict(X_test)
```

In [80]:
```
ridge_pred
```

Out[80]:
```
array([31.32951625, 31.98180665, 30.96523995, 22.45112285, 18.93171888,
       16.21770197, 35.96932532, 14.8453389 , 25.00644473, 37.08826243,
       21.49615236, 30.86395535, 27.9880323 , 33.98239498, 33.72731108,
       40.61743429, 24.27292247, 23.33888547, 25.52862017, 21.42716828,
       32.68689234, 17.88582539, 25.50293435, 25.01797349, 32.58757636,
       20.48521647, 19.51598666, 16.94098815, 38.35803356,  0.33567931,
       32.44411299, 32.10347472, 26.13567232, 23.81384315, 20.64388179,
       19.71829821,  3.56174179, 35.17319673, 27.02020897, 27.65038259,
       34.3408154 , 29.77237182, 18.39828682, 31.55283209, 17.92580288,
       28.51408759, 19.49631857, 21.65517408, 38.03589465, 16.47721333,
       24.56300743, 19.66060562, 24.490545  , 34.33513167, 26.7462751 ,
       34.83714079, 21.08524522, 19.88396747, 18.65820105, 24.71538111,
       20.00248822, 23.58585608, 39.60689645, 42.79543819, 30.3548884 ,
       17.07425788, 23.84421168,  3.23169724, 31.42539336, 28.75103892,
       18.49739555, 27.14667811, 19.64621723, 25.28950017, 25.07871104,
       10.32212282, 38.94009655,  8.26854141, 18.50624966, 30.39028455,
       22.88702308, 21.08817927, 20.09060901, 28.70289649, 30.81533585,
       28.22566424, 26.28189093, 31.61850553, 22.15784726, -6.42142112,
       21.55950809, 19.89786415, 24.96571959, 23.47361425, 19.25709566,
       18.80383821, 27.37954116, 22.19229114, 26.78224659, 23.40784376,
       23.92754566, 19.18858516, 21.09794643, 10.90877661, 13.8058827 ,
       20.78603584, 23.49652544, 14.19685075, 28.86443391, 15.85586096,
