Boston House Data Prediction using popular Regression Algorithms

Importing necessary Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
import seaborn as sns
%matplotlib inline
```

Loading the Dataset

In [2]:

```
data = load_boston()
data.keys()
```

```
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\uti
ls\deprecation.py:87: FutureWarning: Function load_boston is deprecated; `load_boston` is
deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np

        data_url = "http://lib.stat.cmu.edu/datasets/boston"
```

```
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.
  warnings.warn(msg, category=FutureWarning)
```

Out[2]:

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

In [3]:

```python
print(data.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/


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

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 re
```

gression
problems.

```
.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sou
rces 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 Massach
usetts, Amherst. Morgan Kaufmann.
```

In [4]:

```
print(data.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(data.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 [6]:

```
boston_features = data.feature_names
print(boston_features)
```

['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'

```
 'B' 'LSTAT']
```

In [7]:

```python
# Preparing the Dataset

boston_df = pd.DataFrame(data.data,columns=boston_features)
boston_df.head()
```

Out[7]:

	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

Here we can see that the target variable is not added, so we'll add the target column in the below steps

In [8]:

```python
boston_df['PRICE']=data.target
```

Checking the newly added column in the dataframe

In [9]:

```python
boston_df.head()
```

Out[9]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
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	24.0
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	21.6
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	34.7
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	33.4
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	36.2

In [10]:

```python
boston_df.describe(include='all')
```

Out[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TA
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.2371
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.5371
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.0000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.0000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.0000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.0000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.0000

In [11]:

```
boston_df.describe()
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TA
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.2371
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.5371
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.0000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.0000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.0000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.0000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.0000

Checking the Transpose of the overall description of the dataset

In [12]:

```
boston_df.describe().T
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.9762
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.0000
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.7400
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.0000
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.8710
...	...	...	...	...	...	...	...	...
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.0000
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.0000
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.9000
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.9700
PRICE	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.0000

Checking the data type of each column variable

In [13]:

```
boston_df.dtypes
```

Out[13]:

```
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
```

```
PTRATIO    float64
B          float64
LSTAT      float64
PRICE      float64
dtype: object
```

In [14]:

```
boston_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
```

Checking whether any null value is present in the dataset

In [15]:

```
boston_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
```

Finding the correlation of the datapoints in the dataset

In [16]:

```
boston_df.corr()
```

Out[16]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.625505	0.582764	0.289946	0.38
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.311948	-0.314563	-0.391679	0.17

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.595129	0.720760	0.383248	0.35
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.007368	-0.035587	-0.121515	0.04
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.611441	0.668023	0.188933	0.38
...	...	...	...	...	...	...	...	...	...	...	...	
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.910228	1.000000	0.460853	0.44
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.464741	0.460853	1.000000	0.17
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.444413	-0.441808	-0.177383	1.00
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.488676	0.543993	0.374044	0.36
PRICE	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.381626	-0.468536	-0.507787	0.33

Checking the shape of the dataframe used

In [17]:

```
boston_df.shape
```

Out[17]:

```
(506, 14)
```

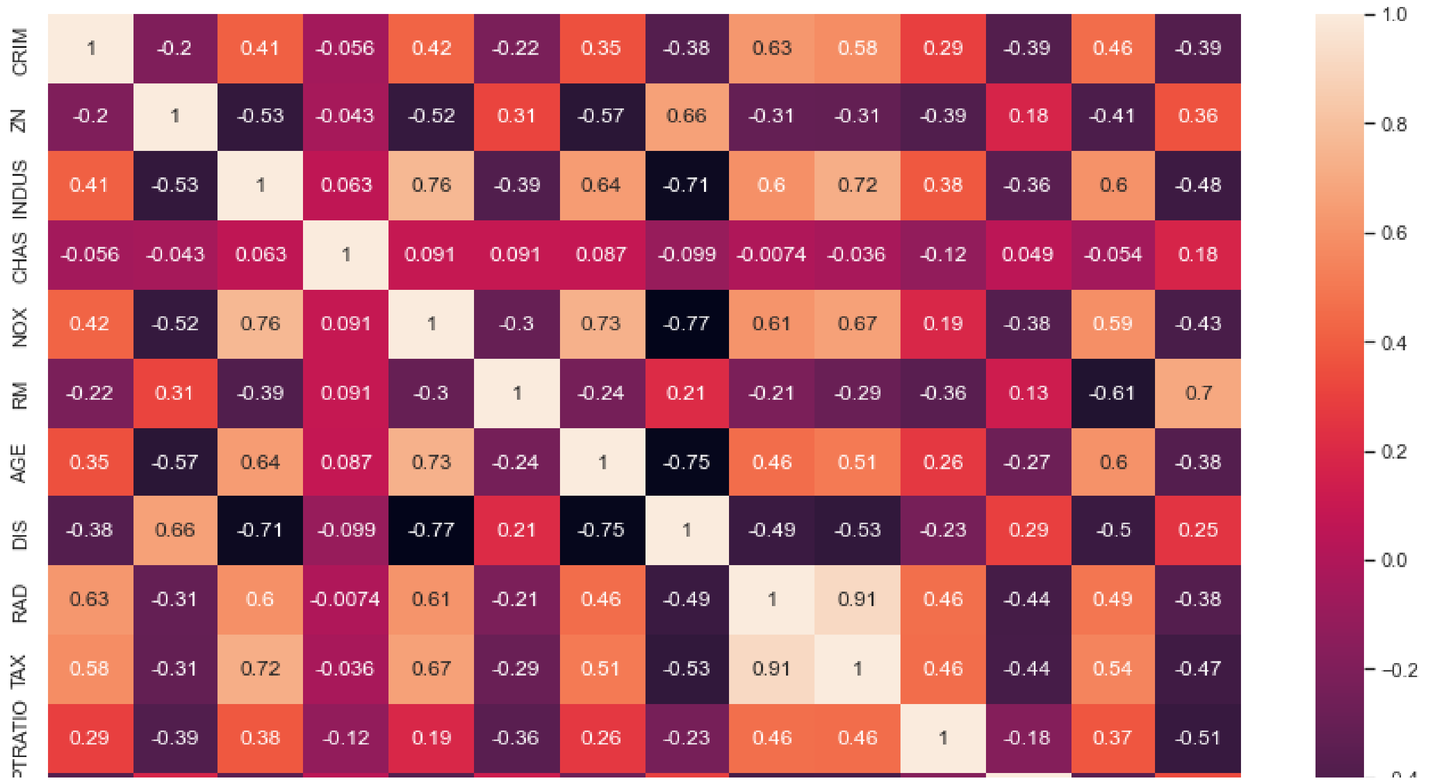
Plotting the Heatmap

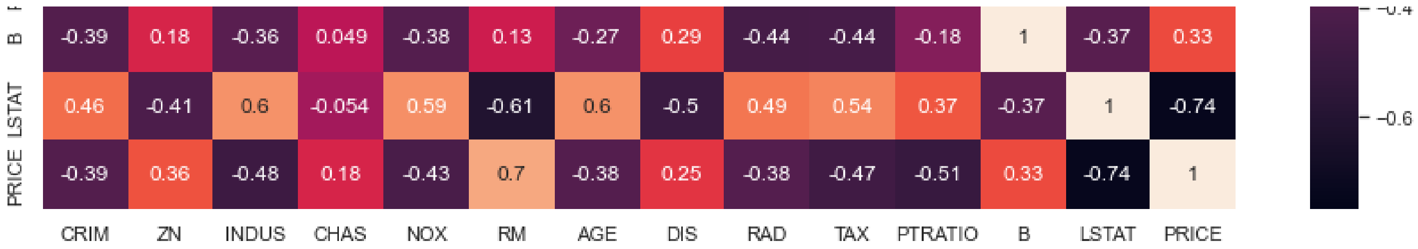
In [18]:

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

Out[18]:

```
<AxesSubplot:>
```





Data Visualization - Performing Basic EDA operations

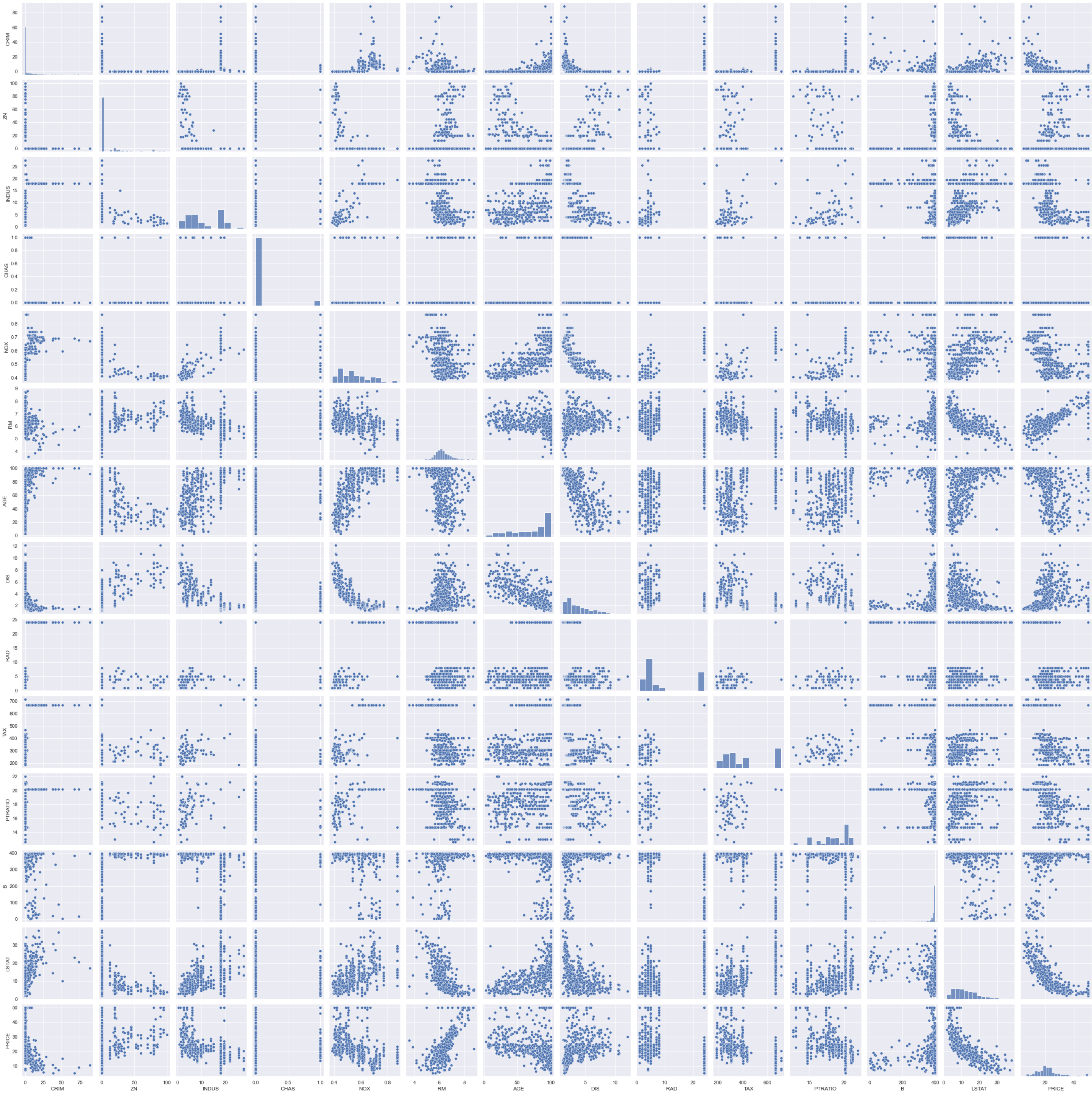
1. Pairplot

In [19]:

```
sns.pairplot(boston_df)
```

Out[19]:

<seaborn.axisgrid.PairGrid at 0x17faabe61d0>

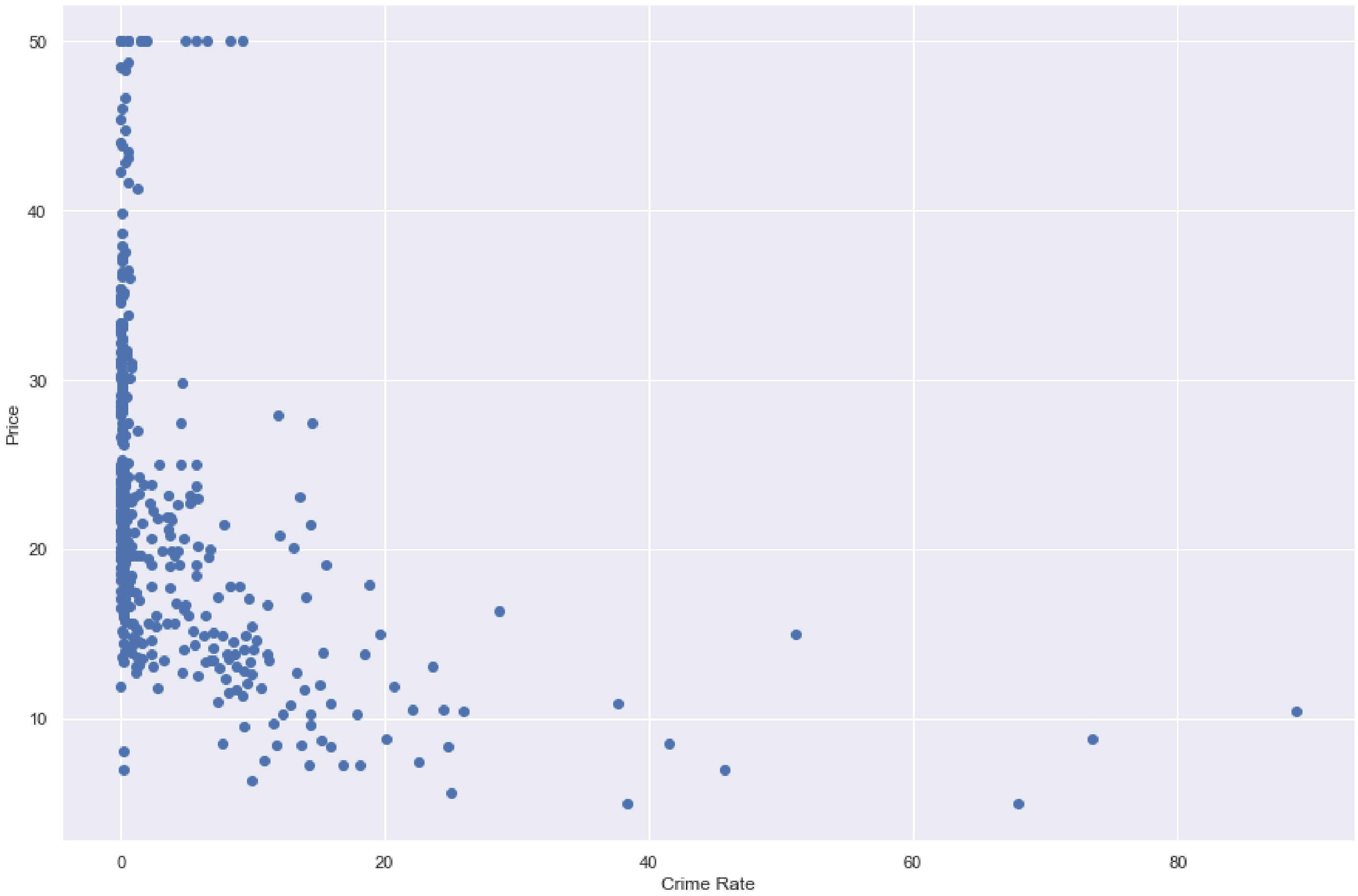


In [20]:

```
plt.scatter(boston_df['CRIM'],boston_df['PRICE'])
plt.xlabel("Crime Rate")
plt.ylabel("Price")
```

Out[20]:

