In [1]:

```
# Importing the Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Loading the Dataset

In [2]:

```
from sklearn.datasets import load_boston
boston =load_boston()
boston.keys()
```

Out[2]:

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

Dataset Description

In [3]:

```
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 25,0
00 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds rive
r; 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 blacks
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/


This dataset was taken from the StatLib library which is maintained at Car
negie 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 diagnost
ics
...', 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 Influenti
al 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 [4]:

```
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 [5]:

```
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 [8]:

```
print(boston.feature_names)
```

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

In [9]:

```
# Prepare the dataframe
df = pd.DataFrame(boston.data,columns=boston.feature_names)
```

In [10]:

```
df.head()
```

Out[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

In [11]:

```
df['Price']=boston.target
```

In [12]:

```
df.head()
```

Out[12]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

In [18]:

```
# To check the shape of the dataset
df.shape
```

Out[18]:

```
(506, 14)
```

In [13]:

```
df.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 [14]:

```
# Statisctical information for the data
df.describe().T
```

Out[14]:

	count	mean	std	min	25%	50%	75%	
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.
Price	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.

In [15]:

```
## Checking the Missing value
df.isnull().sum()
```

Out[15]:

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
Price      0
dtype: int64
```

Exploratory Data Analysis

In [16]:

```
# Checking Correlation between its Features
df.corr()
```

Out[16]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	D
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.3796
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.6644
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.7080
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.0991
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.7692
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.2052
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.7478
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.0000
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.4945
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.5344
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.2324
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.2915
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.4969
Price	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.2499

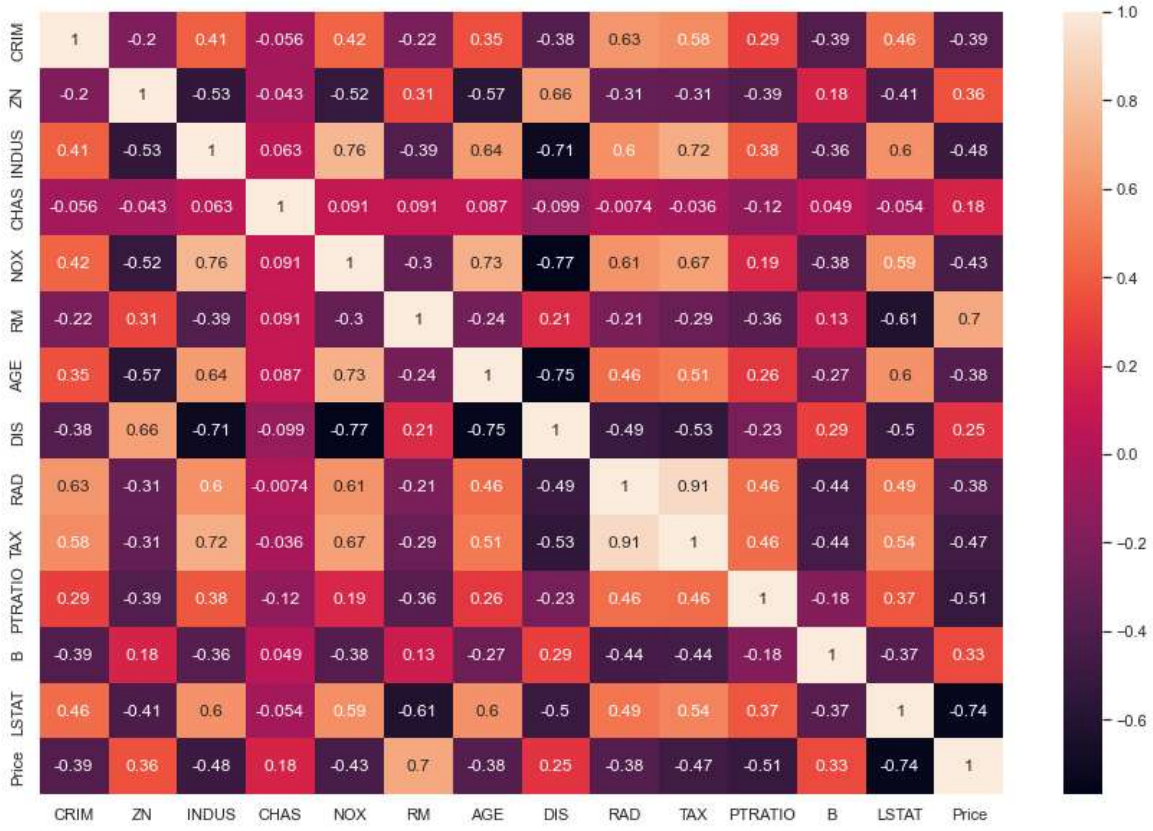
Data Visualization- Performing basic Exploratory operation related to feature

In [17]:

```
import seaborn as sns
sns.set(rc={'figure.figsize':(15,10)})
sns.heatmap(df.corr(),annot=True)
```

Out[17]:

<AxesSubplot: >

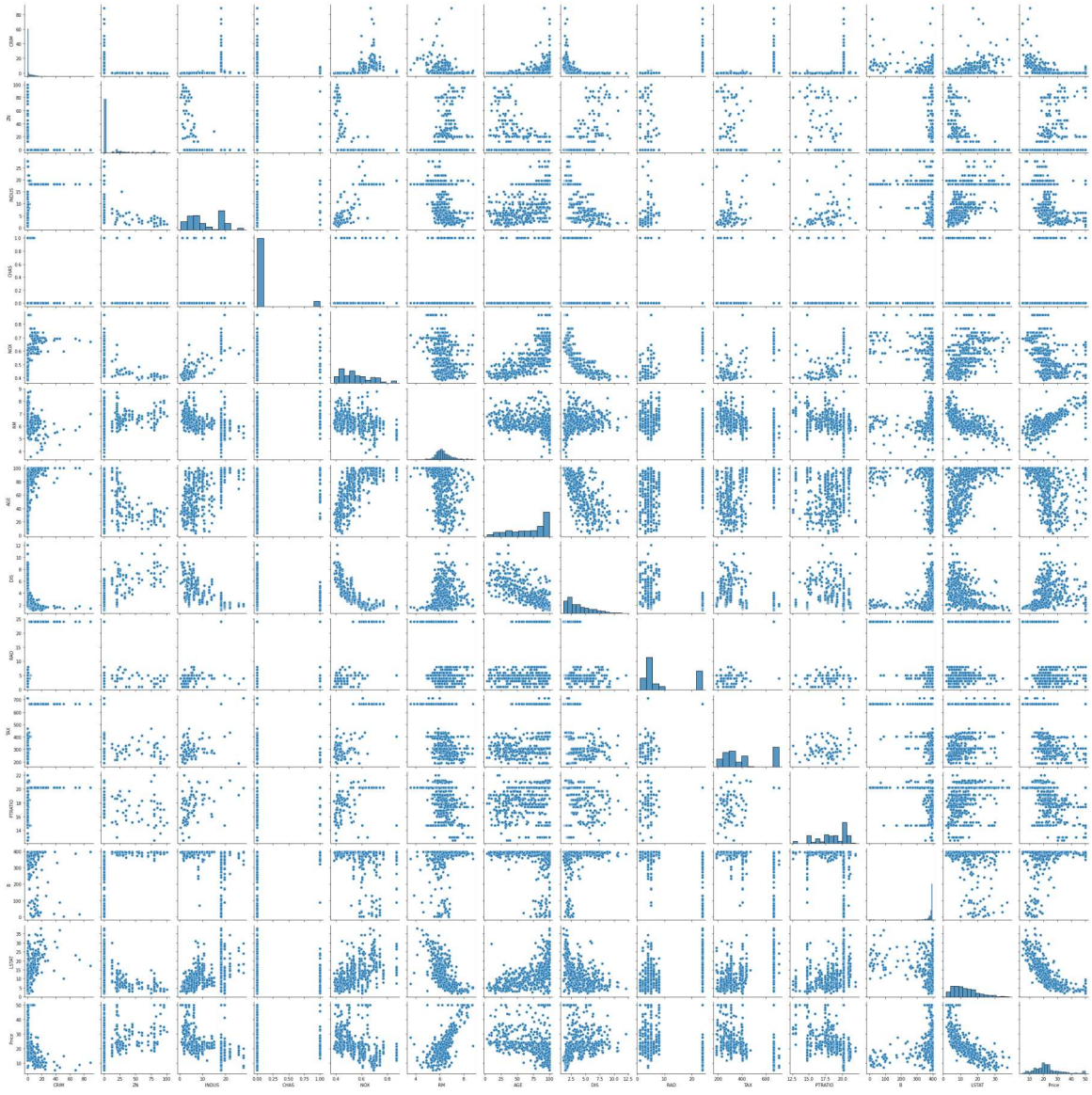


In [19]:

```
# Pairplot
sns.pairplot(df)
```

Out[19]:

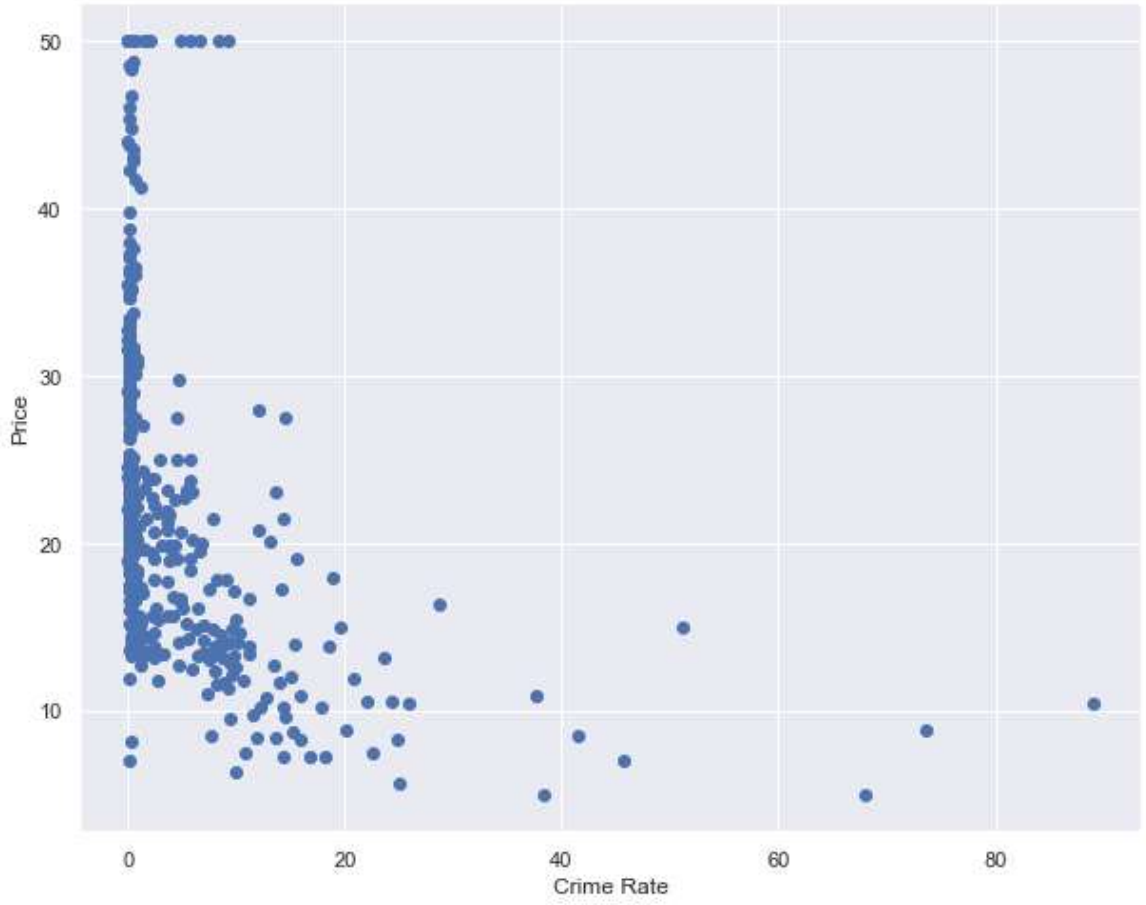
```
<seaborn.axisgrid.PairGrid at 0x1e73403bbb0>
```



Scatterplot - Crime Rate Vs Price

In [20]:

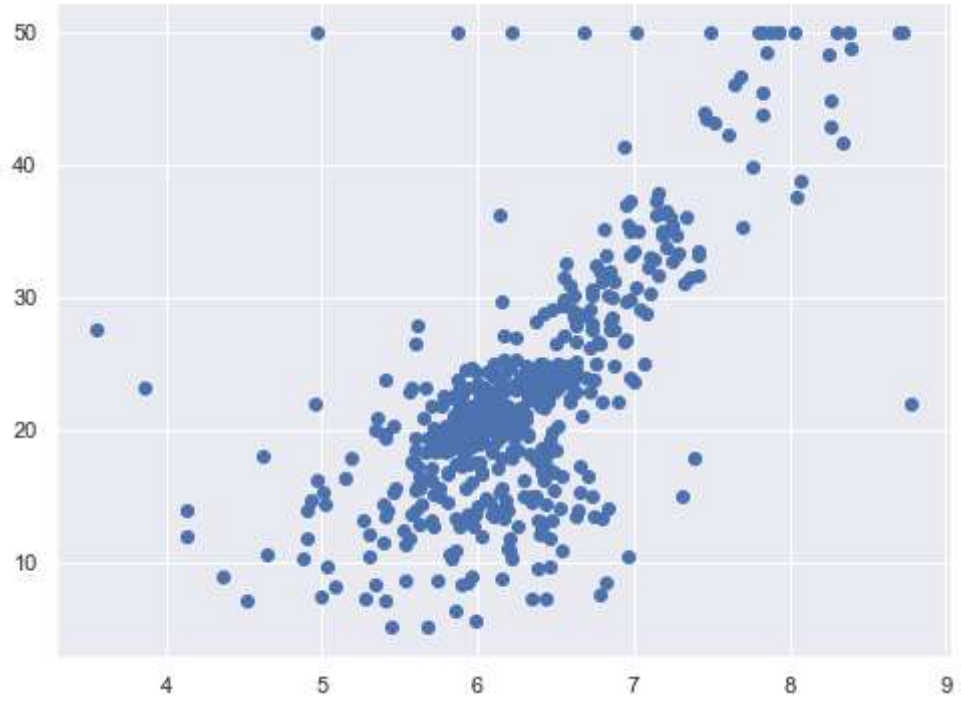
```
sns.set(rc={'figure.figsize':(10,8)})
plt.scatter(df['CRIM'],df['Price'])
plt.xlabel('Crime Rate')
plt.ylabel('Price');
```



Scatterplot - Rooms Vs Price

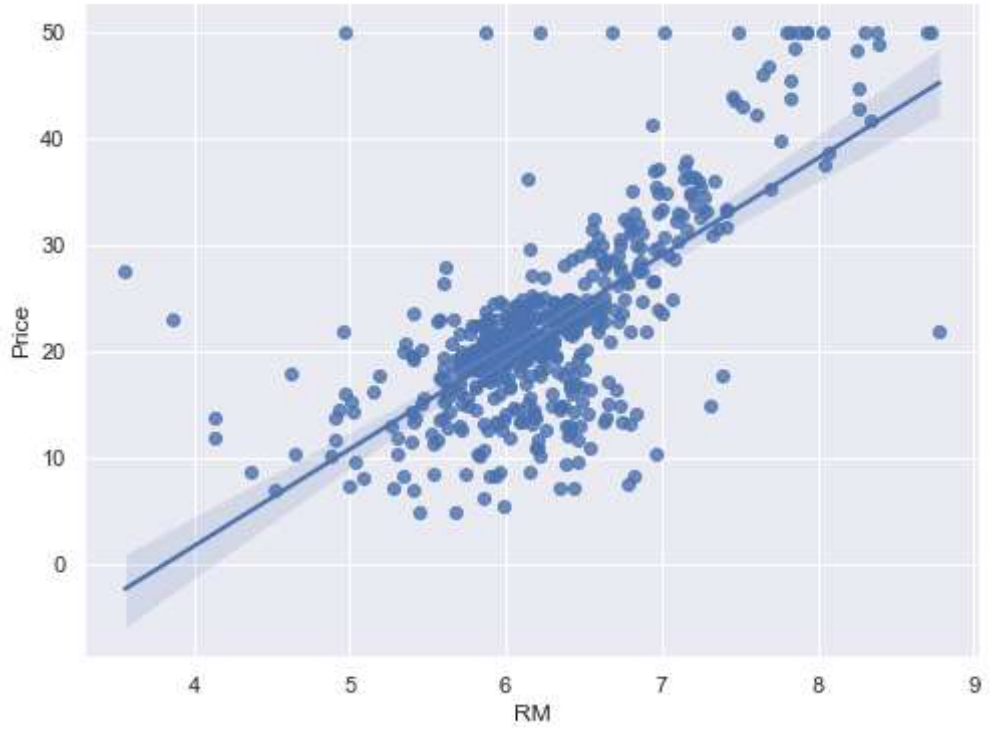
In [21]:

```
sns.set(rc={'figure.figsize':(8,6)})
plt.scatter(df['RM'],df['Price']);
```