       15.26402087, 22.3935837 , 26.6360939 , 28.87654523, 24.25975975,
       18.26463183, 16.26557102, 17.44937859, 15.58602415, 21.2407358 ,
       33.72594686, 30.0710014 , 21.17366551, 14.04587364, 16.21847821,
       29.26644762, 13.18724919, 22.07232566, 24.34918815, 31.88230457,
       33.34230018,  5.95941842, 35.14730418, 24.25694454, 17.55532023,
       24.27022839, 28.4213874 , 34.47544702,  6.3238347 ,  2.03912756,
       28.40127604, 12.59079125, 18.32110122, 19.75915926,  5.51559383,
       14.42137586, 37.15183113, 25.8605775 , 23.29888263, 26.39528404,
       11.42000684, 20.48891462, 35.29528497, 20.61619917, 11.45777136,
       16.36445822, 24.57014519, 10.51041916, 15.13830095, 26.01152356,
       11.22987126, 11.70179781, 19.39451509, 19.59207236, 32.42949   ,
       22.67098418, 25.68376364])
```

In [84]:
```python
# Relationship between real data and predicted data
plt.scatter(y_test,ridge_pred)
plt.xlabel('Test Truth Data')
plt.ylabel('Test Predicted Data')
```

Out[84]:
```
Text(0, 0.5, 'Test Predicted Data')
```



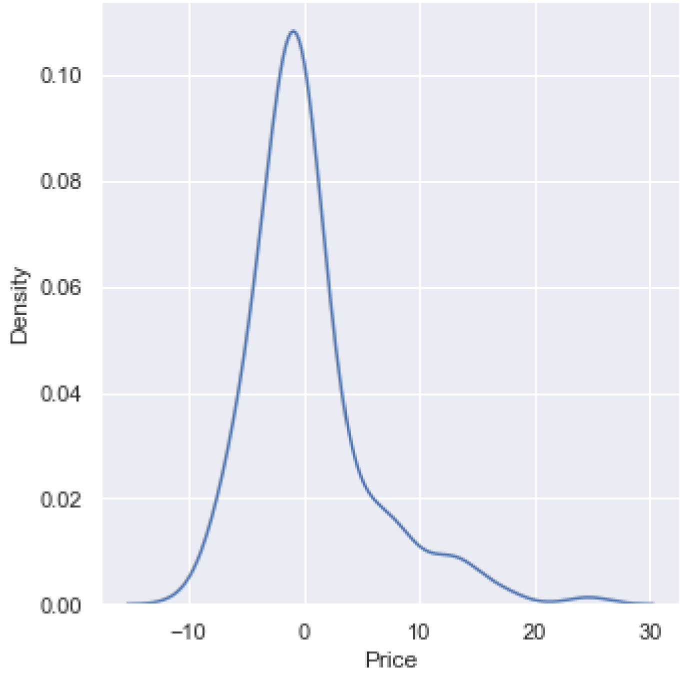
In [86]:
```
# Calculating residual
residuals=y_test - ridge_pred
```

In [87]:
```
residuals
```

Out[87]:
```
305   -2.929516
193   -0.881807
65    -7.465240
349    4.148877
151    0.668281
         ...
442   -0.994515
451   -4.392072
188   -2.629490
76    -2.670984
314   -1.883764
Name: Price, Length: 167, dtype: float64
```

In [88]:
```
# Distribution of residual are approximately Normal Distribution
sns.displot(residuals,kind="kde")
```

Out[88]:
```
<seaborn.axisgrid.FacetGrid at 0x22c8fff4cd0>
```



Observation

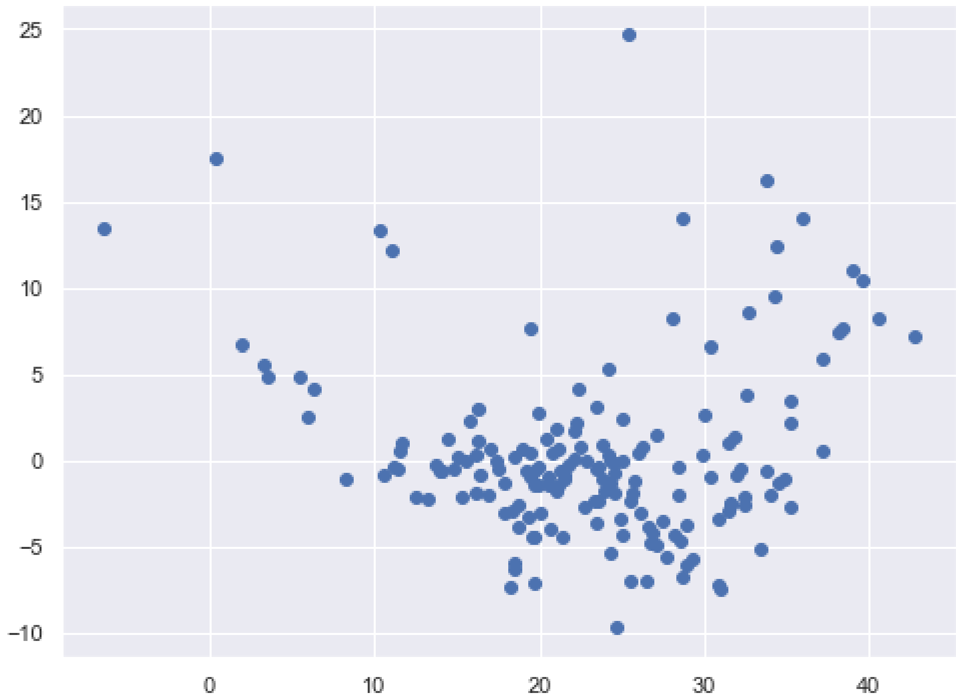
- Distribution is left skewed.

In [89]:

```
# Scatterplot with prediction and residual
# Uniform Distribution
plt.scatter(reg_pred,residuals)
```

Out[89]:

```
<matplotlib.collections.PathCollection at 0x22c900871c0>
```



In [90]:

```
## Performance Metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,ridge_pred))
print(mean_absolute_error(y_test,ridge_pred))
print(np.sqrt(mean_squared_error(y_test,ridge_pred)))
```

```
27.076490001440614
3.5161044263484253
5.20350747106609
```

In [92]:
```
# R Squared
from sklearn.metrics import r2_score
score=r2_score(y_test,ridge_pred)
print(score)
```

```
0.716778228793379
```

In [93]:
```
# Adjusted R Squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[93]:
```
0.6927136338542543
```

In [72]:
```
from sklearn.linear_model import Lasso
lasso=Lasso()
```

In [73]:
```
lasso.fit(X_train,y_train)
```

Out[73]:
```
▾ Lasso
Lasso()
```

In [94]:
```
print(lasso.coef_)
```

```
[-0.05088722  0.          -0.           0.          -0.           2.33991441
 -0.          -0.          -0.          -0.          -1.21265926   0.50346933
 -3.52626441]
```

In [95]:
```
print(lasso.intercept_)
```

```
22.077286135693214
```

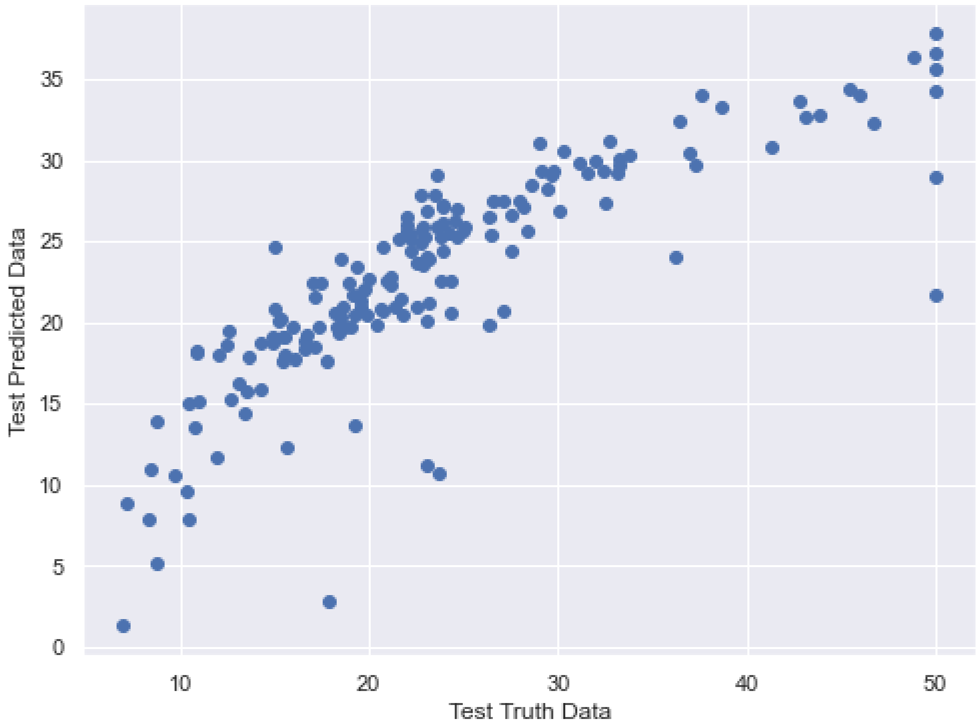
In [96]:
```
lasso_pred=lasso.predict(X_test)
```