Text(0, 0.5, 'Price')



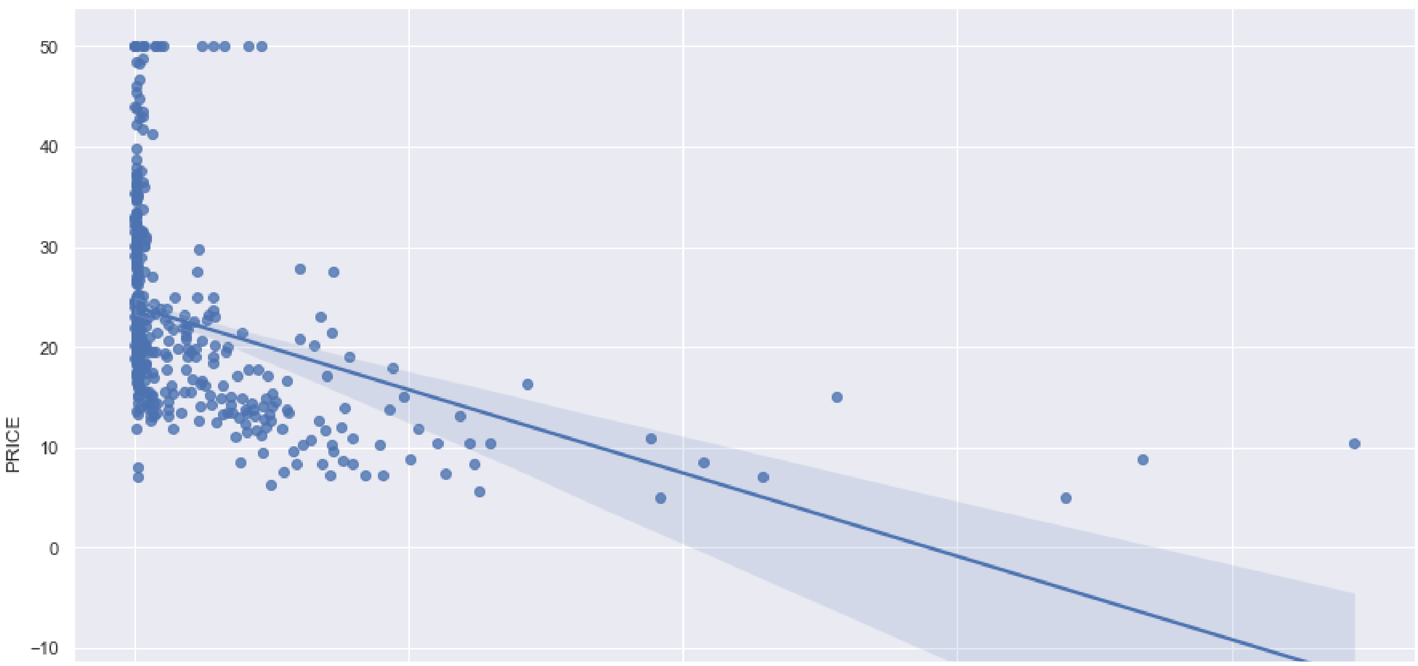
2. Scatterplot - CRIME RATE vs PRICE

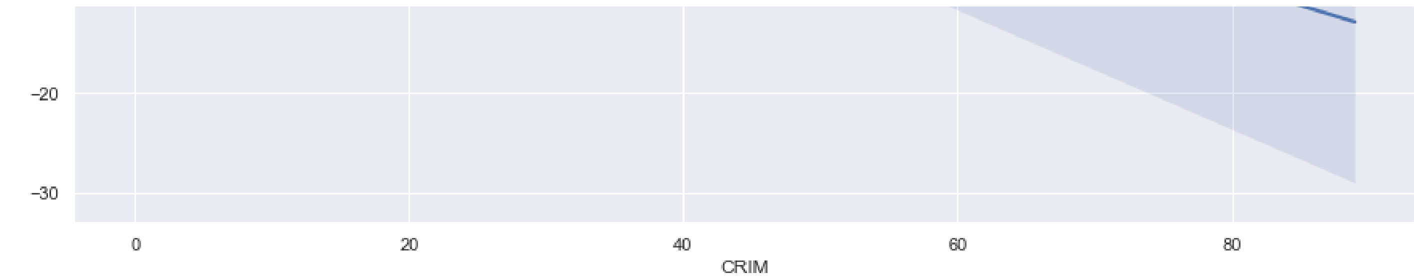
In [21]:

```
sns.regplot(x="CRIM",y="PRICE",data=boston_df)
```

Out[21]:

<AxesSubplot:xlabel='CRIM', ylabel='PRICE'>





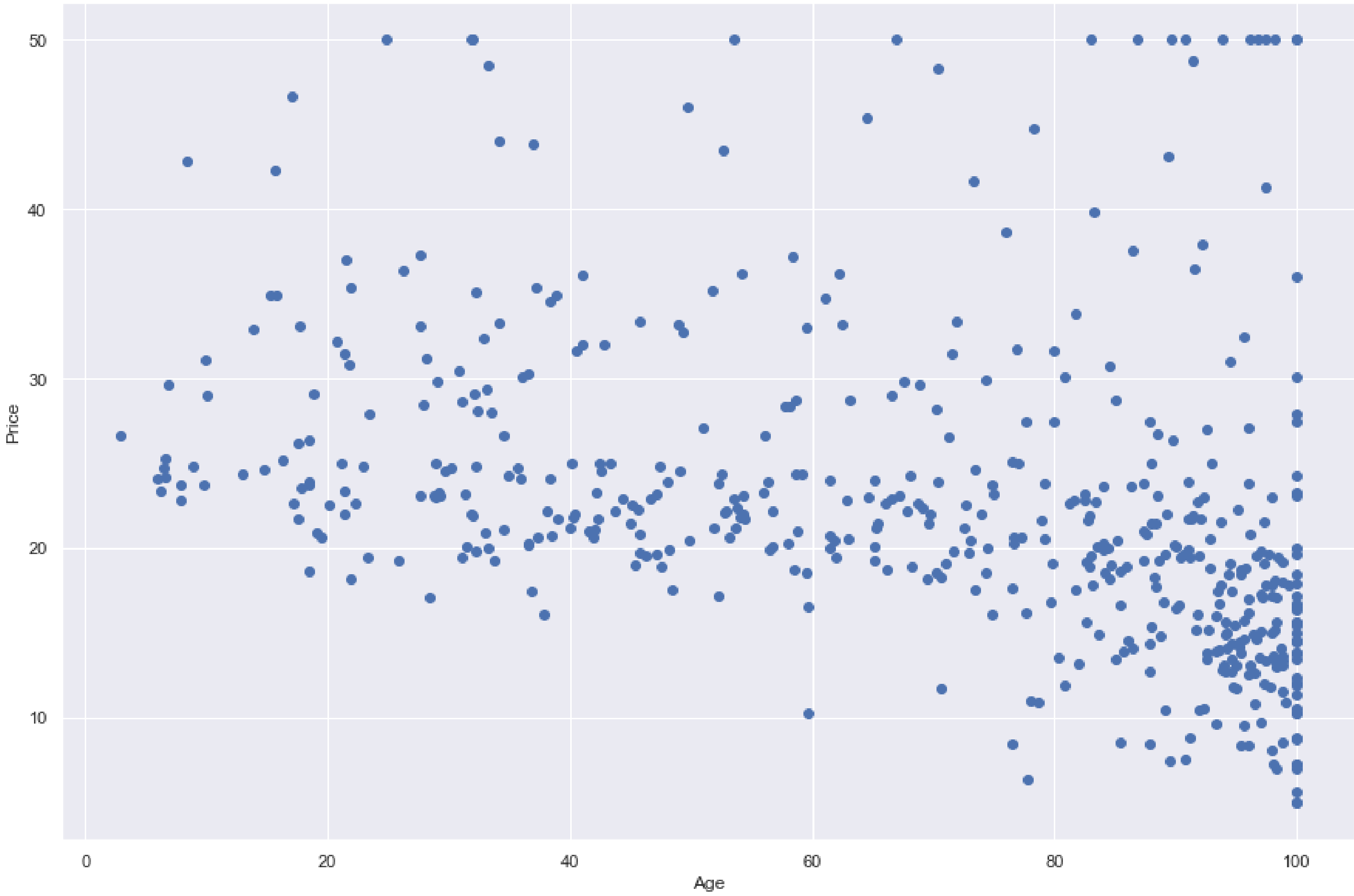
3. ScatterPlot - AGE vs PRICE

In [22]:

```
plt.scatter(boston_df['AGE'],boston_df['PRICE'])
plt.xlabel("Age")
plt.ylabel("Price")
```

Out[22]:

```
Text(0, 0.5, 'Price')
```

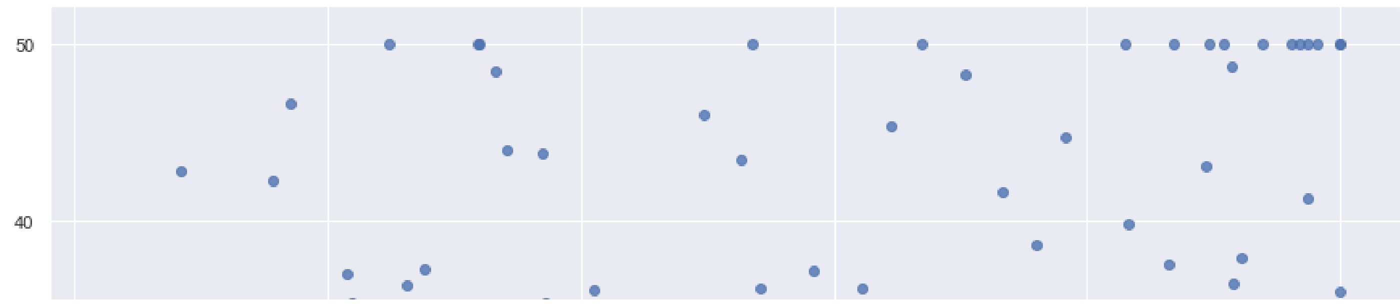


In [23]:

```
sns.regplot(x="AGE",y="PRICE",data=boston_df)
```

Out[23]:

```
<AxesSubplot:xlabel='AGE', ylabel='PRICE'>
```



4. ScatterPlot - ROOMS vs PRICE

In [24]:

```
plt.scatter(boston_df['RM'],boston_df['PRICE'])
plt.xlabel("Rooms Per Dwelling")
plt.ylabel("Price")
```

Out[24]:

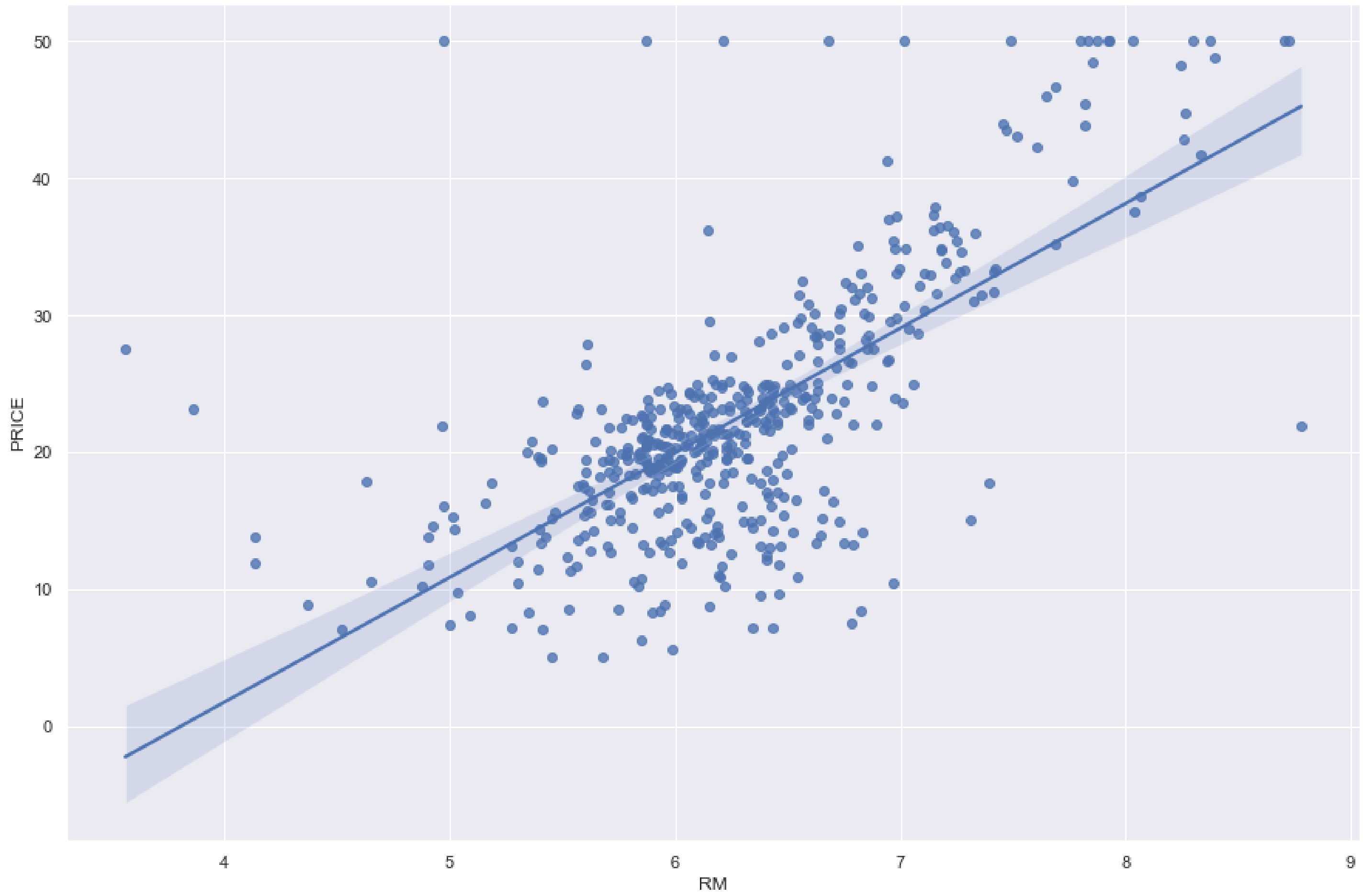
```
Text(0, 0.5, 'Price')
```

In [25]:

```
sns.regplot(x="RM",y="PRICE",data=boston_df)
```

Out[25]:

<AxesSubplot:xlabel='RM', ylabel='PRICE'>

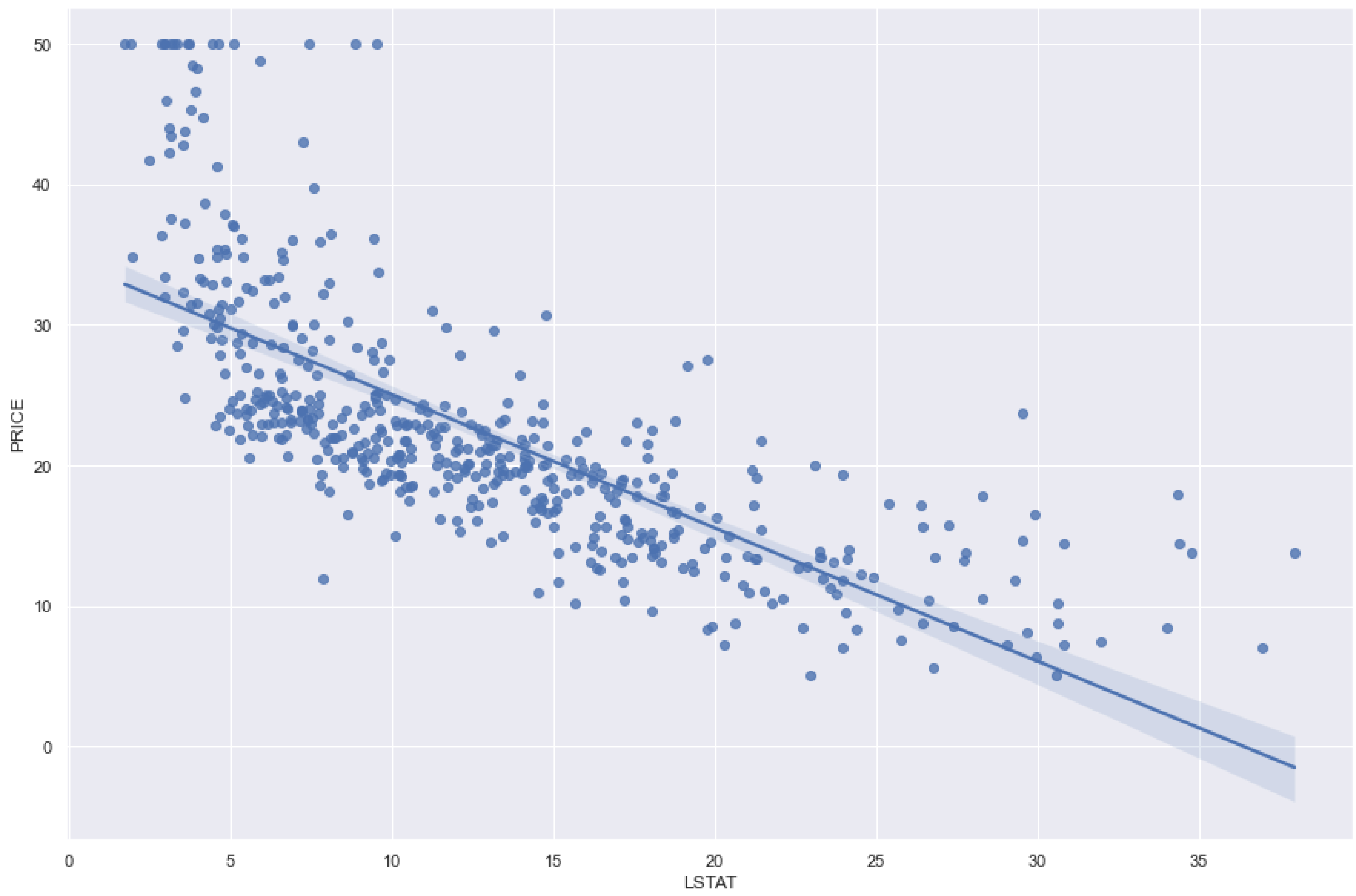


In [26]:

```
sns.regplot(x="LSTAT",y="PRICE",data=boston_df)
```

Out[26]:

<AxesSubplot:xlabel='LSTAT', ylabel='PRICE'>



Detecting the outliers present in the Criminal Rate Data Column using Box-Plot

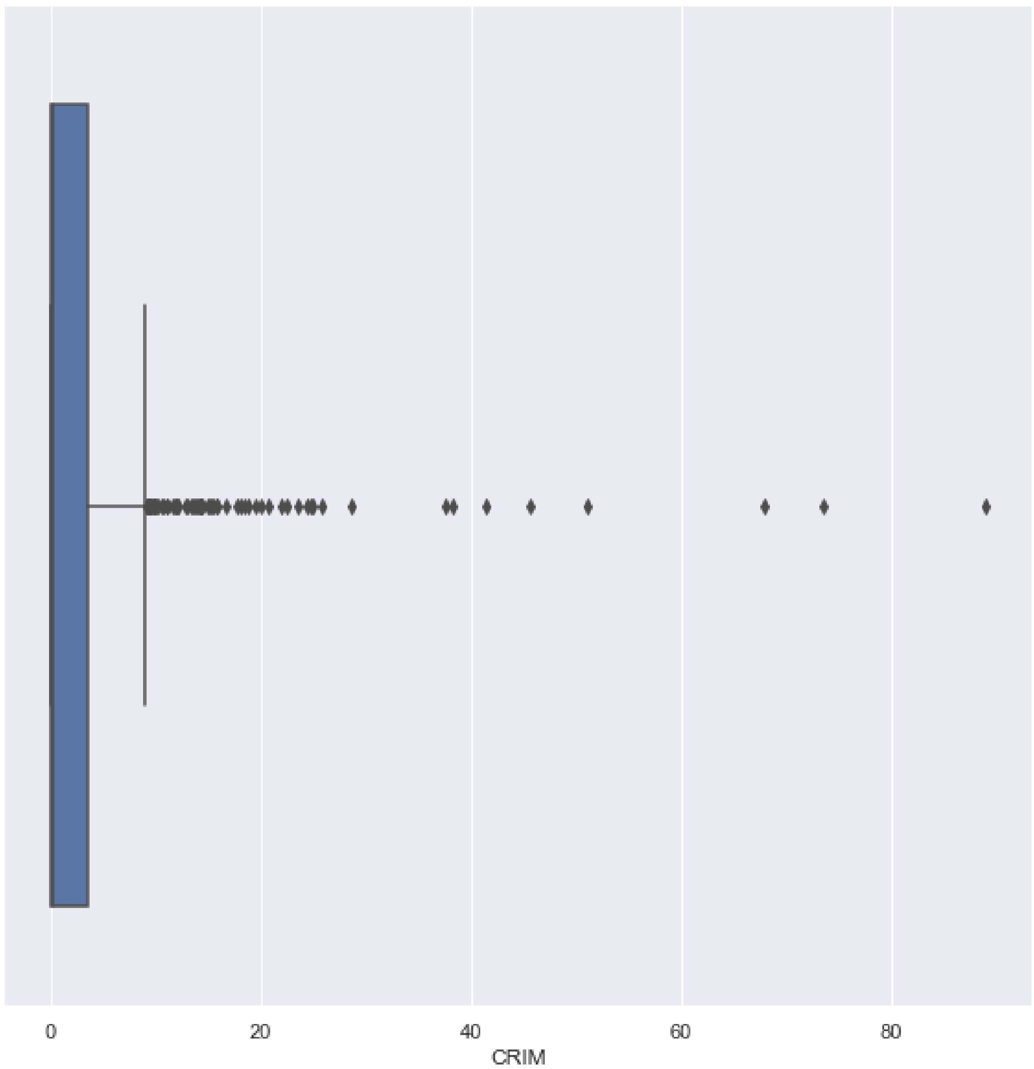
In [27]:

```
sns.set(rc={'figure.figsize':(10,10)})
sns.boxplot(boston_df['CRIM'])
```

```
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_de
corators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From vers
ion 0.12, the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[27]:

```
<AxesSubplot:xlabel='CRIM'>
```



Detecting the outliers present in the AGE Data Column using Box-Plot

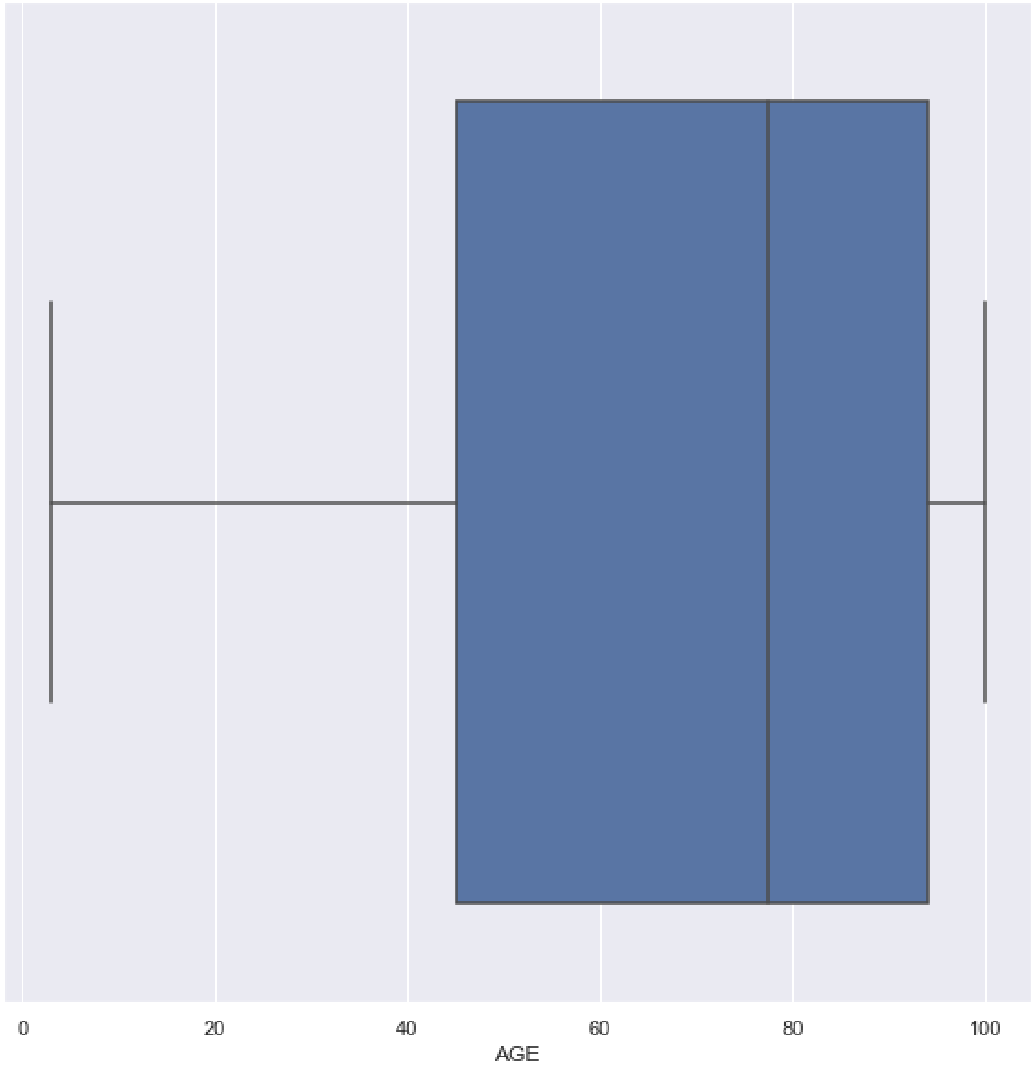
In [28]:

```
sns.set(rc={'figure.figsize':(10,10)})
sns.boxplot(boston_df['AGE'])
```

```
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_de
corators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From vers
ion 0.12, the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[28]:

```
<AxesSubplot:xlabel='AGE'>
```



Detecting the outliers present in the PRICE Column using Box-Plot

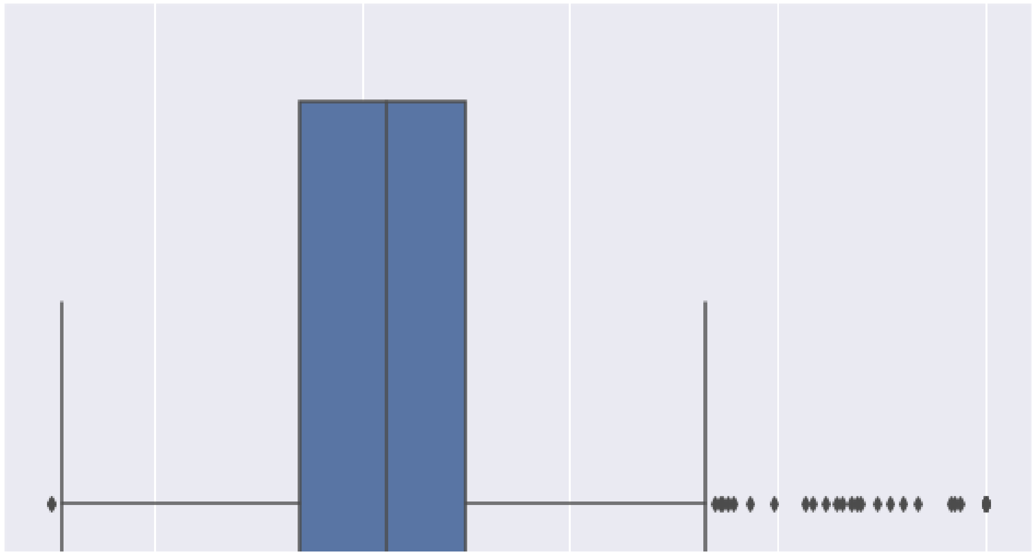
In [29]:

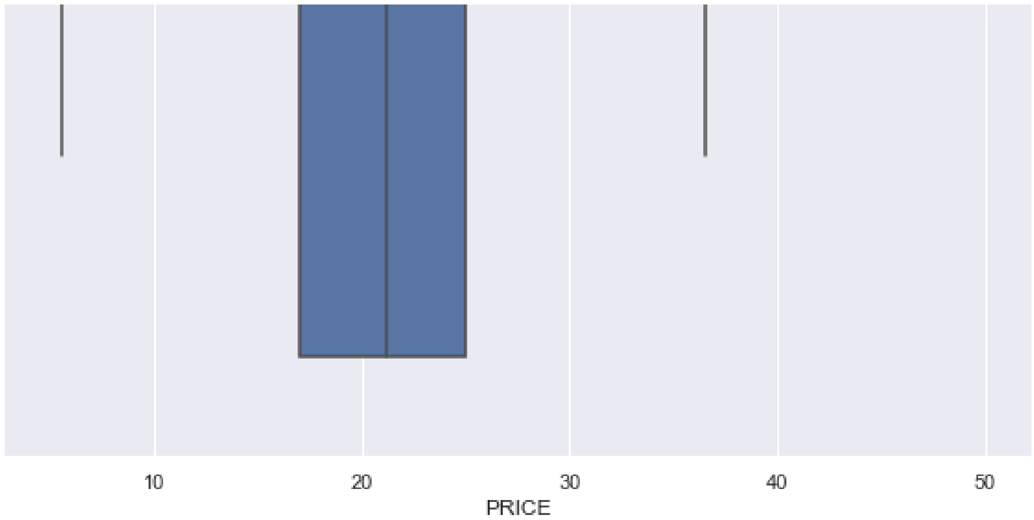
```
sns.set(rc={'figure.figsize':(10,10)})
sns.boxplot(boston_df['PRICE'])
```

```
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_de
corators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From vers
ion 0.12, the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[29]:

<AxesSubplot:xlabel='PRICE'>





Visualizing the DataFrame at a Glance using Pandas .head function

In [30]:

```
boston_df.head()
```

Out[30]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
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	24.0
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	21.6
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	34.7
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	33.4
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	36.2

Splitting the Dataset into Features and Label

In [31]:

```
X = boston_df.iloc[:,:-1]
Y = boston_df.iloc[:,-1]
```

In [32]:

```
X
```

Out[32]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
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	9.67
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	9.08
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	5.64
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	6.48
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	7.88

In [33]:

```
Y
```

Out[33]:

```
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 [34]:

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

Splitting the dataset into Train and Test Data into 70% and 30% Respectively

In [35]:

```
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.3, random_state=15)
```

In [36]:

```
X_train
```

Out[36]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
129	0.88125	0.0	21.89	0.0	0.624	5.637	94.7	1.9799	4.0	437.0	21.2	396.90	18.34
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
257	0.61154	20.0	3.97	0.0	0.647	8.704	86.9	1.8010	5.0	264.0	13.0	389.70	5.12
60	0.14932	25.0	5.13	0.0	0.453	5.741	66.2	7.2254	8.0	284.0	19.7	395.11	13.15
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
375	19.60910	0.0	18.10	0.0	0.671	7.313	97.9	1.3163	24.0	666.0	20.2	396.90	13.44
133	0.32982	0.0	21.89	0.0	0.624	5.822	95.4	2.4699	4.0	437.0	21.2	388.69	15.03
396	5.87205	0.0	18.10	0.0	0.693	6.405	96.0	1.6768	24.0	666.0	20.2	396.90	19.37
245	0.19133	22.0	5.86	0.0	0.431	5.605	70.2	7.9549	7.0	330.0	19.1	389.13	18.46
456	4.66883	0.0	18.10	0.0	0.713	5.976	87.9	2.5806	24.0	666.0	20.2	10.48	19.01