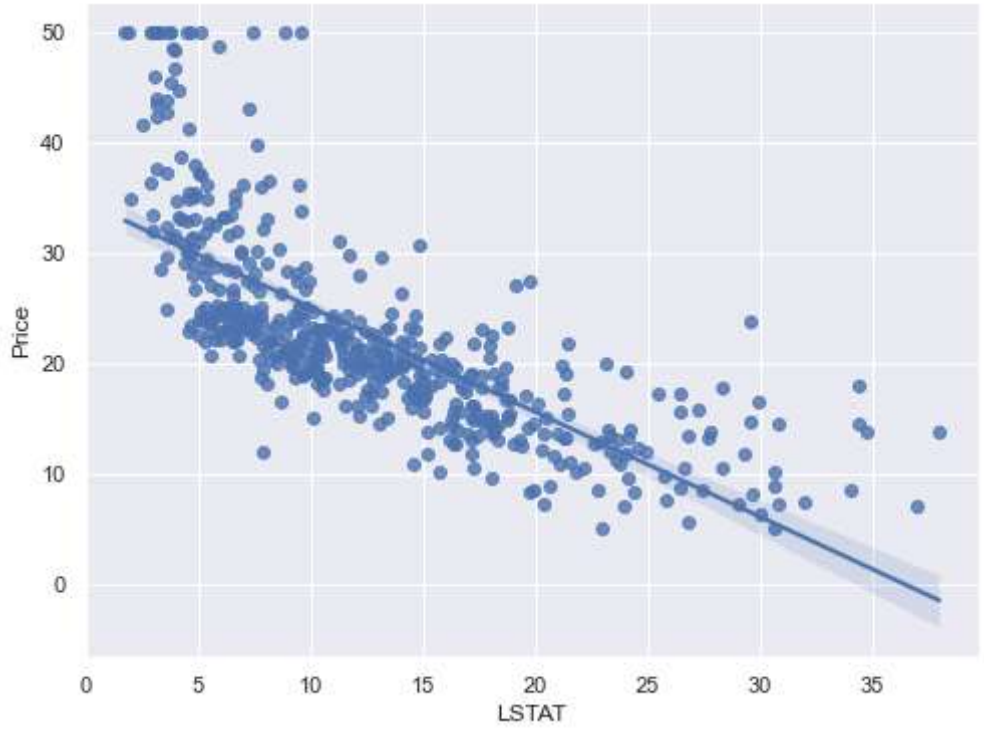
In [22]:

```
sns.regplot(x="RM",y="Price",data=df);
```



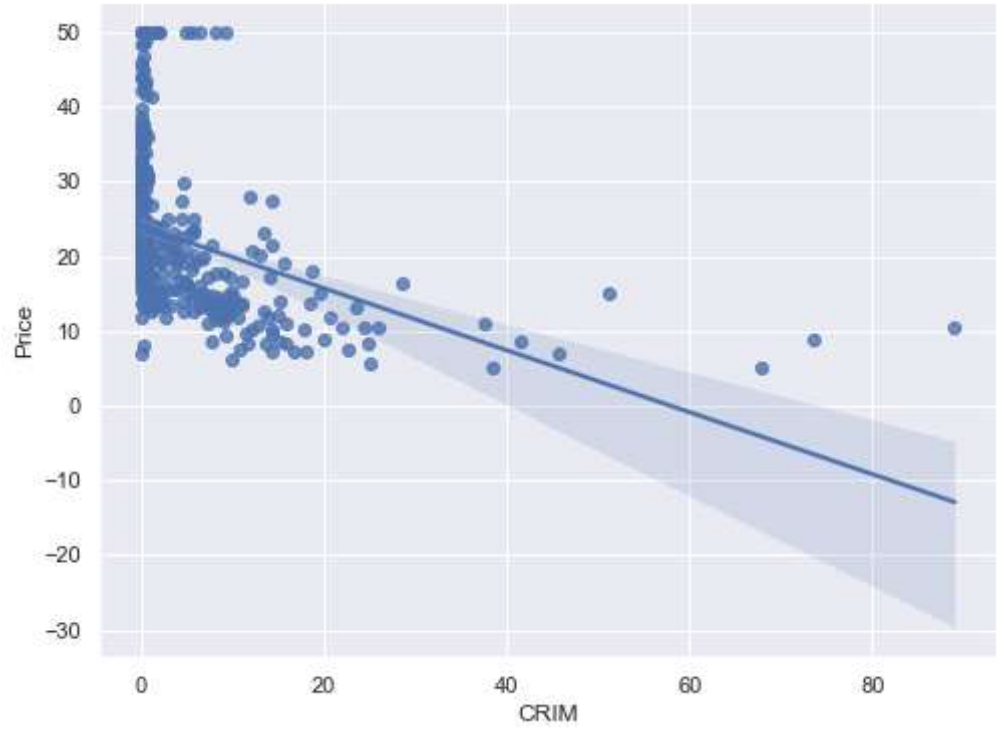
In [23]:

```
sns.regplot(x="LSTAT",y="Price",data=df);
```



In [24]:

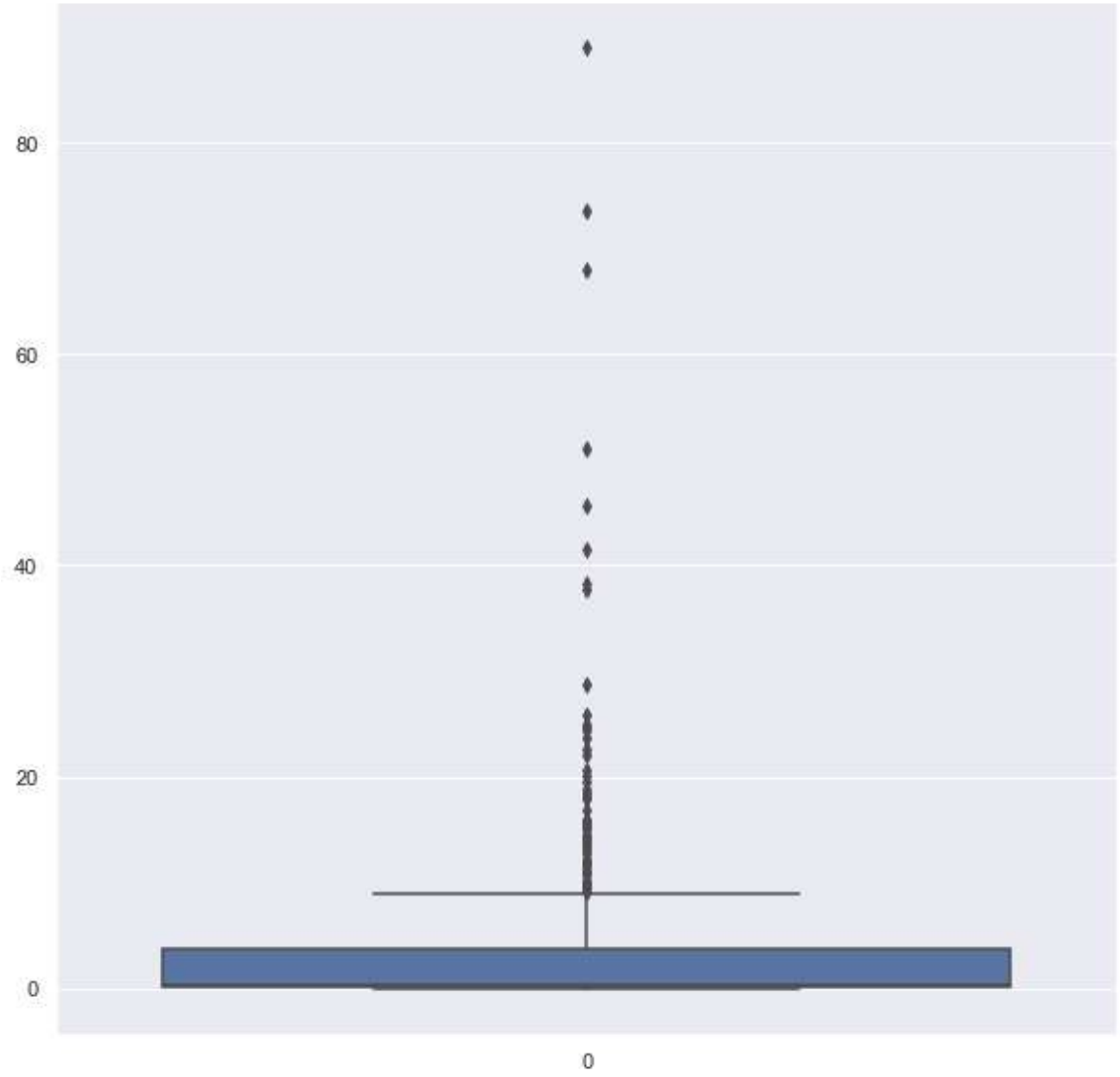
```
sns.regplot(x="CRIM",y="Price",data=df);
```