In [97]:
```
lasso_pred
```

Out[97]:
```
array([25.64194382, 29.81425297, 27.94324255, 27.55256464, 20.99640298,
       18.74520609, 34.28217994, 15.93009427, 20.70883387, 34.07542731,
       19.90502439, 26.60490365, 24.07990755, 29.92866139, 29.22037693,
       36.40160499, 26.2514407 , 19.88117334, 23.967085  , 22.50869347,
       30.87428332, 18.78300957, 23.92041383, 25.68996484, 32.43275786,
       21.59346217, 20.77097939, 19.17145706, 34.09829244,  2.83421427,
       30.5699873 , 29.29565261, 26.85558827, 25.25346658, 19.26477827,
       19.73302762,  7.84289608, 29.77239449, 25.40207471, 25.60513357,
       32.3846261 , 26.89227407, 18.03007537, 29.36340326, 18.91119501,
       27.26813644, 20.46203931, 21.03622196, 34.39115891, 18.05973586,
       23.67935365, 18.6389767 , 22.52686697, 32.78082702, 26.03741902,
       30.39354515, 20.51475327, 20.94796259, 17.76992156, 24.71515119,
       21.39562999, 22.87363803, 36.66878913, 37.88636344, 28.19838095,
       17.67593653, 24.95639783,  5.16197744, 27.44103022, 33.73592095,
       19.49826488, 28.55803328, 19.79798303, 21.76346312, 24.39060267,
       10.72390653, 35.69760879,  8.87515321, 19.7342389 , 30.44923304,
       23.5731452 , 25.969039  , 21.53759864, 26.56759106, 29.08985079,
       27.11618087, 27.52241607, 31.13762774, 25.19009251,  1.294237  ,
       24.63082225, 20.35958514, 25.20780706, 25.3136586 , 19.78586427,
       20.81458041, 26.18578331, 20.65389528, 23.63766687, 22.07598695,
       24.48709112, 20.92346851, 21.99207783, 11.28215423, 15.85225507,
       22.37860011, 20.11691913, 17.8652717 , 27.92655648, 20.66334411,
       16.27505897, 21.25931319, 25.72062784, 25.92038934, 22.43890052,
       18.21105925, 13.73609426, 19.70671479, 19.16823222, 20.81634433,
       28.98351303, 31.19474714, 20.47647933, 14.48614909, 18.32476473,
       25.90552665, 15.20476004, 22.61771889, 24.11123534, 30.06699424,
       27.18076864, 10.97444888, 29.31613765, 29.11544566, 18.4851049 ,
       25.5516343 , 26.58527747, 29.79753531,  7.90378337, 13.9914479 ,
       27.53958439, 15.07581206, 19.1010104 , 20.2136539 ,  9.56591918,
       12.35215313, 32.65105454, 27.04505972, 22.60389757, 23.38431025,
       18.14439345, 21.46916879, 33.35149732, 21.68370567, 11.686795  ,
       22.46129098, 24.462044  , 10.63445232, 17.62333713, 25.46239518,
       13.61074004, 15.25814185, 19.3752538 , 20.11939071, 29.36108994,
       22.66082095, 25.30493792])
```

In [99]:
```
## Relationship between real data and predicted data
plt.scatter(y_test,lasso_pred)
plt.xlabel('Test Truth Data')
plt.ylabel('Test Predicted Data')
```