In [37]:

```
X_train.shape
```

Out[37]:

```
(354, 13)
```

In [38]:

```
y_train.shape
```

Out[38]:

```
(354,)
```

In [39]:

```
y_test.shape
```

Out[39]:

```
(152,)
```

In [40]:

```
X_test.shape
```

Out[40]:

```
(152, 13)
```

Scaling the Train and Test Data using sklearn StandardScaler

In [41]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [42]:

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

In [43]:

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

In [44]:

```
X_train_scaled
```

Out[44]:

```
array([[-0.32742136, -0.47138865,  1.5567594 , ...,  1.25320943,
         0.45613396,  0.81342951],
       [-0.43580738,  1.25986886, -1.45378412, ...,  0.56369103,
         0.38227796, -0.93425629],
       [-0.36172168,  0.3942401 , -1.05704583, ..., -2.51615782,
         0.38070656, -1.0423461 ],
       ...,
       [ 0.30728278, -0.47138865,  1.00395126, ...,  0.7935305 ,
         0.45613396,  0.95801718],
       [-0.41516181,  0.48080298, -0.78137106, ...,  0.28788367,
         0.37473522,  0.83027468],
       [ 0.15426348, -0.47138865,  1.00395126, ...,  0.7935305 ,
        -3.59201284,  0.90748168]])
```

In [45]:

```
X_test_scaled
```

Out[45]:

```
array([[-0.39138546,  0.90829998, -0.71005094, ..., -1.08471245,
         0.39241368, -0.47595831],
       [-0.34329631,  0.31784256, -1.01978929, ..., -2.5383999 ,
         0.2811574 , -0.97794122],
       [-0.38109148, -0.52566804, -1.00810105, ..., -0.85024673,
         0.40680566,  0.24974956],
       ...,
       [ 1.17984502, -0.52566804,  1.04464602, ...,  0.83790643,
         0.40680566,  1.14325115],
       [-0.37791113, -0.52566804, -0.34917654, ...,  1.16615844,
```

```
        0.3914125 ,  0.80486714],
       [-0.34087782, -0.52566804,  1.59837637, ...,  1.30683787,
         0.40680566,  0.77690152]])
```

In [46]:

```
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
```

In [47]:

```
LR
```

Out[47]:

```
▾ LinearRegression
LinearRegression()
```

In [48]:

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

Out[48]:

```
▾ LinearRegression
LinearRegression()
```

In [49]:

```
## print the coefficients and the intercept
print(LR.coef_)
```

```
[-7.30973225e-02  6.66062943e-02  8.45497046e-02  2.21512330e+00
 -2.27372067e+01  3.24861978e+00  2.06578129e-02 -1.59247039e+00
  3.48847293e-01 -1.39796398e-02 -9.58296625e-01  9.98858984e-03
 -5.92254599e-01]
```

In [50]:

```
print(LR.intercept_)
```

```
41.76845495082349
```

In [51]:

```
## Prediction for the test data
reg_pred=LR.predict(X_test)
```

In [52]:

```
reg_pred
```

Out[52]:

```
array([28.93841071, 40.17469652, 23.26283893, 22.72011976, 26.33677317,
        6.50809139, 16.72675328, 13.83049735, 28.38006838, 16.83901688,
       17.50579197, 22.45848043, 15.59048086, 16.11229233, 20.62101705,
       15.20710548,  8.47374859,  7.69857378, 21.45782622, 10.97606569,
       38.72583349, 13.26023439, 23.33227986, 19.27402726, 19.3360351 ,
       19.62525449, 27.32359007, 19.91480848, 19.97039516, 19.98919575,
       21.45883975,  7.54689782, 20.33795817, 19.38369205, 23.37468039,
       19.05153146, 24.46267997, 28.19200979, 20.69966547, 18.68680301,
       28.11584489, 35.29854655, 20.0879725 , 27.8604335 , 25.57788978,
       21.59692292, 21.74601139, 30.24313863, 25.66136714, 20.36289475,
       31.39205843, 15.24938636, 14.28689956, 14.33724217, 17.70236617,
       30.67294605,  8.45801637, 29.38244272, 16.52514507, 26.35269311,
       17.64563127, 27.64146931, 18.83849367, 30.29337701, 34.33685682,
       20.40053045, 23.50840914, 18.29950906, 25.26796658, 18.96662268,
       19.54785317, 23.47174606, 20.06513963, 12.71104176, 34.58884204,
       20.18038844, 36.73034446, 17.86919358, 20.77346653, 13.21836627,
```

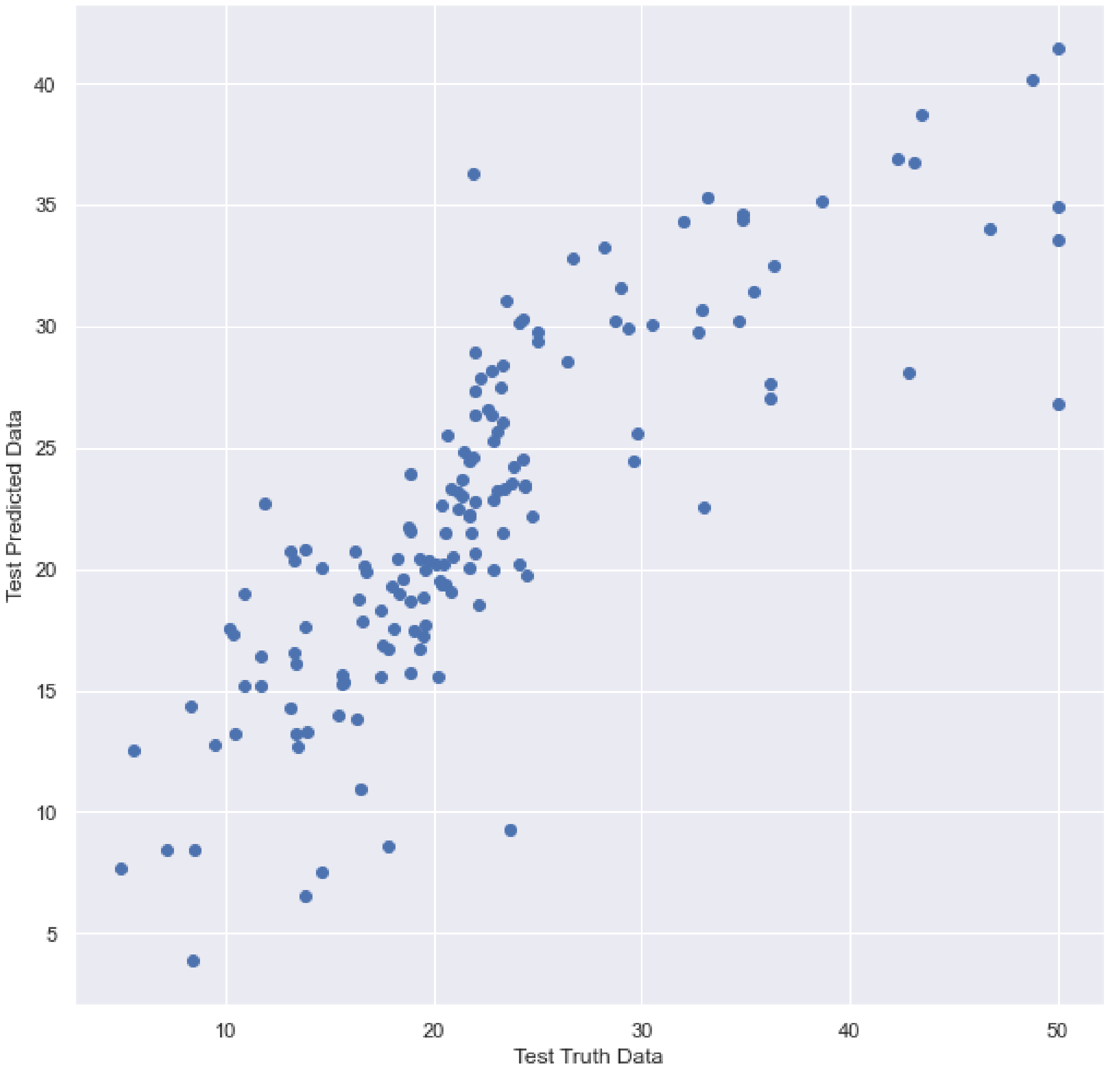
```
       15.31962116, 17.54865527, 36.91230929, 22.57139152, 34.90624299,
       18.75605528, 29.72788323, 23.65381032, 26.57590857, 20.73578394,
       30.18801597, 24.57127722, 36.24961976, 23.92962871, 26.06991012,
       32.79245337, 15.70279192, 27.04841957, 22.23639227, 22.14422298,
       20.04264192, 12.51038729, 30.05440823,  3.91508266, 24.43278658,
       13.96740998, 23.30762929, 24.83459343, 21.47895717, 35.16407276,
       29.91813952, 23.00538811, 20.21991859, 26.7787996 , 18.54381139,
       25.54068621, 29.75000896, 15.542685  , 15.20621604, 31.59691836,
       24.48797917, 12.79369385, 20.42987578, 33.54619872, 30.10169183,
       16.38623272, 22.77814   , 41.41331714, 34.03046388, 24.20339851,
       34.37334928, 22.64842338,  8.61530203, 15.61618271, 13.20934389,
       22.16702318, 32.47200523,  9.2780732 , 31.05708968, 27.49363576,
       22.81749188, 17.4918494 , 20.47648792, 28.54144204, 23.17545754,
       20.19885139, 19.7480775 , 17.29339103, 33.25948721, 18.95593496,
       17.24115455, 16.74058242])
```

In [53]:

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

Out[53]:

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



In [54]:

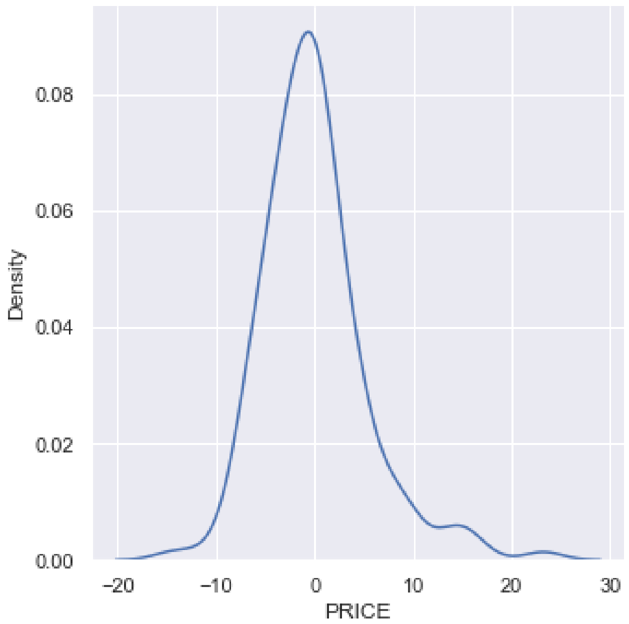
```
## residuals
residuals=y_test-reg_pred
```

In [55]:

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

Out[55]:

```
<seaborn.axisgrid.FacetGrid at 0x17fb65292a0>
```

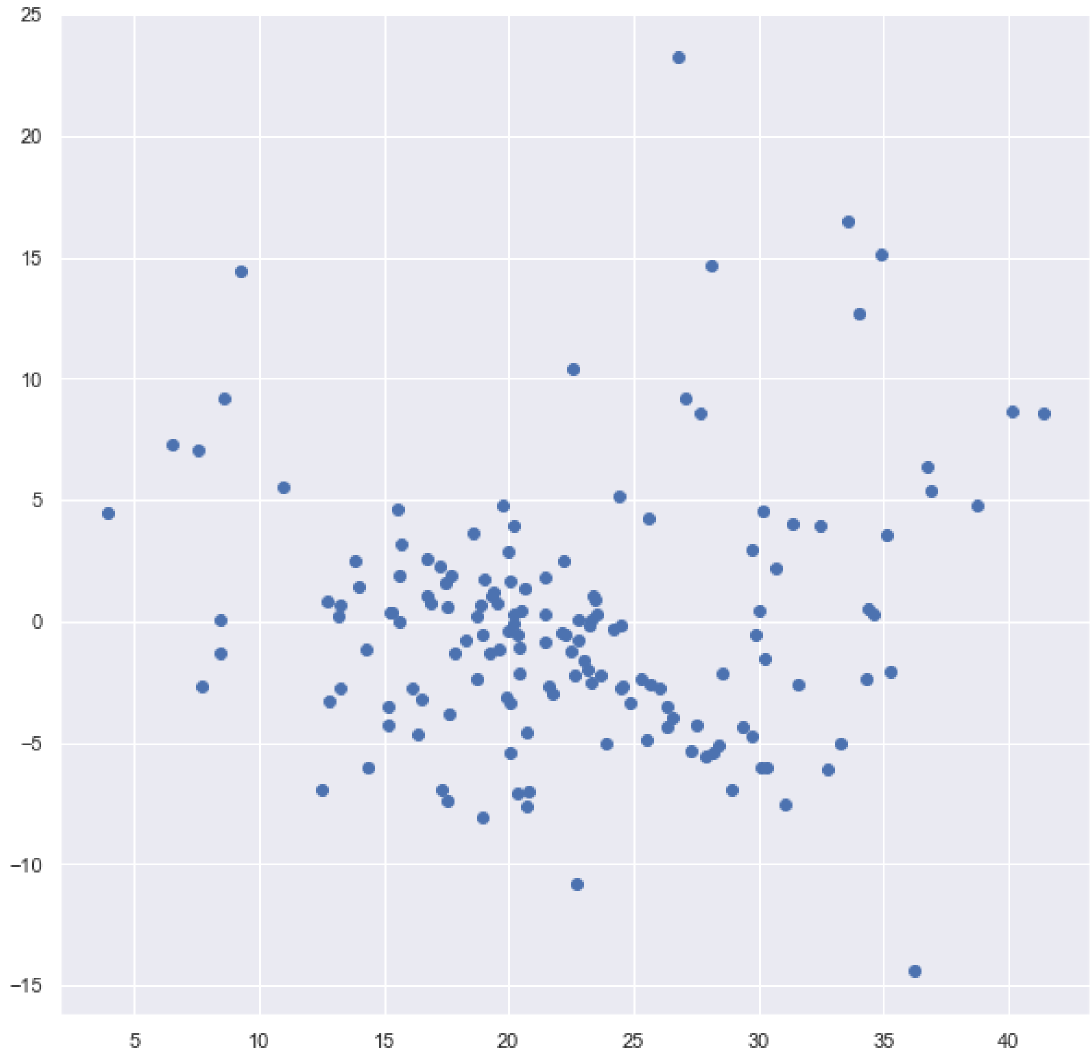


In [56]:

```
plt.scatter(reg_pred,residuals)
```

Out[56]:

```
<matplotlib.collections.PathCollection at 0x17fb8c36b30>
```



In [57]:

```
## Performance Metrics
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)))
```

```
27.58101431810535
3.7410552297491826
5.251762972384164
```

In [58]:

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

```
0.6745262607211457
```

In [59]:

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

Out[59]:

```
0.6438656910789349
```

Replacing the Xtrain and Xtest Data with their scaled value and check whether there is any significant changes in the R^2

In [60]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

In [61]:

```
regressor.fit(X_train_scaled,y_train)
```

Out[61]:

```
▾ LinearRegression
LinearRegression()
```

In [62]:

```
print(regressor.coef_)
```

```
[-0.57477822  1.53891131  0.57966473  0.52327069 -2.61571068  2.30712032
  0.58203021 -3.24197468  3.03389615 -2.37665943 -2.0847086   0.95347108
 -4.21904768]
```