Detecting the Outliers present in the Crimal Rate & Price using Box-Plot

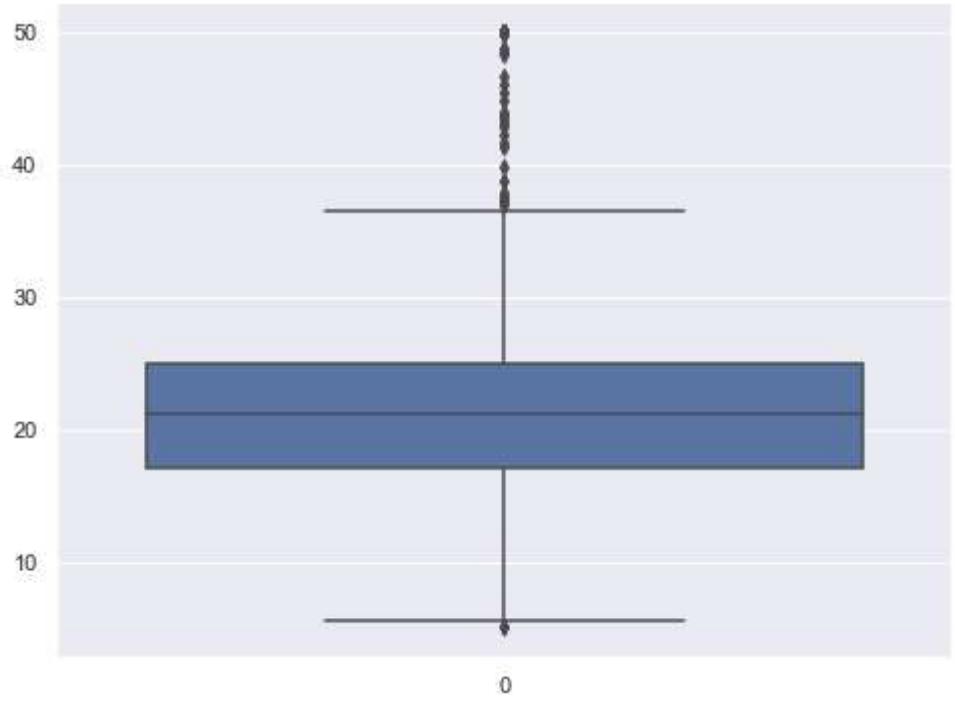
In [35]:

```
plt.figure(figsize=(10,10))
sns.boxplot(df['CRIM']);
```



In [36]:

```
sns.boxplot(df['Price']);
```



In [37]:

```
# Checking the DataFrame and first 5 rows
df.head()
```

Out[37]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

Splitting the Dataset into Features and Label

In [38]:

```
# Independent and Dependent Features

X=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

In [41]:

```
X
```

Out[41]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	E
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.9(
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.9(
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
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
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.9(
...	...	...	...	...	...	...	...	...	...	...	...	..
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
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.9(
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.9(
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
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.9(

506 rows × 13 columns

In [42]:

```
y
```

Out[42]:

```
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
```

Model Training

In [43]:

```
from sklearn.model_selection import train_test_split
```

Splitting the Dataset into Train & Test Data

In [44]:

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

In [45]:

```
X_train.shape
```

Out[45]:

```
(339, 13)
```

In [46]:

```
y_train.shape
```

Out[46]:

```
(339,)
```

In [47]:

```
X_test.shape
```

Out[47]:

```
(167, 13)
```

In [48]:

```
y_test.shape
```

Out[48]:

```
(167,)
```

Scaling the Train & Test Data Set using Standard Scaler

In [49]:

```
## Standardizied the Dataset
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [50]:

```
scaler
```

Out[50]:

```
StandardScaler()
```

In [51]:

```
X_train_scaled = scaler.fit_transform(X_train)
```

In [52]:

```
X_test_scaled = scaler.transform(X_test)
```

In [53]:

```
X_train_scaled
```

Out[53]:

```
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 [54]:

```
X_test_scaled
```

Out[54]:

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

Model Training

In [55]:

```
from sklearn.linear_model import LinearRegression
```

In [56]:

```
regression = LinearRegression()
```

In [57]:

```
regression
```

Out[57]:

```
LinearRegression()
```

In [58]:

```
regression.fit(X_train,y_train)
```

Out[58]:

```
LinearRegression()
```

In [59]:

```
## Print the coeeficent and the intercept
print(regression.coef_)
```

```
[-1.56353750e-01  6.60616548e-02 -2.01655290e-02  1.57889806e+00
 -1.53490777e+01  3.34105533e+00  1.14773954e-02 -1.55360652e+00
  3.05450100e-01 -1.21583833e-02 -8.14405729e-01  1.32763069e-02
 -5.45290138e-01]
```

In [60]:

```
print(regression.intercept_)
```

```
34.21329725760603
```

Prediction for the Test data

In [61]:

```
reg_pred=regression.predict(X_test)
```

In [62]:

```
reg_pred
```

Out[62]:

```
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])
```

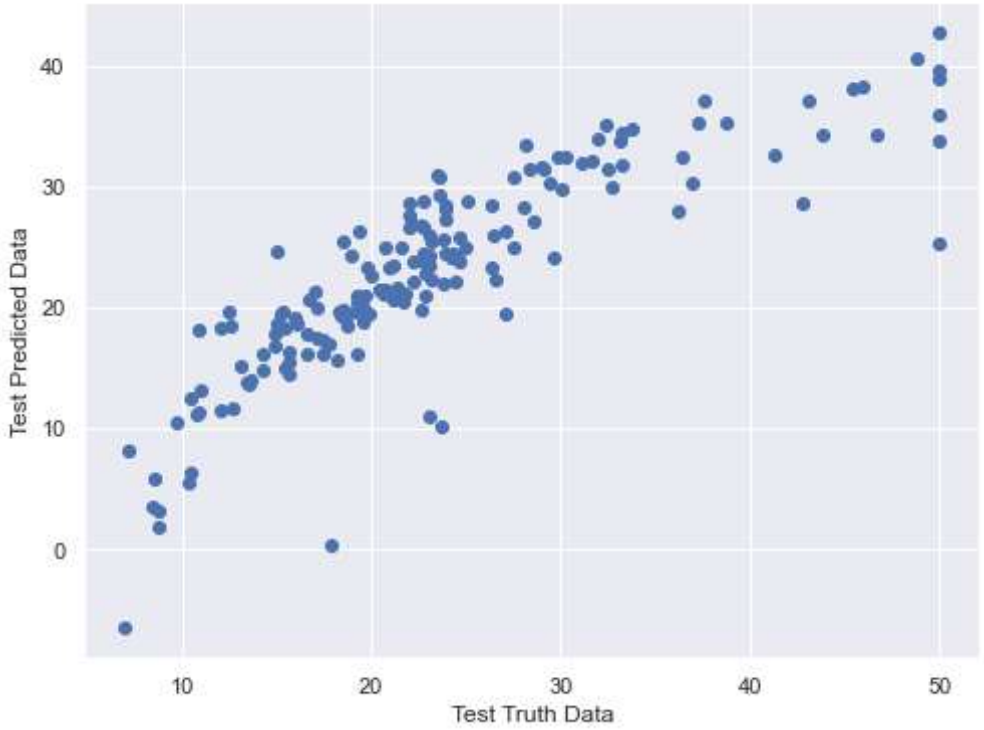
Assumptions of Linear Regression

In [63]:

```
plt.scatter(y_test,reg_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

Out[63]:

```
Text(0, 0.5, 'Test Predicted Data')
```



Residuals

In [64]:

```
residuals = y_test-reg_pred
```

In [65]:

```
residuals
```

Out[65]:

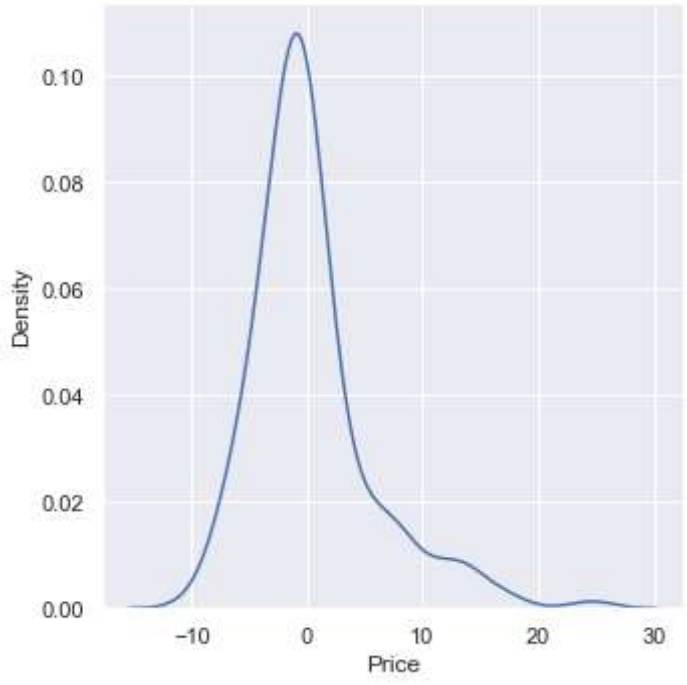
```
305   -3.038496
193   -0.887944
65    -7.498956
349    4.286033
151    0.705072
         ...
442   -1.004380
451   -4.387684
188   -2.638009
76    -2.661709
314   -1.885761
Name: Price, Length: 167, dtype: float64
```

In [66]:

```
sns.displot(residuals,kind='kde')
```

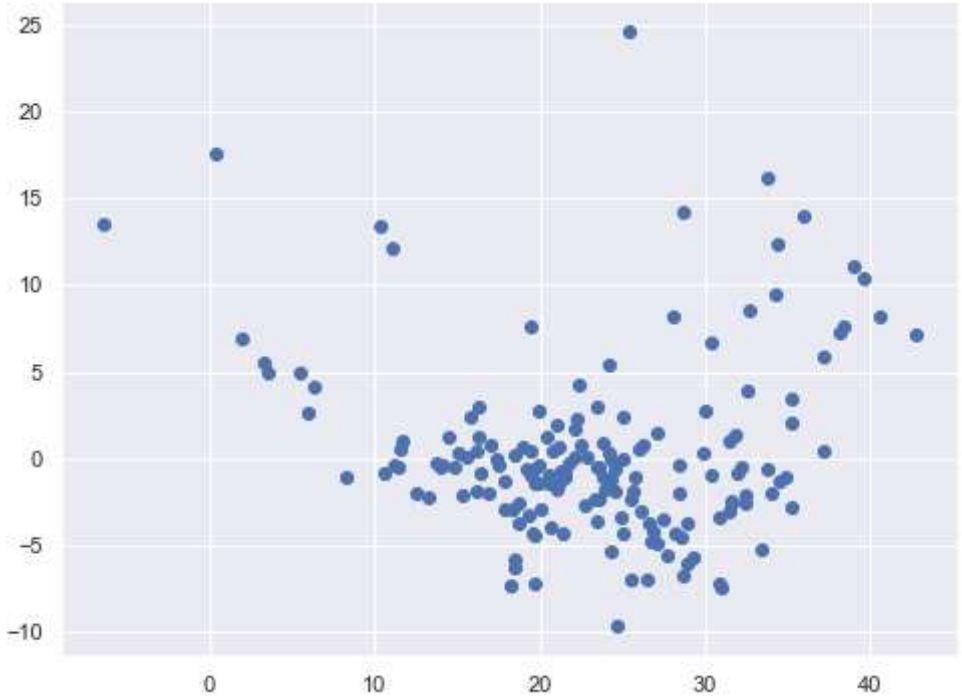
Out[66]:

```
<seaborn.axisgrid.FacetGrid at 0x1e742e3ee80>
```



In [67]:

```
## Scatter plot with predictions and residual
## Uniform distribution
plt.scatter(reg_pred,residuals);
```



Performance Matrics

In [68]:

```
from sklearn.metrics import mean_squared_error #MSE
from sklearn.metrics import mean_absolute_error #MAE

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)))
```

```
27.10099170996272
3.5206585298798116
5.205861284164487
```

R Square & Adjusted R square

In [69]:

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

```
0.716521939396753
```

In [70]:

```
## Adjusted R square

1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[70]:

```
0.6924355682343857
```

Evaluating the Dataset Using Ridge Regression

In [71]:

```
from sklearn.linear_model import Ridge
ridge = Ridge()
```

In [72]:

```
ridge
```

Out[72]:

```
Ridge()
```

In [73]:

```
ridge.fit(X_train,y_train)
```

Out[73]:

```
Ridge()
```

In [74]:

```
print(ridge.coef_)
```

```
[-1.52632064e-01  6.72194813e-02 -4.61868347e-02  1.45844821e+00
 -7.76898482e+00  3.39133018e+00  4.68041987e-03 -1.43319434e+00
  2.87697853e-01 -1.27154968e-02 -7.34594385e-01  1.36842091e-02
 -5.54979507e-01]
```

In [75]:

```
print(ridge.intercept_)
```

```
28.878811583899946
```

In [76]:

```
ridge_pred = ridge.predict(X_test)
```

In [77]:

```
ridge_pred
```

Out[77]:

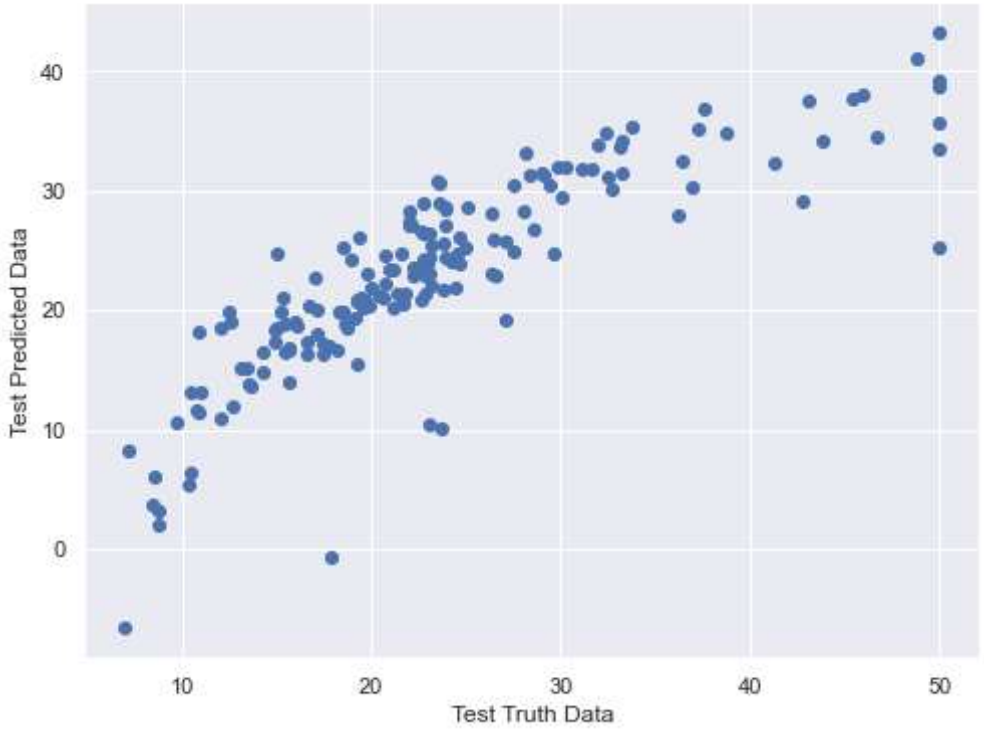
```
array([31.30151302, 31.90126709, 30.82061573, 22.88812126, 20.31302387,
       16.53738624, 35.72812342, 14.8816675 , 24.63531595, 36.860872  ,
       21.24748513, 30.46713945, 27.88750209, 33.88238471, 33.62448674,
       41.10851559, 24.75396381, 23.02945622, 25.51134988, 22.69511097,
       32.34032533, 18.39610936, 25.19196888, 25.25847581, 32.47842797,
       20.99995617, 19.2369491 , 17.40718641, 38.00318156, -0.61359473,
       32.04483047, 31.84921686, 26.43869186, 23.96961375, 20.30489338,
       19.93571226,  3.79628876, 35.24712422, 27.18060759, 27.45307204,
       34.54591726, 29.52160214, 18.50398038, 31.41198471, 17.43190724,
       28.64573998, 20.33141004, 21.39198549, 37.67651978, 16.60438885,
       24.20570719, 19.80900064, 24.26312305, 34.20633563, 27.07201815,
       35.29920299, 20.89256765, 20.83539778, 18.77362424, 24.77911964,
       20.14183389, 23.37554369, 39.23285572, 43.28676132, 30.45714887,
       17.07059688, 23.63831031,  3.2257982 , 31.19529737, 29.2191736 ,
       19.07024992, 26.74560227, 19.32823778, 25.20390851, 25.00399356,
       10.14495703, 38.69984714,  8.27960583, 18.53194663, 30.27261436,
       22.83546957, 21.45322051, 20.0093348 , 28.32764591, 30.62195844,
       28.53535251, 25.73580192, 31.53955632, 22.90414042, -6.6242741 ,
       22.28964733, 19.83165899, 24.71186029, 23.8500573 , 19.05308704,
       18.56224278, 27.08227103, 21.84161734, 26.54715488, 23.14882612,
       23.64448642, 18.91127471, 20.94359851, 10.47836918, 13.74582082,
       20.16243679, 23.05737754, 13.58013444, 28.93337707, 16.60270565,
       15.20197879, 22.03277272, 26.40083872, 28.57075083, 24.18673827,
       18.21939098, 15.54580923, 17.11763407, 16.8511474 , 21.08776724,
       33.44724064, 30.07125145, 21.46975178, 15.11884516, 16.40552026,
       28.95082498, 13.21512515, 21.71702584, 24.4563997 , 31.55015189,
       33.24347968,  6.08415598, 34.94161154, 24.69756072, 18.0754461 ,
       24.06272989, 28.18826271, 34.2155616 ,  6.35685482,  2.00959625,
       28.36484928, 13.10018643, 18.82610551, 21.08244808,  5.40736266,
       14.02778007, 37.60352183, 26.13219721, 23.46037424, 26.08457696,
       11.54741208, 20.55527427, 34.91876361, 20.72969929, 10.96716828,
       16.38493095, 24.44222592, 10.61017709, 16.45530346, 26.02233123,
       11.61373636, 11.96109835, 19.91938337, 19.9460085 , 32.07539476,
       21.82189814, 25.68879093])
```

In [78]:

```
plt.scatter(y_test,ridge_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

Out[78]:

```
Text(0, 0.5, 'Test Predicted Data')
```



Residuals

In [79]:

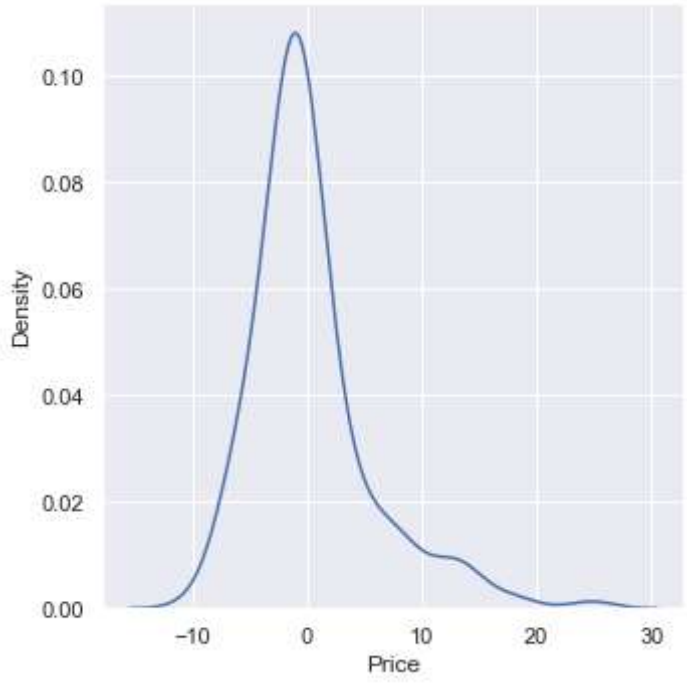
```
ridge_residuals = y_test - ridge_pred
ridge_residuals
```

Out[79]:

```
305   -2.901513
193   -0.801267
65    -7.320616
349    3.711879
151   -0.713024
         ...
442   -1.519383
451   -4.746009
188   -2.275395
76    -1.821898
314   -1.888791
Name: Price, Length: 167, dtype: float64
```

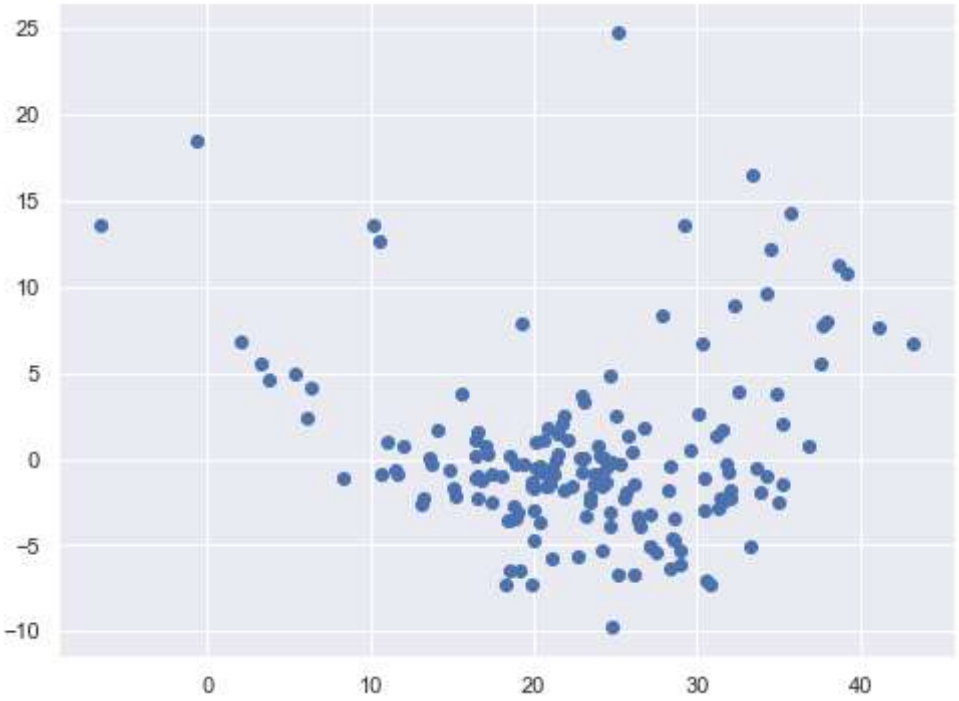
In [80]:

```
sns.displot(ridge_residuals,kind="kde");
```



In [81]:

```
plt.scatter(ridge_pred,ridge_residuals);
```



In [82]:

```
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)))
```

```
27.63979628041237
3.566042008593924
5.257356396556388
```

R square & Adjusted Rsquare

In [83]:

```
from sklearn.metrics import r2_score
ridge_score=r2_score(y_test,ridge_pred)
print(ridge_score)
```

```
0.7108860100436925
```

In [84]:

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

Out[84]:

```
0.6863207690670128
```

Evaluating the dataset using Lasso Regression

In [85]:

```
from sklearn.linear_model import Lasso
lasso = Lasso()
```

In [86]:

```
lasso
```

Out[86]:

```
Lasso()
```

In [87]:

```
lasso.fit(X_train,y_train)
```

Out[87]:

```
Lasso()
```

In [88]:

```
print(lasso.intercept_)
```

```
41.983204728562384
```

In [89]:

```
print(lasso.coef_)
```

```
[-0.10475563  0.07649297 -0.01718277  0.         -0.          0.24993515
  0.03023573 -0.80122944  0.26755925 -0.01537057 -0.58410577  0.01011944
 -0.76343344]
```

In [90]:

```
lasso_pred=lasso.predict(X_test)
```

In [91]:

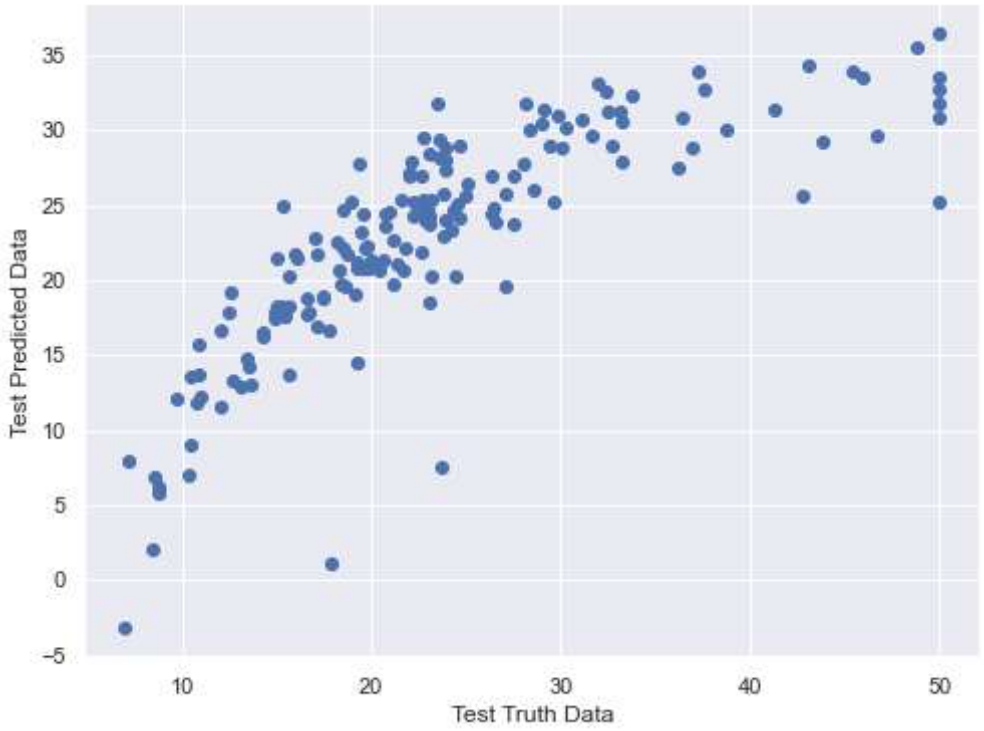
```
lasso_pred
```

Out[91]:

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

In [92]:

```
plt.scatter(y_test,lasso_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data");
```



Residuals

In [93]:

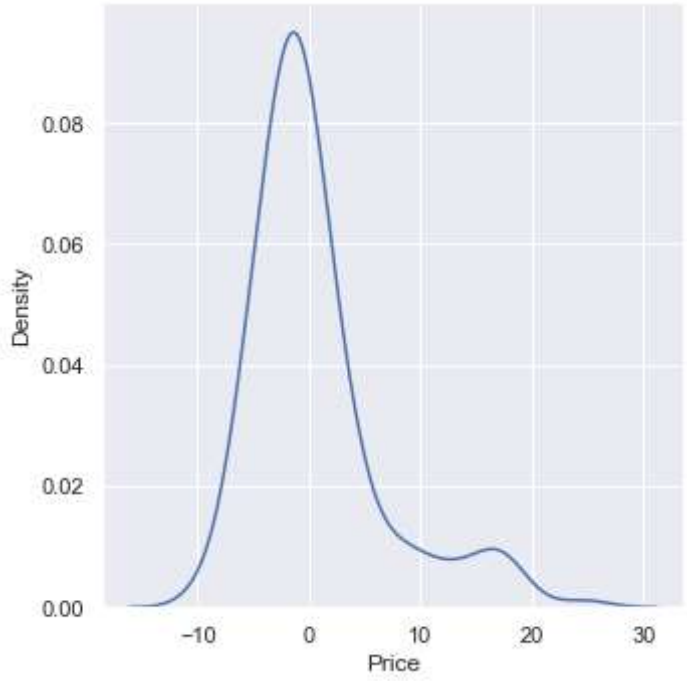
```
lasso_residuals = y_test-lasso_pred
lasso_residuals
```

Out[93]:

```
305   -1.650622
193    0.341366
65    -8.291364
349    2.788674
151   -4.864509
         ...
442   -1.374711
451   -3.092962
188   -1.147333
76    -1.267739
314   -1.938590
Name: Price, Length: 167, dtype: float64
```

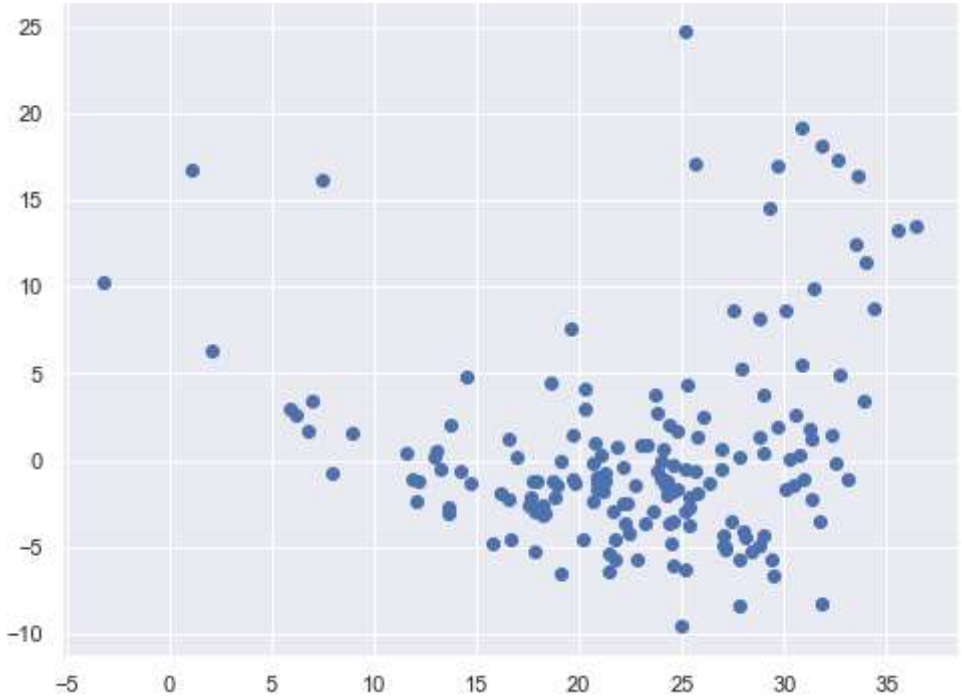
In [94]:

```
sns.displot(lasso_residuals,kind="kde");
```



In [96]:

```
## Scatter plot with predictions and residual
##Uniform distribution
plt.scatter(lasso_pred,lasso_residuals);
```



In [97]:

```
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)))
```

```
36.11332777156728
4.067062767559684
6.009436560241507
```

R Square & Adjusted R squared

In [98]:

```
from sklearn.metrics import r2_score
lasso_score=r2_score(y_test,lasso_pred)
print(lasso_score)
```

```
0.6222523430812359
```

In [99]:

```
## Adjusted R square
#display adjusted R-squared
1 - (1-lasso_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[99]:

```
0.5901561369378114
```

Evaluating the dataset with Elastic Net Regression

In [100]:

```
from sklearn.linear_model import ElasticNet
elasticnet = ElasticNet()
```

In [101]:

```
elasticnet
```

Out[101]:

```
ElasticNet()
```

In [102]:

```
elasticnet.fit(X_train,y_train)
```

Out[102]:

```
ElasticNet()
```

In [103]:

```
print(elasticnet.intercept_)
```

```
39.51456077615768
```

In [104]:

```
print(elasticnet.coef_)
```

```
[-0.12157848  0.07459975 -0.03899046  0.         -0.          0.70123205
  0.02893353 -0.84498184  0.28961626 -0.01546206 -0.61704509  0.01082719
 -0.73127687]
```

In [105]:

```
elasticnet_pred=elasticnet.predict(X_test)
```

In [106]:

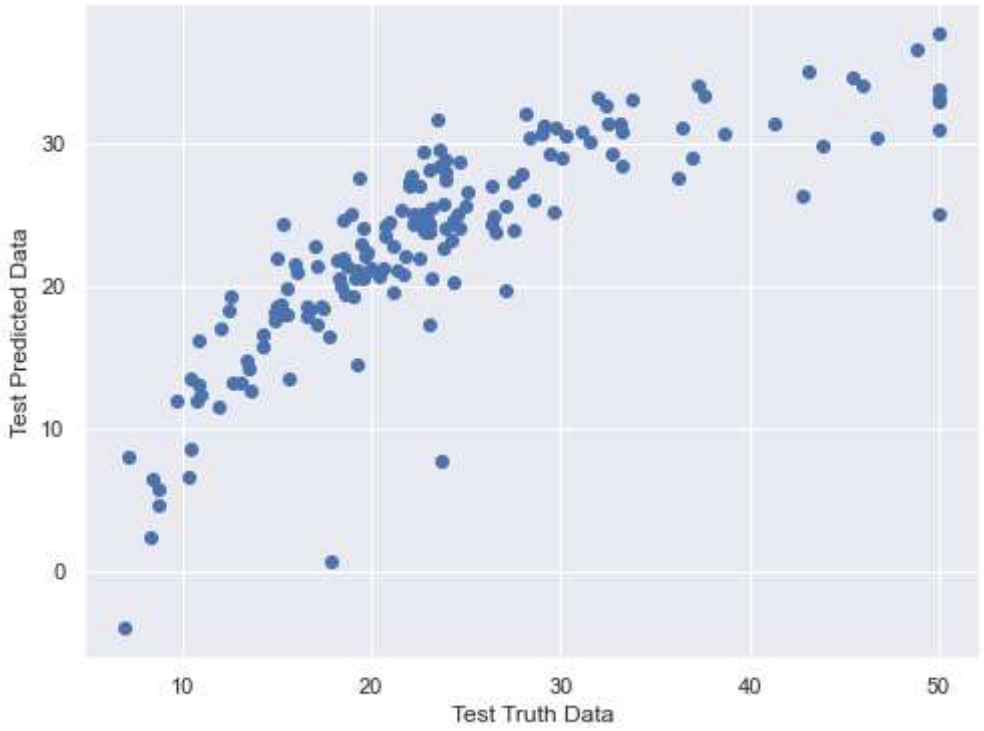
```
elasticnet_pred
```

Out[106]:

```
array([30.32727993, 30.82005104, 31.58152723, 23.74897548, 24.04066339,
       16.64208335, 33.72175392, 15.7796708 , 23.54360404, 33.38864033,
       20.64135992, 27.30279352, 27.61114038, 33.16496835, 31.30632428,
       36.60242063, 25.02604143, 24.32320649, 25.42409477, 22.74163791,
       31.42753395, 18.15172597, 24.66377356, 25.5930192 , 31.0902106 ,
       22.99406956, 19.75673457, 17.62225051, 34.05998978,  0.71393175,
       30.54158704, 30.05134167, 28.13413609, 23.98640017, 18.3419393 ,
       20.59655372,  2.36980673, 33.98725736, 27.65073017, 27.07927693,
       30.32046718, 28.98490321, 17.05004568, 31.23586807, 17.83741534,
       28.86472005, 21.03042701, 21.08354432, 34.61346048, 18.06846239,
       25.10524461, 18.26828696, 24.68020774, 29.84488335, 26.99724854,
       33.00363459, 21.04127003, 21.96291557, 20.9156221 , 21.93093756,
       20.51191466, 22.75831806, 33.25356336, 37.69317773, 29.21343037,
       16.44883562, 23.88399261,  5.73678016, 31.41397208, 26.2387969 ,
       19.29642893, 26.03173494, 19.20268502, 25.02297417, 23.84443518,
        7.68082768, 32.86644065,  8.05404899, 21.42877667, 29.02221506,
       24.02748648, 23.72705434, 21.43227946, 27.30244657, 29.59030615,
       28.01870181, 25.66336436, 30.59211532, 24.96621901, -3.98999389,
       24.18939757, 21.92147658, 25.29138706, 24.59227886, 21.58691263,
       18.50577433, 27.393029  , 20.27186886, 26.99600445, 22.37968176,
       24.27900606, 19.42893094, 22.0778287 , 17.35434124, 14.17049383,
       19.60954073, 23.7971747 , 12.6676568 , 29.38676612, 21.73985069,
       13.18186575, 20.55801528, 24.84735541, 26.63076152, 25.07411166,
       16.12276859, 14.45382955, 18.57745942, 19.85077271, 21.25627948,
       31.00768345, 29.21525794, 22.02696279, 14.74351416, 18.55224908,
       28.34160315, 12.37902394, 22.62806951, 24.32172973, 28.42443954,
       32.06470207,  6.48450097, 32.57411341, 25.20463016, 17.27912488,
       23.14124551, 27.06784434, 30.86575934,  8.5643464 ,  4.70402049,
       27.82665403, 13.5627324 , 17.9576227 , 24.31428469,  6.64669939,
       13.52691815, 35.07905289, 28.71229522, 24.42196943, 27.53944472,
       13.05632256, 20.78080524, 30.71452488, 20.60297639, 11.49834979,
       18.48955421, 24.06063117, 11.95041588, 18.07009953, 24.8600991 ,
       11.94641703, 13.19510438, 20.02129294, 18.73517709, 31.02814718,
       21.18425857, 25.77300816])
```

In [108]:

```
plt.scatter(y_test,elasticnet_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data");
```



Residuals

In [109]:

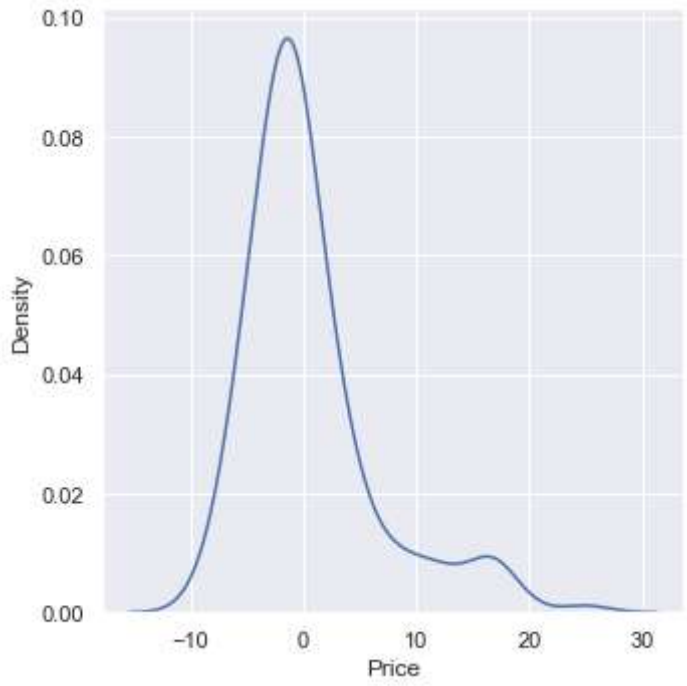
```
elasticnet_residuals=y_test-elasticnet_pred
elasticnet_residuals
```

Out[109]:

```
305   -1.927280
193    0.279949
65    -8.081527
349    2.851025
151   -4.440663
         ...
442   -1.621293
451   -3.535177
188   -1.228147
76    -1.184259
314   -1.973008
Name: Price, Length: 167, dtype: float64
```

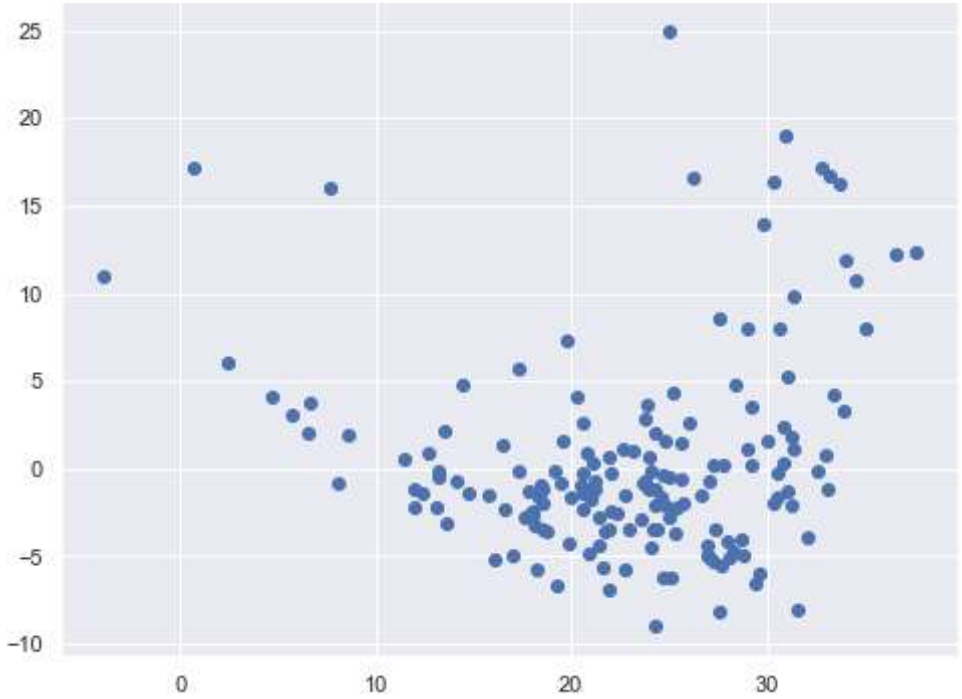
In [110]:

```
sns.displot(elasticnet_residuals,kind="kde");
```



In [111]:

```
## Scatter plot with predictions and residual
##Uniform distribution
plt.scatter(elasticnet_pred,elasticnet_residuals);
```



In [112]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,elasticnet_pred))
print(mean_absolute_error(y_test,elasticnet_pred))
print(np.sqrt(mean_squared_error(y_test,elasticnet_pred)))
```

```
34.65454968329032
4.002630706596711
5.886811503971426
```

Rsquare & Adjusted R square

In [113]:

```
from sklearn.metrics import r2_score
elasticnet_score=r2_score(y_test,elasticnet_pred)
print(elasticnet_score)
```

```
0.6375112527086368
```

In [114]:

```
## Adjusted R square
#display adjusted R-squared
1 - (1-elasticnet_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[114]:

```
0.6067115552263642
```

Implemented Linear, Lasso, Ridge & Elastic-Net Regresion on Boston Housing Dataset.