Out[99]:
```
Text(0, 0.5, 'Test Predicted Data')
```



In [100…

```python
# Calculating residual
residuals=y_test - lasso_pred
```

In [101…

```python
residuals
```

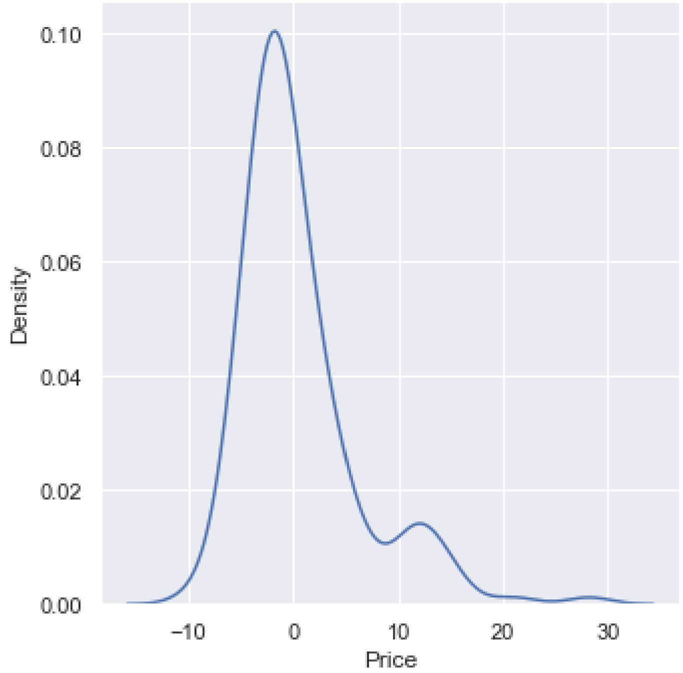
Out[101]:

```
305    2.758056
193    1.285747
65    -4.443243
349   -0.952565
151   -1.396403
         ...
442   -0.975254
451   -4.919391
188    0.438910
76    -2.660821
314   -1.504938
Name: Price, Length: 167, dtype: float64
```

In [102…

```python
# Distribution of residual are approximately Normal Distribution
sns.displot(residuals,kind="kde")
```

Out[102]: <seaborn.axisgrid.FacetGrid at 0x22c900a7c10>



Observation

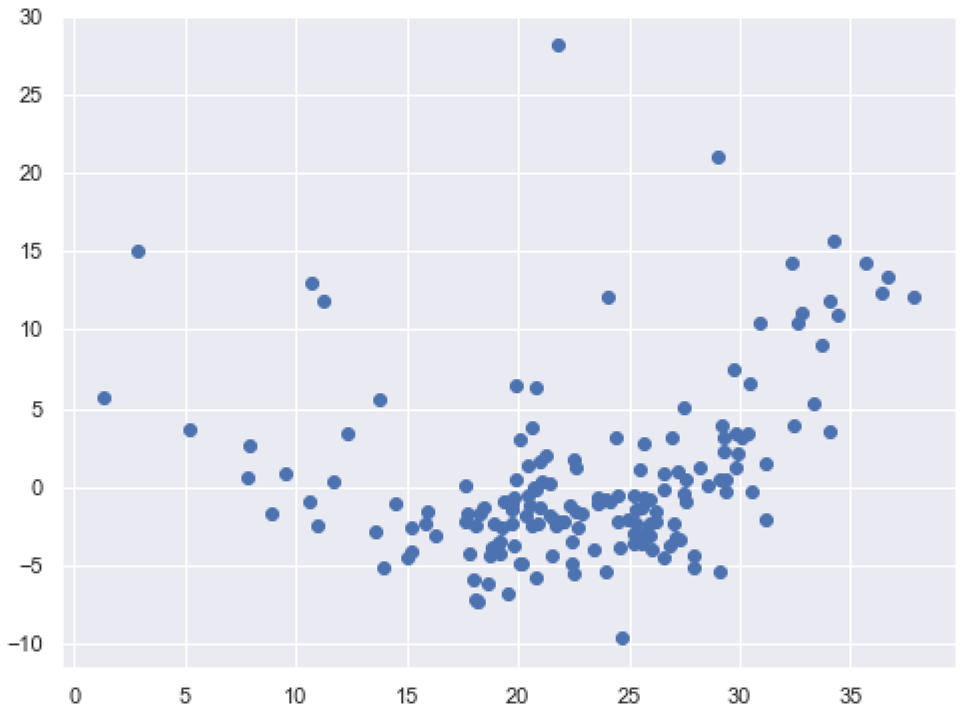
- Distribution is right skewed.