In [63]:

```
print(regressor.intercept_)
```

```
22.598870056497184
```

In [64]:

```
## Prediction for the test data
reg_pred_scaled=regressor.predict(X_test_scaled)
reg_pred_scaled
```

Out[64]:

```
array([29.42463134, 40.61980423, 23.29897949, 22.74061402, 26.4525965 ,
        6.28990307, 16.47974461, 13.56370461, 28.95497615, 17.49308995,
       17.47719586, 22.97436652, 16.3614426 , 15.45013899, 21.12309223,
       14.91234242,  8.43924416,  8.04022778, 21.83340869, 11.56712765,
```

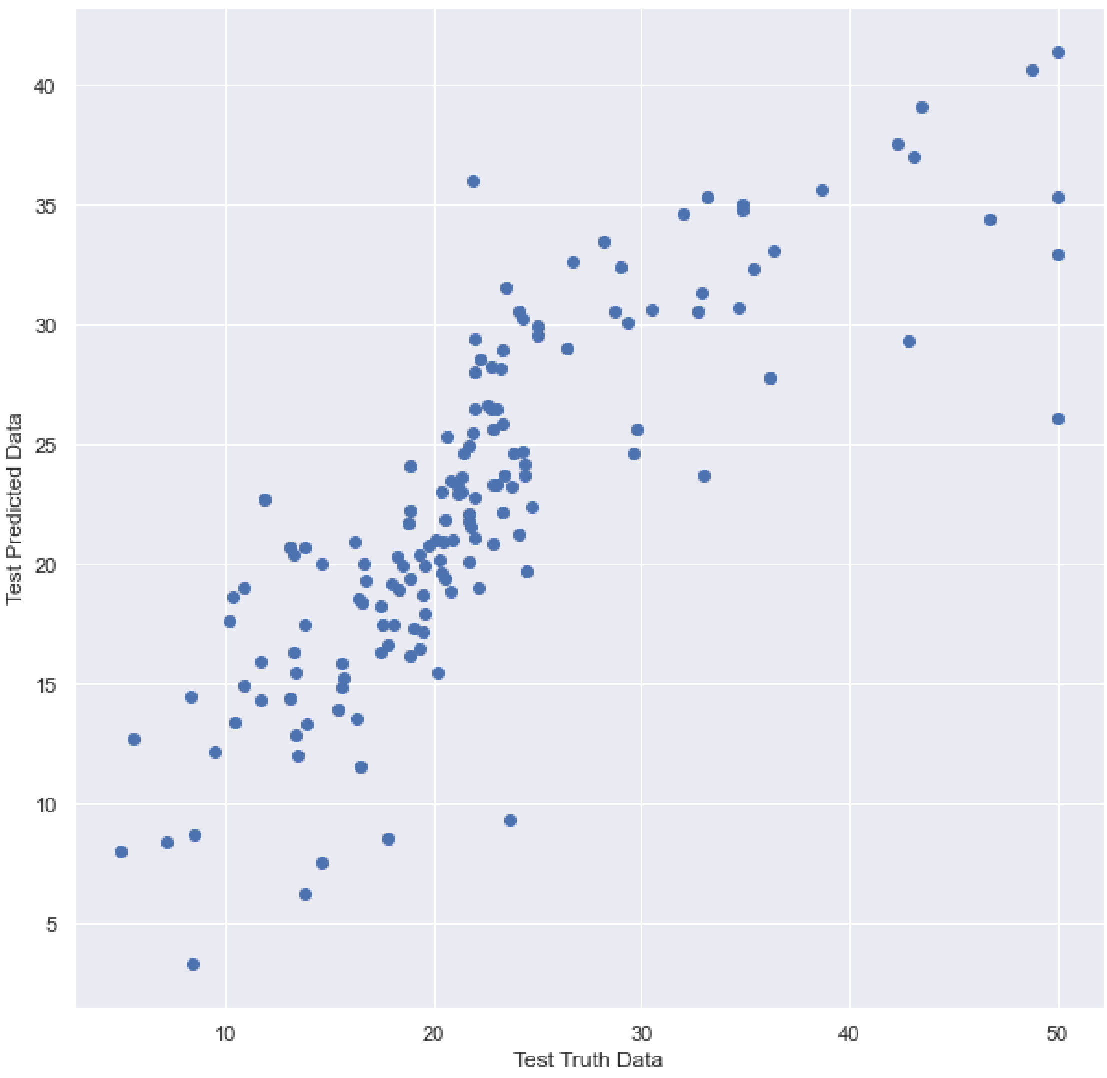
```
       19.92020457, 28.03197224, 19.32099938, 19.96826541, 20.86872688,
       22.19425428,  7.53319237, 20.773365  , 19.39044594, 23.72136932,
       18.87022269, 24.96021168, 28.23281392, 20.69551791, 19.37194935,
       29.28811809, 35.34873981, 19.99782097, 28.54655757, 25.61194664,
       22.23749836, 21.74517391, 30.55628098, 26.44523454, 20.37401324,
       32.33920514, 14.87300921, 14.39367596, 14.45346953, 17.92770041,
       31.34936014,  8.73807517, 29.57151886, 16.30931465, 26.45033129,
       17.47457576, 27.75310272, 18.71862476, 30.21772465, 34.66598234,
       20.40381065, 23.27971114, 18.24107235, 25.64969651, 18.94833594,
       20.15482152, 24.18153851, 20.07589036, 12.03550143, 35.02016365,
       21.25567391, 37.04163541, 18.4118881 , 20.74489414, 13.39241986,
       15.23374827, 17.61224194, 37.55997143, 23.70708553, 35.31240389,
       18.59071991, 30.55847821, 23.6698027 , 26.6435274 , 20.9614213 ,
       30.74266834, 25.44464509, 35.99961779, 24.06620718, 25.83510891,
       32.60717018, 16.1425784 , 27.7594317 , 22.0735267 , 21.78537948,
       20.03481481, 12.71558176, 30.65271935,  3.36324905, 24.63843861,
       13.96277842, 23.67535194, 24.59990408, 21.54826575, 35.59706163,
       30.12203431, 23.03074256, 20.97569756, 26.07544888, 19.00141849,
       25.34055397, 29.91583008, 15.45227059, 14.32601875, 32.36723659,
       24.67705534, 12.18934835, 20.32484065, 32.96968048, 30.56629772,
       15.94230652, 22.82085236, 41.36355971, 34.37172257, 24.61744623,
       34.76112273, 22.98704168,  8.57551228, 15.90253854, 12.85275527,
       22.37127071, 33.12591892,  9.33521016, 31.56154069, 28.16215999,
       23.35092939, 17.32183768, 20.98538519, 28.98690929, 23.22604912,
       20.98033326, 19.73015481, 18.5988905 , 33.46069977, 19.05492711,
       17.1429949 , 16.62854312])
```

In [65]:

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

Out[65]:

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



In [66]:

```
## residuals
residuals_scaled=y_test-reg_pred_scaled
```

In [67]:

```
residuals_scaled
```

Out[67]:

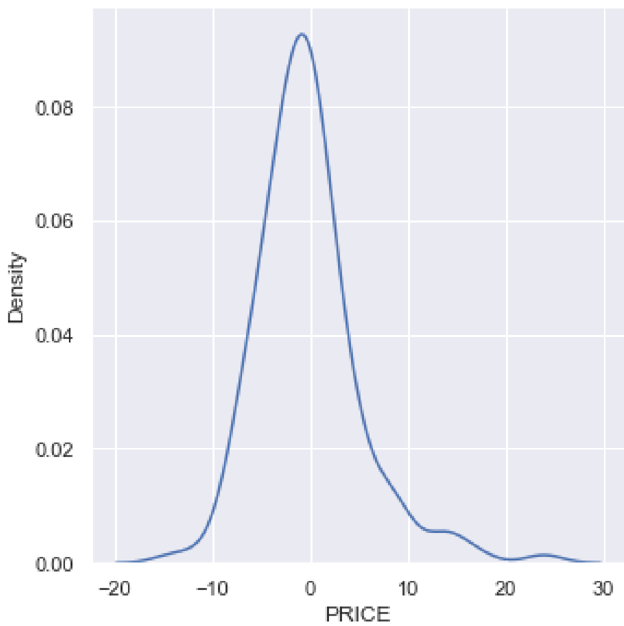
```
301    -7.424631
262     8.180196
172    -0.198979
505   -10.840614
111    -3.652596
         ...
380    -8.198891
307    -5.260700
381    -8.154927
106     2.357005
139     1.171457
Name: PRICE, Length: 152, dtype: float64
```

In [68]:

```
sns.displot(residuals_scaled,kind="kde")
```

Out[68]:

```
<seaborn.axisgrid.FacetGrid at 0x17fb8c82f50>
```

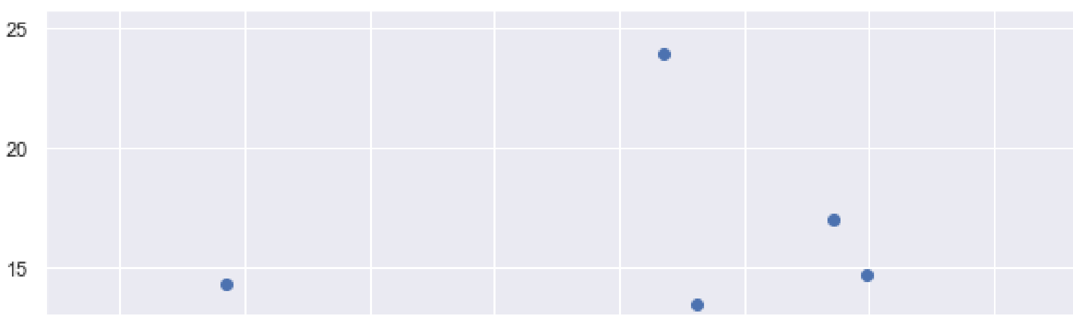


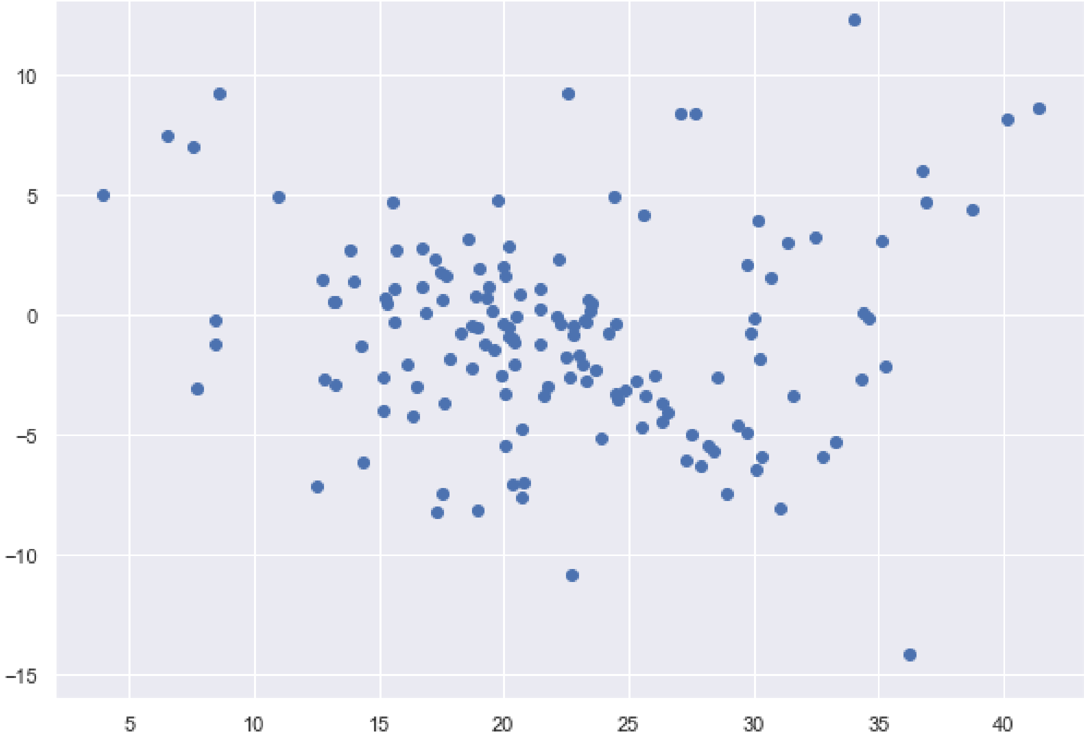
In [69]:

```
plt.scatter(reg_pred,residuals_scaled)
```

Out[69]:

```
<matplotlib.collections.PathCollection at 0x17fb8d721a0>
```





In [70]:

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

```
27.46473809683593
3.727436748688041
5.2406810718489565
```

In [71]:

```
from sklearn.metrics import r2_score
score_scaled=r2_score(y_test,reg_pred_scaled)
print(score_scaled)
```

```
0.6758983950483787
```

In [72]:

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

Out[72]:

```
0.6453670844369941
```

There is no such significant changes between the Normal and Scaled R^2 Values

Evaluating the Dataset using Ridge Regression

In [73]:

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

In [74]:

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

Out[74]:

```
▾ Ridge
Ridge()
```

In [75]:

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

In [76]:

```
ridge_pred
```

Out[76]:

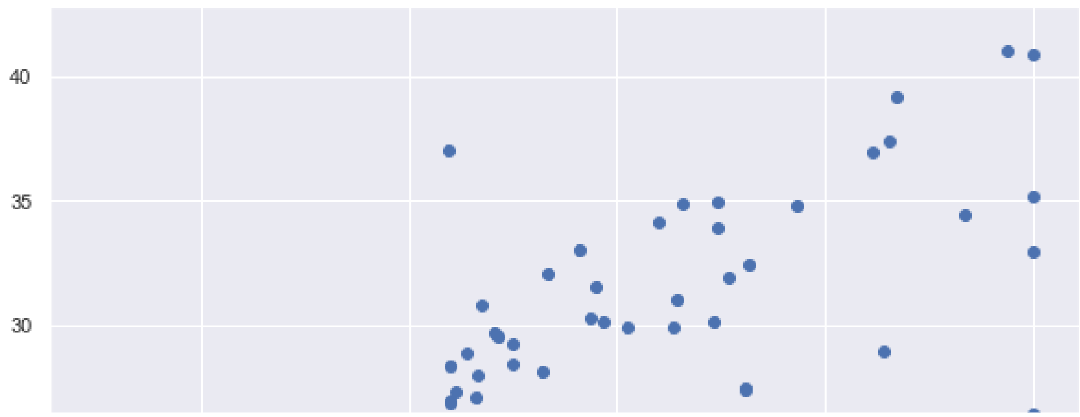
```
array([28.34700411, 40.99020024, 22.61027099, 23.2155383 , 26.11759627,
        5.98219852, 15.62597673, 12.83874366, 27.99127334, 16.39679102,
       17.63269965, 22.29196843, 15.78474533, 16.47304297, 20.43928823,
       15.53700685,  8.49051102,  7.8570057 , 22.21636121, 10.37300533,
       39.15601791, 13.50243764, 22.42297964, 19.37771    , 20.02993273,
       19.47530705, 26.96124868, 20.72561217, 20.17750241, 20.71514059,
       21.94343695,  9.32729636, 20.38013688, 19.1333901 , 23.19582726,
       20.02576519, 23.47458314, 28.86578648, 20.87954425, 18.47360772,
       28.94436027, 34.87608624, 20.54470438, 27.33611564, 25.40585055,
       21.40416119, 21.0462387 , 30.30832271, 26.11474415, 20.26341559,
       31.91067008, 17.09132649, 14.36271739, 14.53180137, 17.96166841,
       31.01618851,  8.82019251, 28.42608421, 15.4304323 , 26.85162603,
       16.61153694, 27.46829035, 18.41043268, 29.5195852 , 34.09605507,
       20.22223502, 22.90603938, 18.54047753, 25.00272711, 19.63226713,
       18.43735905, 23.78779977, 20.24490781, 13.18344169, 34.96474836,
       20.79023033, 37.42422335, 17.13861604, 21.01270366, 13.99123209,
       14.4508423 , 17.43909906, 36.97607769, 22.90718041, 35.16515341,
       19.2989718 , 29.88230126, 23.07193895, 26.22457409, 20.75840548,
       30.13141762, 24.5859121 , 37.01350018, 23.78615258, 25.57973946,
       32.02685983, 15.78405085, 27.41134644, 21.26214375, 23.12465509,
       19.41357858, 12.66093485, 29.90958765,  4.38255416, 24.25073335,
       15.91089374, 22.97921236, 23.89152687, 20.77540019, 34.81754799,
       30.12105276, 22.52794502, 19.86807171, 26.46315873, 18.84789258,
       24.91207356, 29.2459565 , 16.2150825 , 14.22966397, 31.55193231,
       24.08301293, 12.64748472, 19.93409229, 32.92761943, 29.70919044,
       16.78158311, 22.41770084, 40.88693989, 34.39226363, 24.06135042,
       33.92040686, 22.83345534, 10.45132671, 15.85212532, 14.81175304,
       22.48341944, 32.44637478,  9.10872096, 30.80920283, 27.10737493,
       22.70310752, 16.47854624, 20.17080451, 28.16656697, 22.97102641,
       20.38098202, 20.3644423 , 18.08376687, 33.00326745, 18.90459902,
       16.85547231, 16.76993899])
```

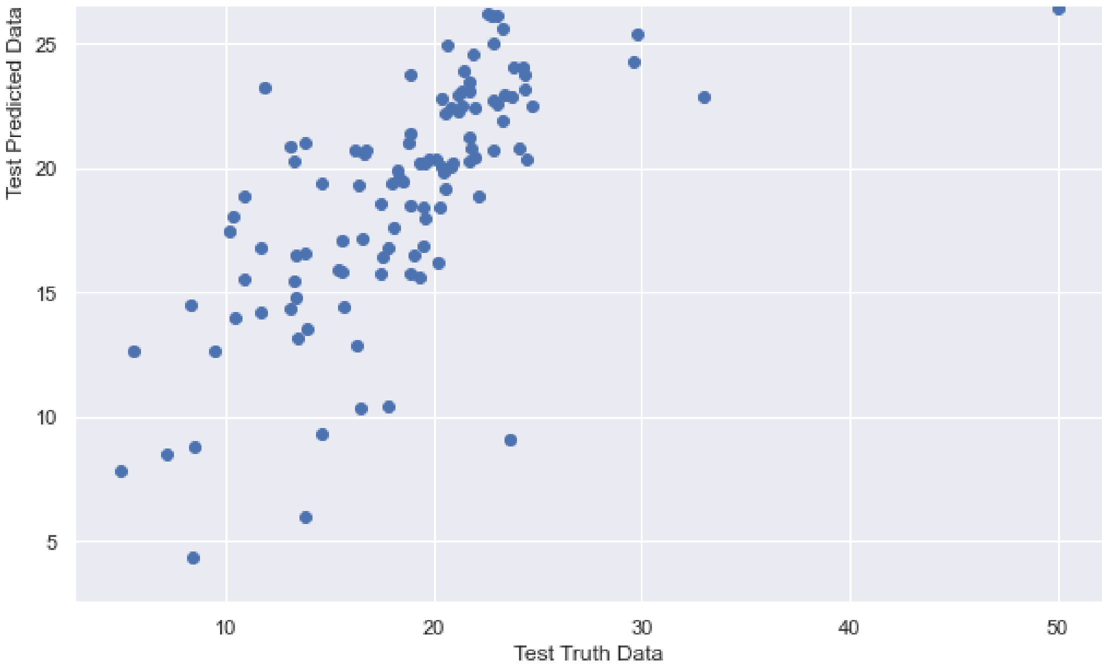
In [77]:

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

Out[77]:

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





In [78]:

```
## residuals
ridge_residuals=y_test-ridge_pred
ridge_residuals
```

Out[78]:

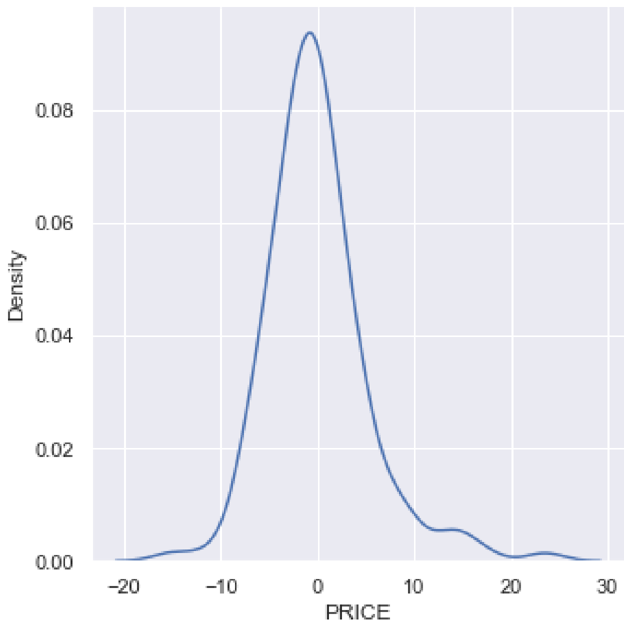
```
301    -6.347004
262     7.809800
172     0.489729
505   -11.315538
111    -3.317596
         ...
380    -7.683767
307    -4.803267
381    -8.004599
106     2.644528
139     1.030061
Name: PRICE, Length: 152, dtype: float64
```

In [79]:

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

Out[79]:

```
<seaborn.axisgrid.FacetGrid at 0x17fb8d9f850>
```

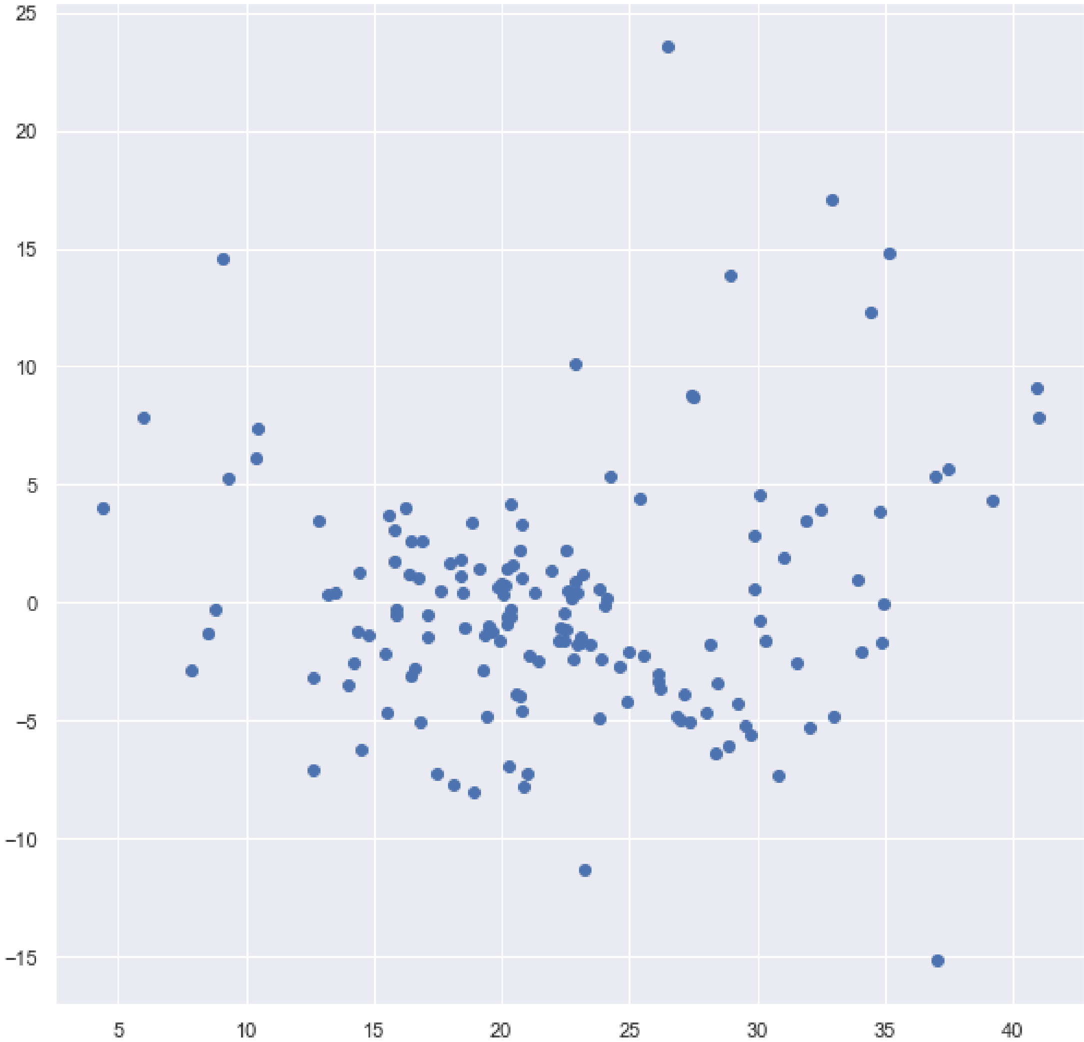


In [80]:

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

Out[80]:

```
<matplotlib.collections.PathCollection at 0x17fb8ea9540>
```



In [81]:

```
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)))
```

```
26.895629415693488
3.6794629931440492
5.186099634184971
```

In [82]:

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

```
0.6826142441600589
```

In [83]:

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

Out[83]:

```
0.6527155860012239
```

Evaluating the Dataset using the Lasso Regression

In [84]:

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

In [85]:

```
lasso
```

Out[85]:

```
▼ Lasso
Lasso()
```

In [86]:

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

Out[86]:

```
▼ Lasso
Lasso()
```

In [87]:

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

```
42.22410959384636
```

In [88]:

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

```
[-0.02663527  0.07056985 -0.          0.         -0.          0.5581624
  0.03947285 -0.70744747  0.26299305 -0.01560035 -0.68467839  0.00800488
 -0.81411349]
```

In [89]:

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

In [90]:

```
lasso_pred
```

Out[90]:

```
array([27.12941361, 36.88288348, 23.53663729, 25.39359024, 24.84286728,
        4.74509463, 13.77197408, 14.59393706, 27.91697514, 20.70575638,
       17.19959477, 23.38059776, 18.76539026, 15.71949933, 21.9400187 ,
       16.89252795,  7.77093828,  7.50463158, 23.9060242 , 10.67279224,
       36.68242587, 16.43054605, 21.60211303, 18.79002068, 22.97393494,
       21.98666538, 27.36347518, 20.51518792, 20.88869769, 24.0578185 ,
       24.76600528, 11.71271316, 21.09083519, 18.65597701, 22.96610777,
       24.27196609, 23.65172725, 30.72143679, 19.57450749, 20.41043529,
       26.31788589, 30.87741489, 20.55445125, 28.70729577, 22.47098468,
       23.75433348, 21.10560936, 28.90566851, 28.19825584, 16.67068047,
       32.12026299, 20.94566571, 16.26353947, 16.30748208, 19.98072517,
       30.37613072,  8.87735163, 26.18796243, 12.67216926, 27.34816607,
       17.23128131, 27.34355876, 18.57341514, 30.83959245, 33.33076023,
       20.40398752, 23.76843679, 19.73860467, 26.12468643, 19.82619535,
       19.14066668, 24.65846193, 19.44224362, 14.85090005, 34.12218508,
       24.57761427, 35.32778313, 17.67055649, 20.75709165, 14.43617187,
       13.07633825, 15.7003081 , 34.14255471, 24.5261761 , 31.5570532 ,
       18.31798381, 29.61277049, 23.18107675, 26.92377089, 23.20020534,
```

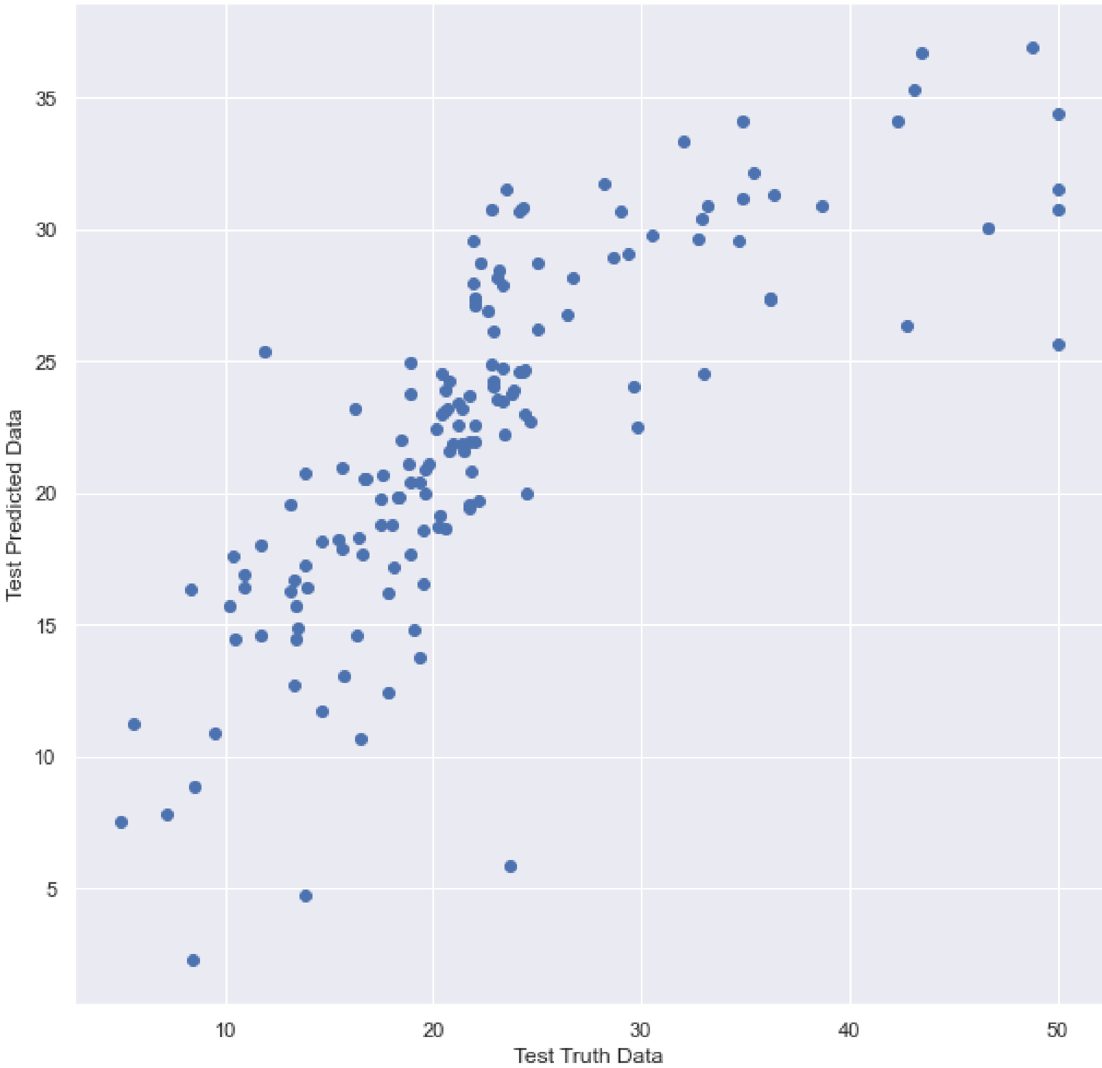
```
       29.55871397, 27.93439204, 29.58002763, 24.96195126, 23.44437412,
       28.19490581, 17.65750882, 27.42232493, 19.52255178, 21.93159263,
       18.13777849, 11.19682599, 29.79131801,  2.30146038, 24.03640166,
       18.25897984, 22.23769927, 21.57007722, 20.81030838, 30.89776734,
       29.08864257, 21.89925985, 23.03711848, 25.66765445, 19.68908392,
       23.18863185, 28.70904018, 18.68770727, 14.57359524, 30.66186561,
       24.58434053, 10.89134835, 19.84919038, 30.72379031, 30.70017339,
       18.0433517 , 22.58478993, 34.40515005, 30.02136614, 23.9168915 ,
       31.20069332, 24.54281215, 12.44231477, 17.88352247, 14.46959123,
       22.73879405, 31.34618692,  5.81778454, 31.55147121, 28.4583525 ,
       24.25812821, 14.77655907, 21.84372431, 26.78499449, 22.59591976,
       22.40993065, 19.96668011, 17.62667583, 31.72425704, 16.39858989,
       16.53095951, 16.19477664])
```