In [103…

```
# Scatterplot with prediction and residual
# Uniform Distribution
plt.scatter(lasso_pred,residuals)
```

Out[103]:

```
<matplotlib.collections.PathCollection at 0x22c90197460>
```



In [104…

```
## Performance Metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,lasso_pred))
```

```
print(mean_absolute_error(y_test,lasso_pred))
print(np.sqrt(mean_squared_error(y_test,lasso_pred)))
```

```
32.16822537607396
3.9064325476573205
5.67170392175702
```

In [105…

```
# R Squared
from sklearn.metrics import r2_score
score=r2_score(y_test,lasso_pred)
print(score)
```

```
0.6635183597615237
```

In [106…

```
# Adjusted R Squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[106]:

```
0.6349284164732871
```

In [75]:

```
# Elastic net regression
from sklearn.linear_model import ElasticNet
elastic=ElasticNet()
```

In [76]:

```
# fitting the data
elastic.fit(X_train,y_train)
```

Out[76]:

```
▾ ElasticNet
ElasticNet()
```

In [107…

```
print(elastic.coef_)
```

```
[-0.43490082  0.32641408 -0.2449198   0.19136153 -0.15045186  2.09765048
 -0.         -0.         -0.         -0.30010971 -1.13949057  0.64784661
 -2.30243571]
```

In [108…

```
print(elastic.intercept_)
```

```
22.077286135693214
```

In [109…

```
elastic_pred=elastic.predict(X_test)
```

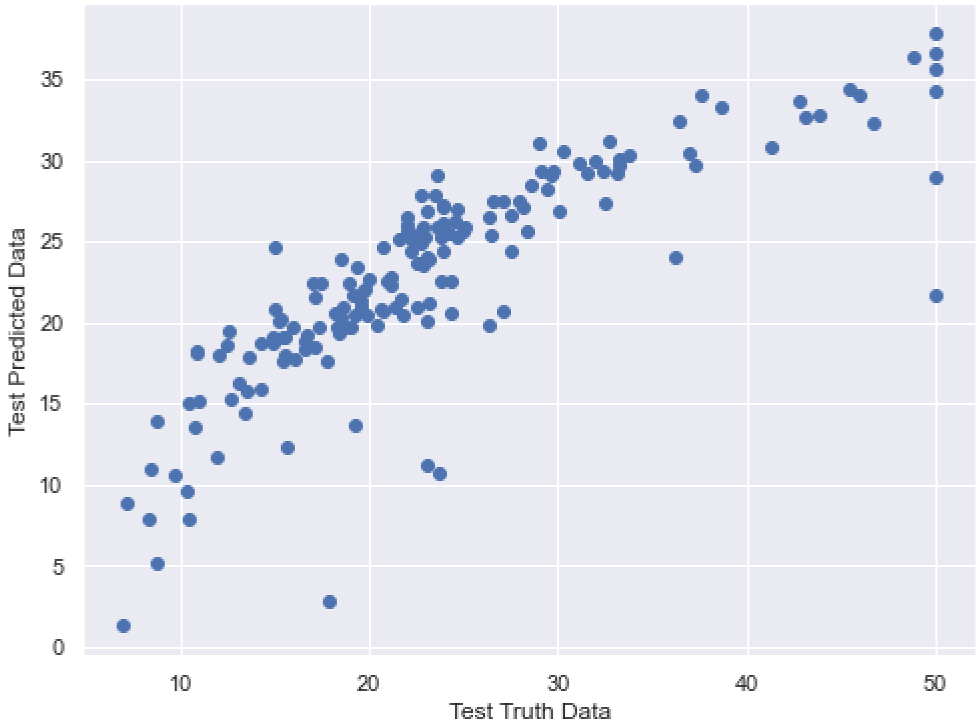
In [110…