In [91]:

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

Out[91]:

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



In [92]:

```
## residuals
lasso_residuals=y_test-lasso_pred
lasso_residuals
```

Out[92]:

```
301    -5.129414
262    11.917117
172    -0.436637
505   -13.493590
```

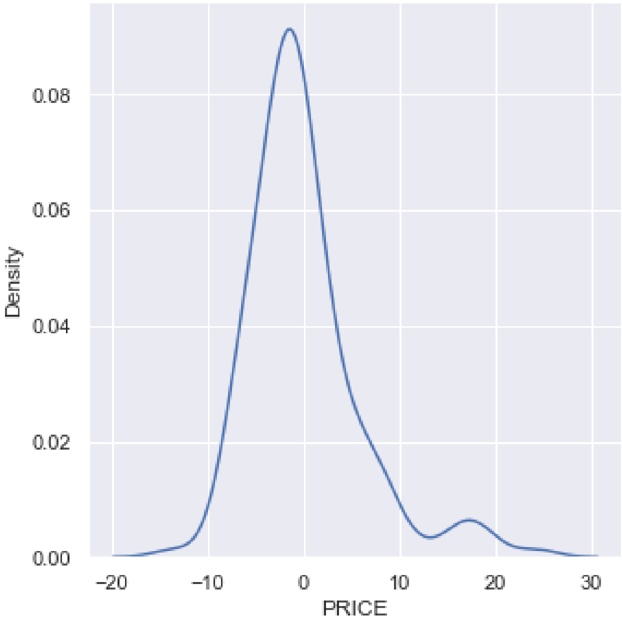
```
111     -2.042867
          ...
380     -7.226676
307     -3.524257
381     -5.498590
106      2.969040
139      1.605223
Name: PRICE, Length: 152, dtype: float64
```

In [93]:

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

Out[93]:

```
<seaborn.axisgrid.FacetGrid at 0x17fb8eed330>
```

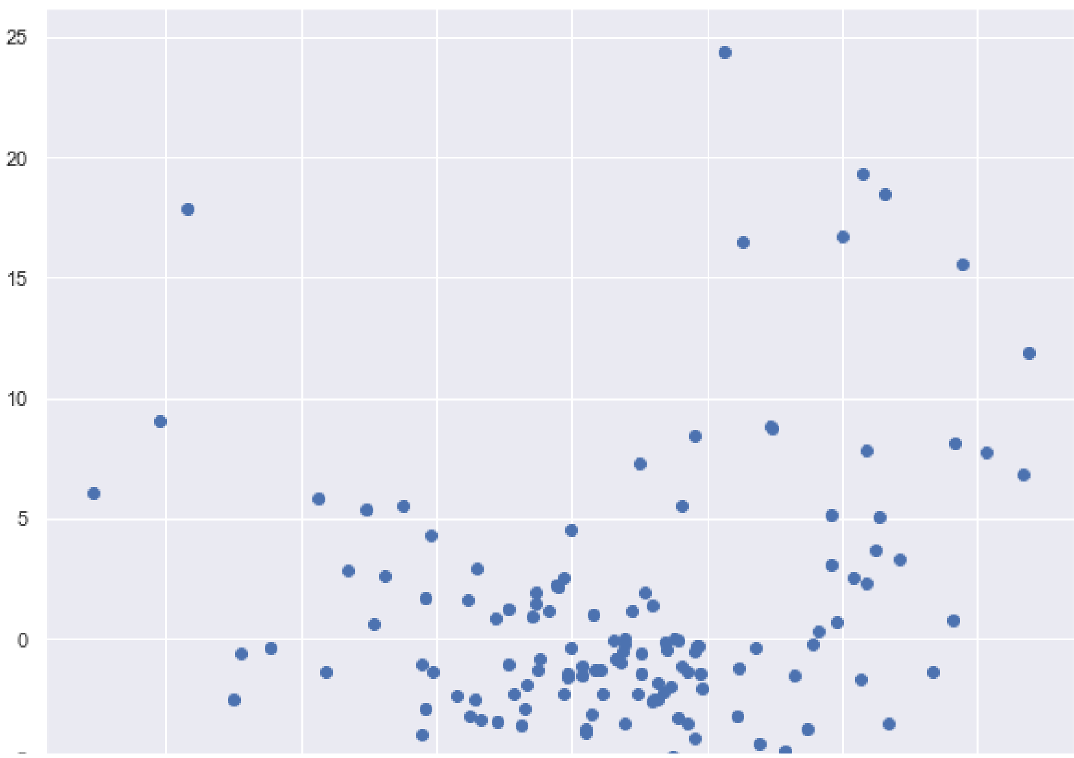


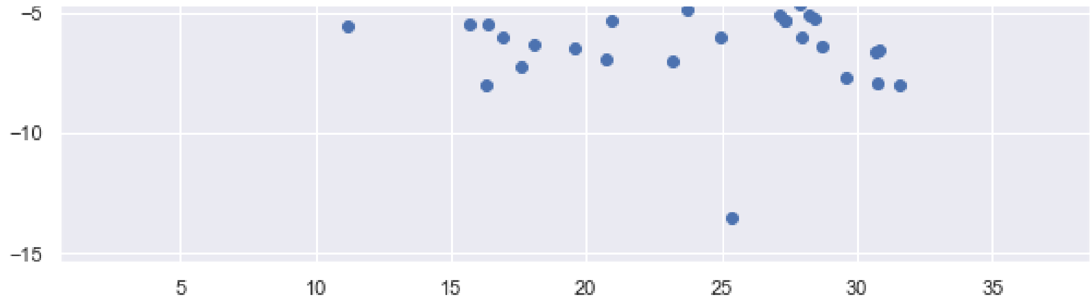
In [94]:

```
## SCatter plot with predictions and residual
##uniform distribution
plt.scatter(lasso_pred,lasso_residuals)
```

Out[94]:

```
<matplotlib.collections.PathCollection at 0x17fba74c760>
```





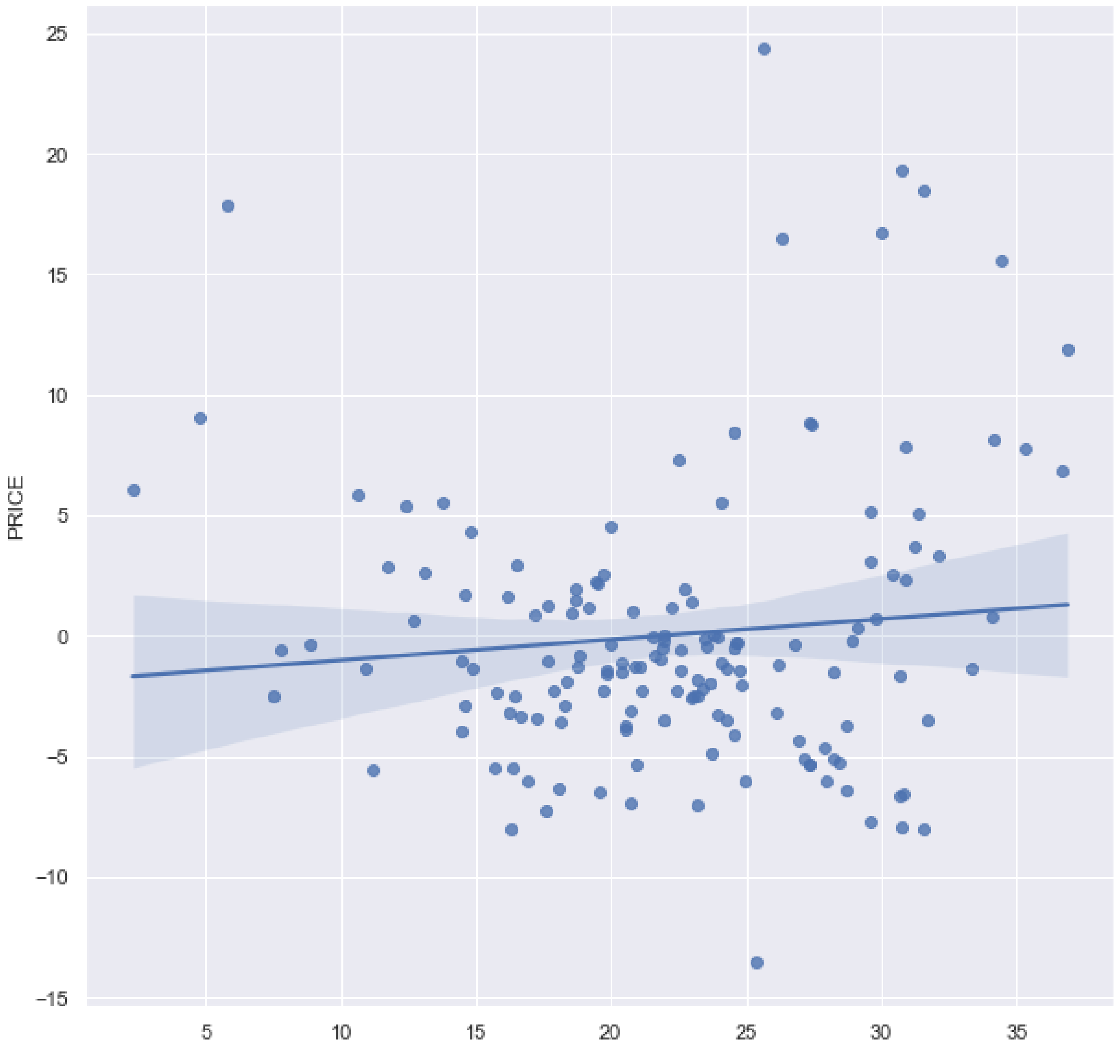
In [95]:

```
sns.regplot(lasso_pred,lasso_residuals)
```

```
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_de
corators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From v
ersion 0.12, the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[95]:

```
<AxesSubplot:ylabel='PRICE'>
```



In [96]:

```
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)))
```

```
33.31417646922572
4.044140676389867
5.771843420366298
```

In [97]:

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

```
0.6068712534869765
```

In [98]:

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

Out[98]:

```
0.5698373860618365
```

Evaluating the ElasticNet Regression

In [99]:

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

In [100]:

```
elasticnet
```

Out[100]:

```
▾ ElasticNet
ElasticNet()
```

In [101]:

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

Out[101]:

```
▾ ElasticNet
ElasticNet()
```

In [102]:

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

```
41.43181147909337
```

In [103]:

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

```
[-0.04587096  0.07269826 -0.          0.         -0.          0.79596544
  0.03686871 -0.7795076   0.29552182 -0.01671115 -0.70696726  0.00824164
 -0.79346039]
```

In [104]:

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

In [105]:

```
elasticnet_pred
```

Out[105]:

```
array([27.26828373, 37.51032106, 23.5103766 , 25.16133916, 24.8620994 ,
        4.96350039, 13.75939807, 14.15667878, 27.93675223, 20.29567914,
```

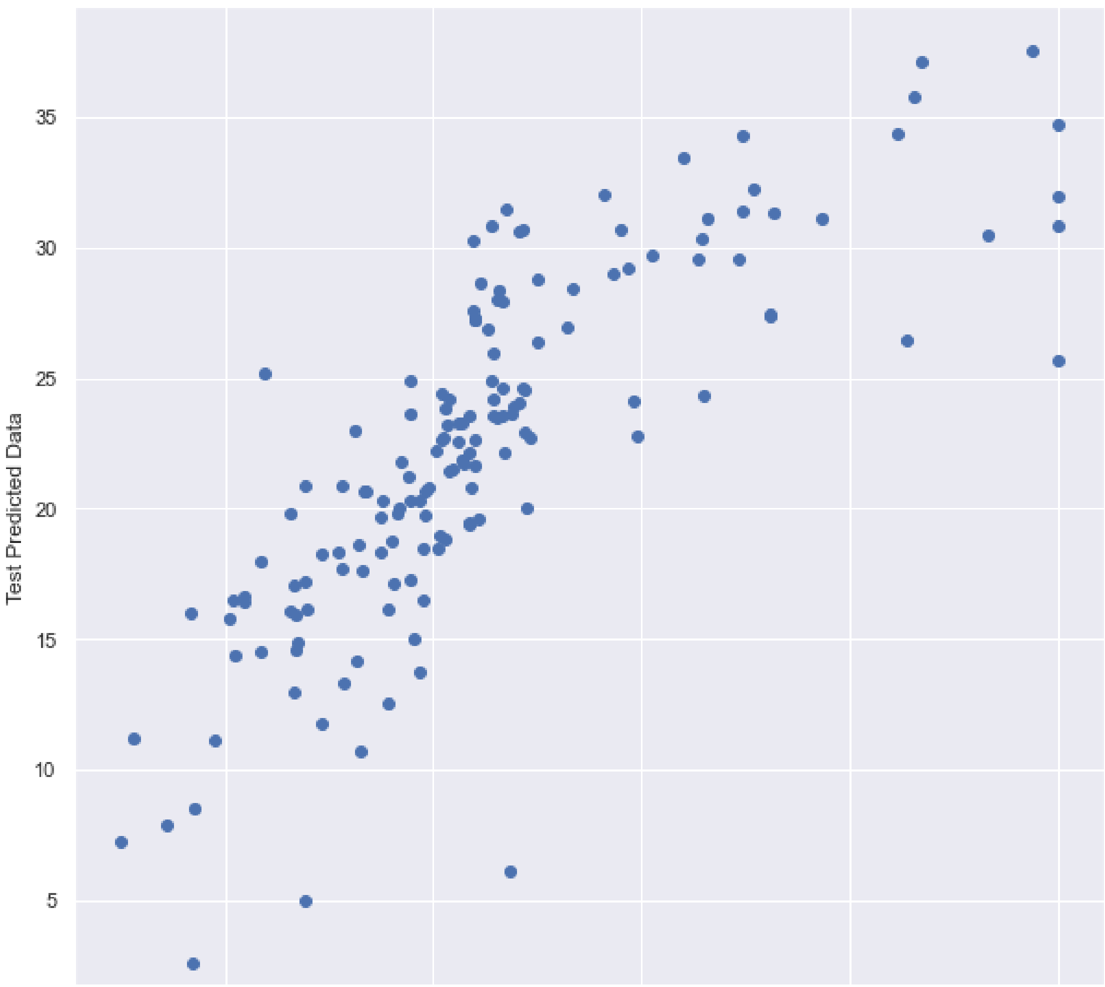
```
       17.11111674, 23.24198129, 18.29521516, 15.94525879, 21.63377027,
       16.40644521,  7.87286401,  7.22000653, 23.84317529, 10.73149498,
       37.09206294, 16.14321934, 21.46168756, 18.71686962, 22.66965331,
       21.75464982, 27.23463365, 20.62922864, 20.67907094, 23.58241666,
       24.58592064, 11.78176239, 20.80240837, 18.79055074, 22.951107  ,
       24.17318451, 23.54136541, 30.86092198, 19.808766  , 20.29363844,
       26.46814145, 31.09658387, 20.6554074 , 28.63220283, 22.81079176,
       23.5971913 , 21.19882451, 28.96528236, 28.01436765, 17.07321262,
       32.26511008, 20.85186443, 16.06946393, 16.03035783, 19.76871859,
       30.34973931,  8.49742367, 26.34728922, 12.96971314, 27.25290027,
       17.19984168, 27.43741824, 18.47001984, 30.68086412, 33.4438989 ,
       20.28502535, 23.64998184, 19.64133836, 25.98274388, 20.02589897,
       18.98966344, 24.53715483, 19.41113857, 14.84442847, 34.27319022,
       24.07758544, 35.79058396, 17.59623148, 20.89894275, 14.3681251 ,
       13.31437778, 15.81759266, 34.35110555, 24.3429957 , 31.92712315,
       18.59530486, 29.55954147, 23.26274262, 26.90558219, 22.98676213,
       29.53320464, 27.56199386, 30.2395347 , 24.87823579, 23.54384238,
       28.43667746, 17.27931099, 27.3885288 , 19.4826149 , 22.12587463,
       18.26416243, 11.20908887, 29.71976701,  2.60692363, 24.14034378,
       18.30165931, 22.13797235, 21.69294449, 20.83264279, 31.11039637,
       29.20630786, 21.89492996, 22.71580248, 25.65511909, 19.59974065,
       23.21237664, 28.75998786, 18.43942393, 14.54893282, 30.72271777,
       24.62367812, 11.14817335, 19.78183593, 30.81838114, 30.59386778,
       17.97997007, 22.66230113, 34.72279392, 30.46533435, 23.87706966,
       31.36784604, 24.37016055, 12.54069607, 17.67449394, 14.57784374,
       22.73312952, 31.35300892,  6.12363363, 31.4891038 , 28.38592949,
       24.17505825, 15.00700429, 21.5219948 , 26.93744597, 22.54457767,
       22.23913207, 20.00321318, 16.47854978, 32.03802161, 16.60981382,
       16.4738673 , 16.12693736])
```

In [106]:

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

Out[106]:

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



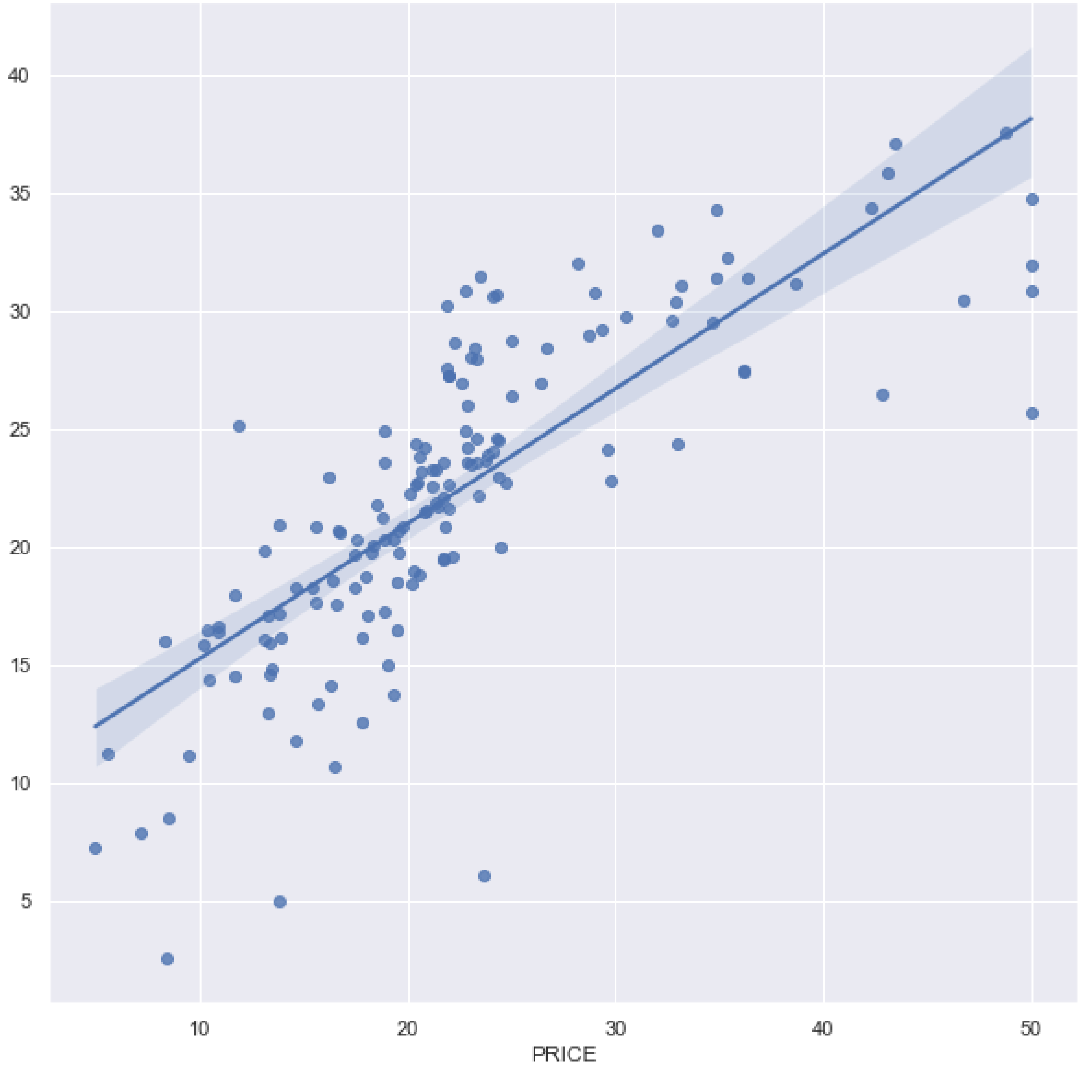
In [107]:

```
sns.regplot(y_test,elasticnet_pred)
```

```
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_de
corators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From v
ersion 0.12, the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[107]:

```
<AxesSubplot:xlabel='PRICE'>
```



In [108]:

```
## residuals
elasticnet_residuals=y_test-elasticnet_pred
elasticnet_residuals
```

Out[108]:

```
301     -5.268284
262     11.289679
172     -0.410377
505    -13.261339
111     -2.062099
          ...
380     -6.078550
307     -3.838022
381     -5.709814
106      3.026133
139      1.673063
```

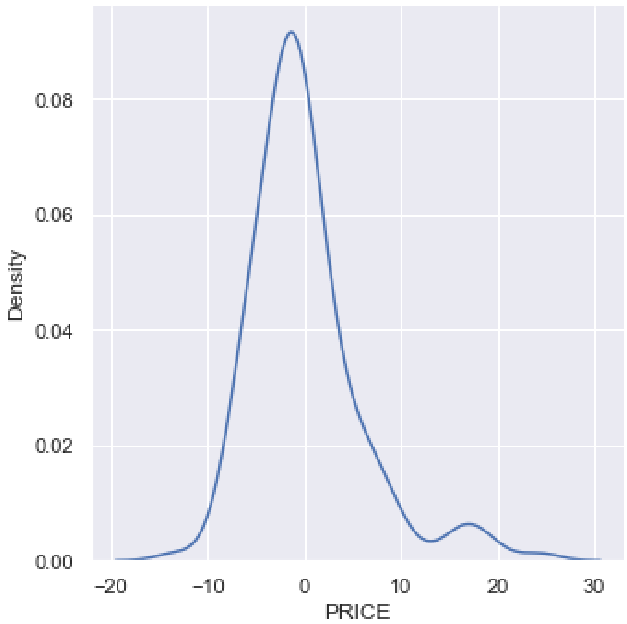
Name: PRICE, Length: 152, dtype: float64

In [109]:

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

Out[109]:

<seaborn.axisgrid.FacetGrid at 0x17fba7ae2f0>

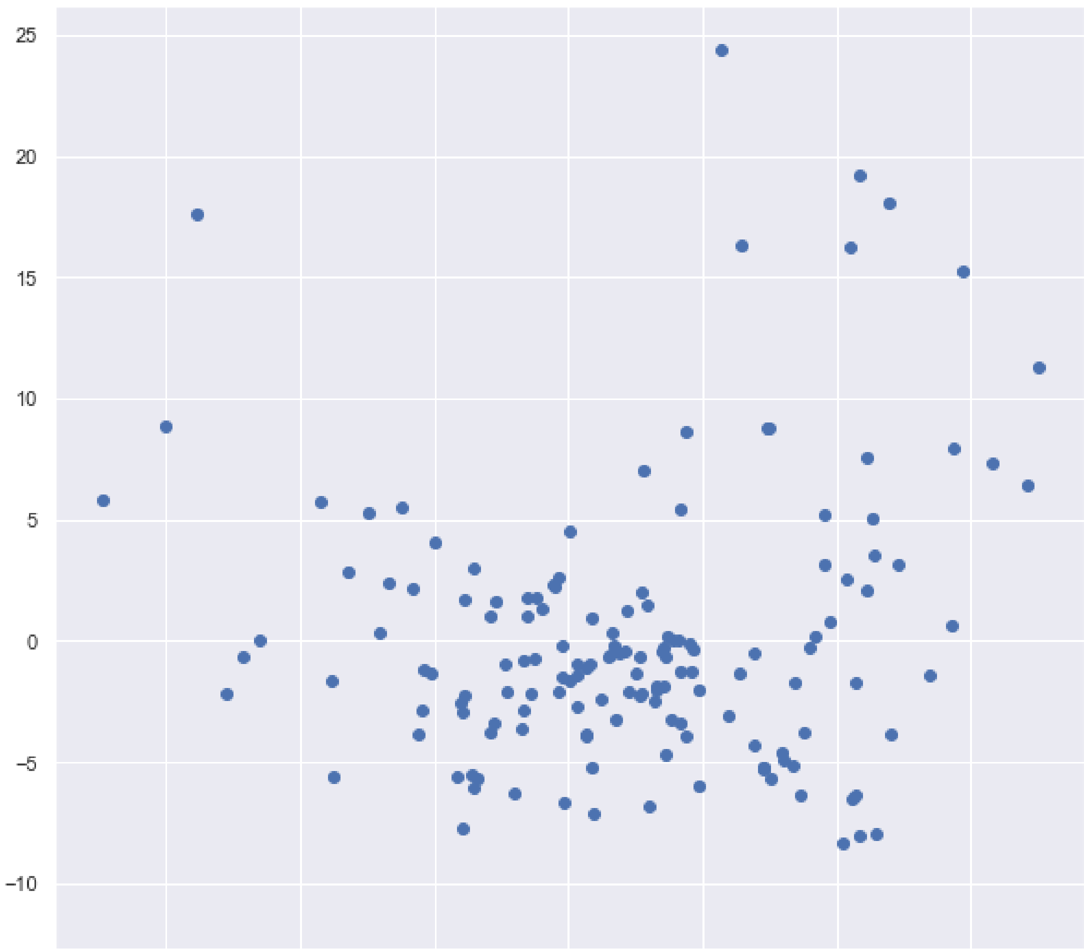


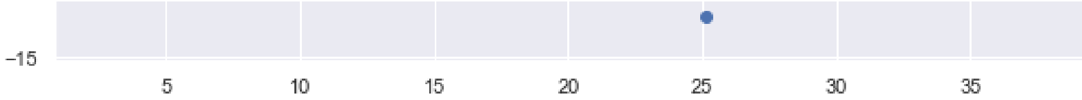
In [110]:

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

Out[110]:

<matplotlib.collections.PathCollection at 0x17fba83fb80>





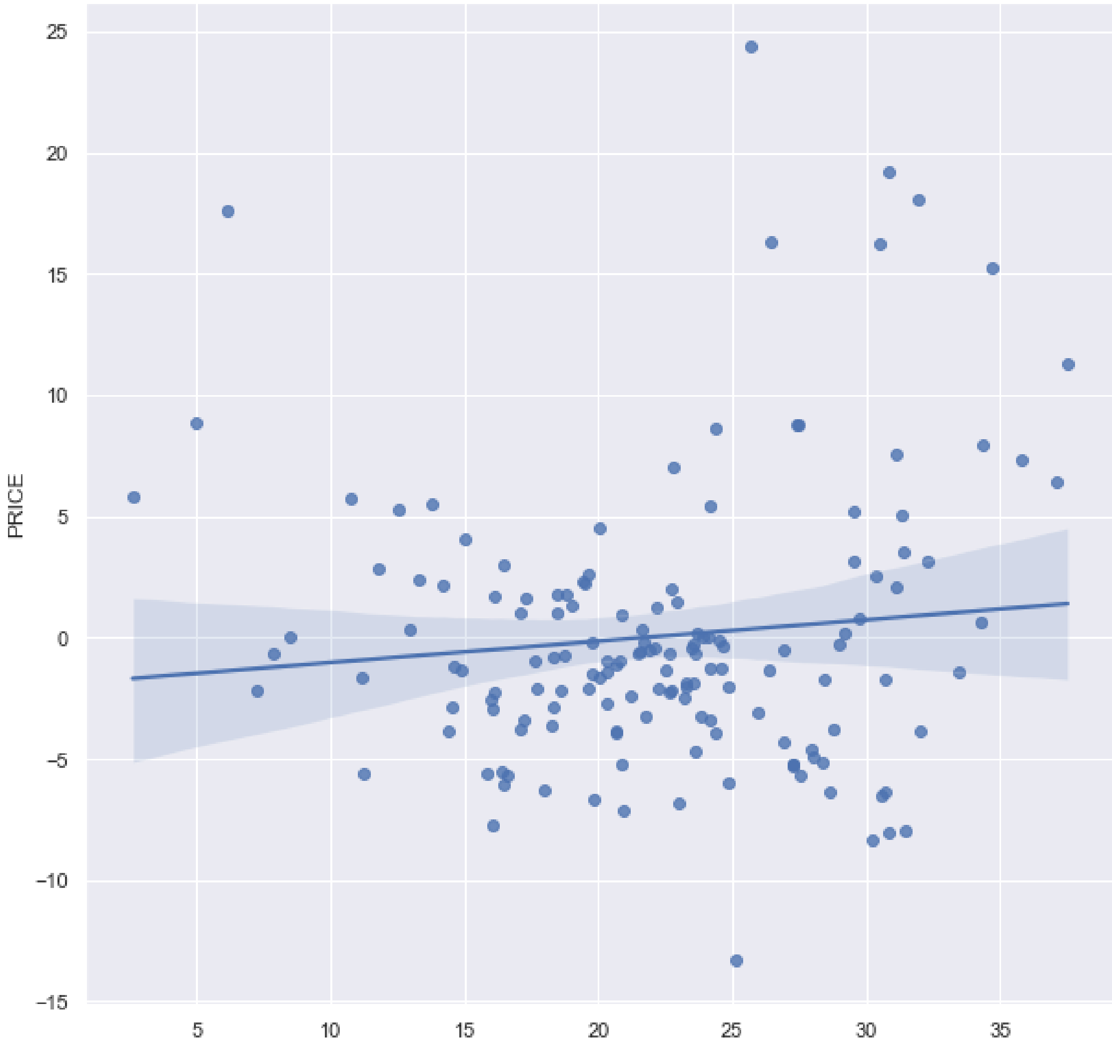
In [111]:

```
sns.regplot(elasticnet_pred,elasticnet_residuals)
```

```
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_de
corators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From v
ersion 0.12, the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[111]:

```
<AxesSubplot:ylabel='PRICE'>
```



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)))
```

```
32.36843282824359
3.9801854082590764
5.6893262191795255
```

In [113]:

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

```
0.6180316377889994
```

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]:

```
0.5820491109140501
```

In [ ]:

```
```