```
elastic_pred
```

Out[110]:
```
array([26.0417533 , 29.72847396, 28.13249256, 27.33126697, 20.42880538,
       17.74088482, 31.34694254, 16.67485774, 22.66361605, 32.11606238,
       20.44062928, 27.05265082, 24.30388496, 29.10453835, 29.42032134,
       34.87404662, 25.31690008, 21.08018038, 24.04009667, 22.78241695,
       28.62957505, 18.35172223, 23.50225053, 24.94025282, 31.31440303,
       21.87551246, 22.30554751, 18.38033279, 33.5961939 ,  5.07350586,
       31.03524275, 28.19235387, 27.2862085 , 24.92462838, 19.28719449,
       20.2043877 ,  9.65913955, 29.64752478, 24.48773946, 25.34376165,
       30.68019641, 26.22751049, 18.01125345, 29.21052894, 20.61959202,
       27.27830384, 19.56149084, 19.72195809, 33.16071763, 19.16416141,
       23.05862027, 18.66118548, 22.77766754, 31.26962741, 25.0249516 ,
       29.94893114, 20.8407824 , 19.87498778, 18.27542547, 22.76517295,
       20.81723461, 22.76805785, 34.51940602, 36.11020157, 27.24493161,
       18.27047552, 24.17101249,  7.04406772, 26.85508847, 32.20329184,
       18.80351309, 27.46659498, 19.30788561, 20.8108208 , 23.90532445,
       14.22176442, 33.73545716, 10.78055539, 20.93445818, 29.78172162,
       23.99677889, 25.93581443, 22.61951728, 26.7428785 , 28.14408623,
       25.89892069, 26.67011775, 30.84900884, 24.58079972,  2.73998551,
       24.21010745, 20.83883219, 25.05619448, 24.60834531, 21.10174986,
       22.49049602, 26.06687733, 22.19194864, 23.68670917, 22.18041458,
       25.06905312, 20.26607354, 23.08760718, 10.49569995, 17.33370695,
       22.43253387, 21.15234101, 17.76115425, 26.58899485, 21.8834536 ,
       16.2603945 , 20.64765689, 25.09688902, 26.39218729, 22.53942481,
       18.04368235, 17.14638997, 20.67683019, 18.81134056, 19.82073711,
       26.78232308, 30.65570208, 20.62008364, 17.08768963, 19.18709774,
       25.65086934, 15.16746519, 21.83976175, 23.88029317, 29.16441724,
       27.25789963, 10.94687118, 29.43663882, 28.05096936, 18.27555631,
       24.83638128, 26.62936138, 30.15776119,  9.24328778, 10.68301626,
       27.04760052, 14.96584806, 18.73604079, 20.41851801,  9.92593546,
       14.91131492, 31.67918212, 27.61903571, 22.57544206, 24.72549427,
       15.35714158, 22.50035819, 31.86456612, 21.15334479, 12.67869784,
       22.65361671, 24.53650868, 11.9779066 , 18.36583206, 24.65573814,
       13.97917875, 14.8116782 , 18.92727223, 19.69541499, 28.7493839 ,
       22.61167089, 24.82104593])
```

In [111…
```python
# Relationship between real data and predicted data
plt.scatter(y_test,lasso_pred)
plt.xlabel('Test Truth Data')
plt.ylabel('Test Predicted Data')
```

Out[111]:
```
Text(0, 0.5, 'Test Predicted Data')
```



In [112…

```
# Calculating residual
residuals=y_test - elastic_pred
```

In [113…

```
residuals
```

Out[113]:

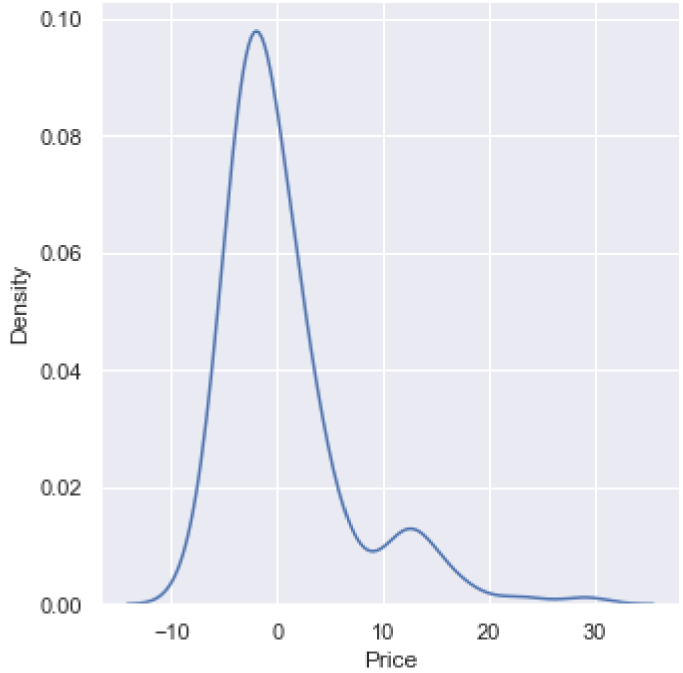
```
305     2.358247
193     1.371526
65     -4.632493
349    -0.731267
151    -0.828805
          ...
442    -0.527272
451    -4.495415
188     1.050616
76     -2.611671
314    -1.021046
Name: Price, Length: 167, dtype: float64
```

In [114…

```
# Distribution of residual are approximately Normal Distribution
sns.displot(residuals,kind="kde")
```

Out[114]:

```
<seaborn.axisgrid.FacetGrid at 0x22c900a40a0>
```



Observation

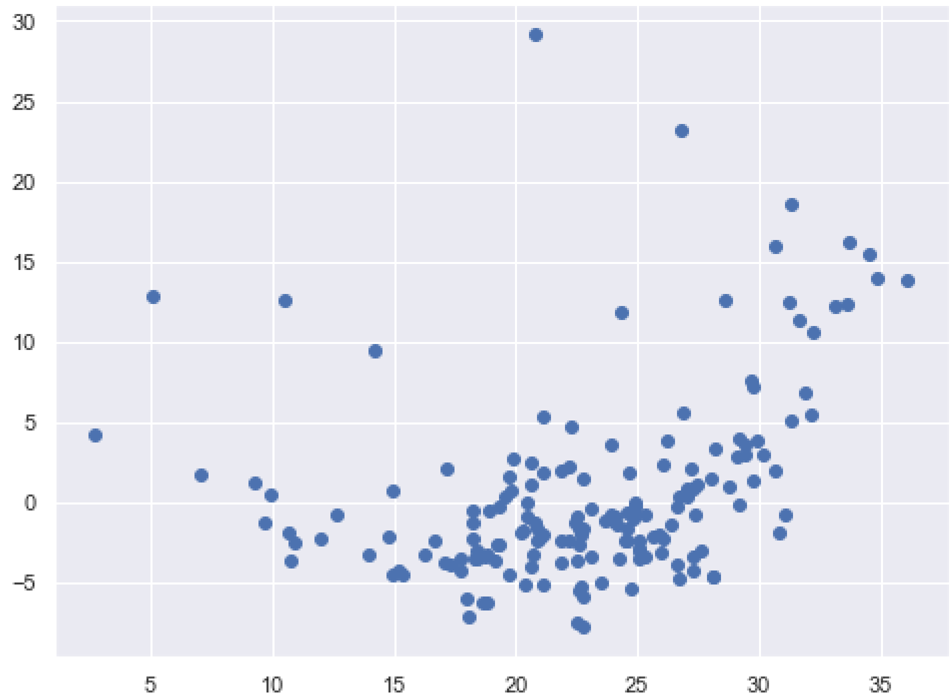
- Distribution is right skewed.

In [115…

```
# Scatterplot with prediction and residual
# Uniform Distribution
plt.scatter(elastic_pred,residuals)
```

Out[115]:

```
<matplotlib.collections.PathCollection at 0x22c9024fb80>
```



In [116…

```
## Performance Metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,elastic_pred))
print(mean_absolute_error(y_test,elastic_pred))
print(np.sqrt(mean_squared_error(y_test,elastic_pred)))
```

```
35.341543853934674
4.035696708769101
5.944875427957652
```

In [117…

```
#  R Squared
from sklearn.metrics import r2_score
score=r2_score(y_test,elastic_pred)
print(score)
```

```
0.6303252509112042
```

In [119…

```
# Adjusted R Squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[119]:

```
0.5989149781128098
```

THE END