In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

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

In [3]:

```
boston=load_boston()
```

```
/usr/local/lib/python3.9/site-packages/sklearn/utils/deprecation.py:8
7: 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 abo
ut
    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 h
ousing
    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)
```

In [4]:

```
boston.keys()
```

Out[4]:

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

In [5]:

```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median V
alue (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 r
iver; 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 1
940
        - DIS      weighted distances to five Boston employment centre
s
        - 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 bla
ck 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/ (ht
tps://archive.ics.uci.edu/ml/machine-learning-databases/housing/)


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

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

The Boston house-price data has been used in many machine learning pap
ers that address regression
```

```
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influ
ential Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learn
ing. In Proceedings on the Tenth International Conference of Machine L
earning, 236-243, University of Massachusetts, Amherst. Morgan Kaufman
n.
```

In [6]:

```
print(boston.data)
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+0
0]
 [2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+0
0]
 [2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+0
0]
 ...
 [6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+0
0]
 [1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+0
0]
 [4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+0
0]]
```

In [7]:

```
print(boston.target) #Price of House
```

```
[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' 'PTRATI
O'
 'B' 'LSTAT']
```

Preparing dataframe for data

In [9]:

```
boston_data=pd.DataFrame(boston.data,columns=boston.feature_names)
```

In [10]:

```
boston_data.head()
```

Out[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.

In [11]:

```
boston.feature_names
```

Out[11]:

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RA
D',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [12]:

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

In [13]:

```
boston_data.head()
```

Out[13]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.

EDA

In [14]:

```
boston_data.info()
```

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

In [15]:

```
boston_data.describe()
```

Out[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12

In [16]:

```
boston_data.isnull().sum()
```

Out[16]:

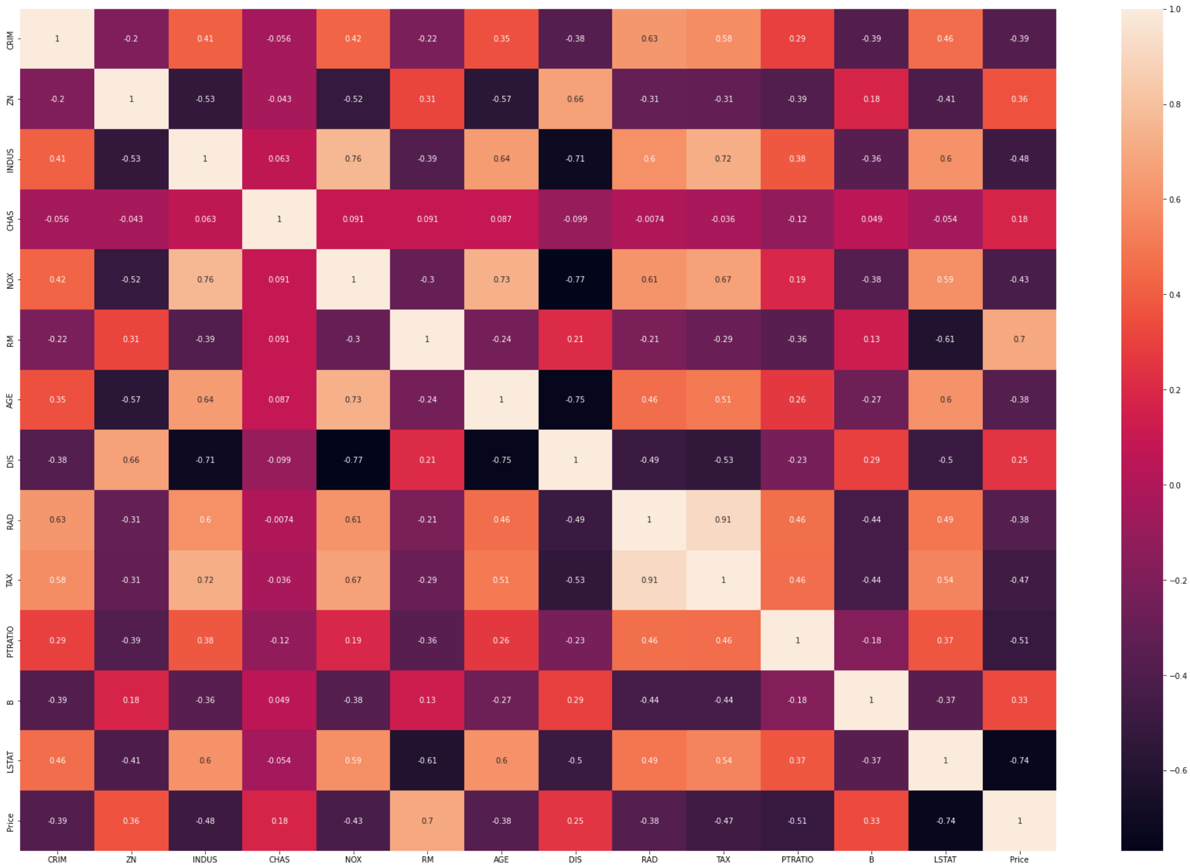
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
Price      0
dtype: int64
```

In [17]:

```
plt.figure(figsize=(30,20))
sns.heatmap(boston_data.corr(),annot=True)
```

Out[17]:

```
<AxesSubplot:>
```



In [18]:

```
boston_data.corr()
```

Out[18]:

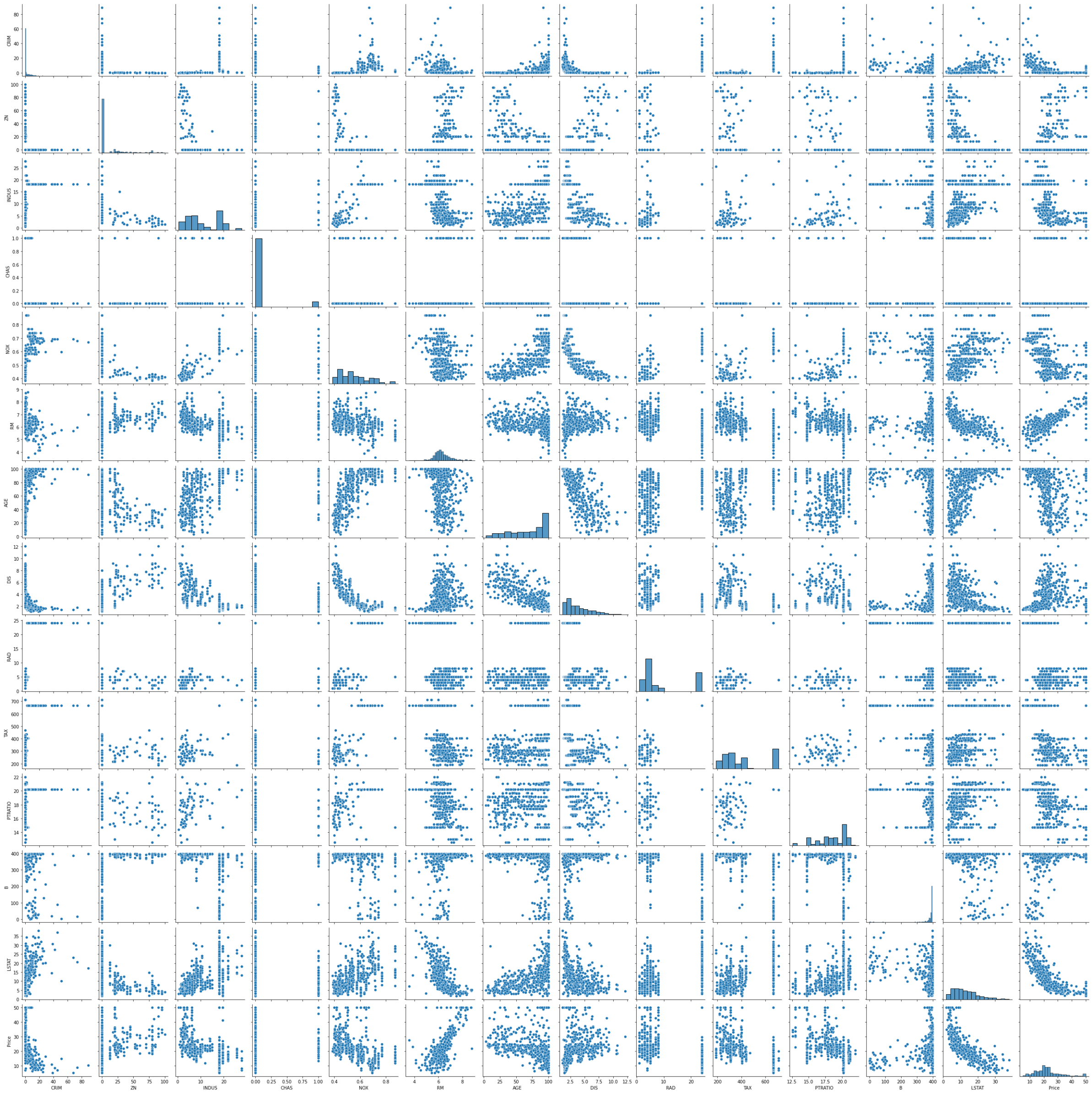
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027
CHAS	-0.055892	-0.042697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

In [19]:

```
sns.pairplot(boston_data)
```

Out[19]:

<seaborn.axisgrid.PairGrid at 0x133cb4e50>



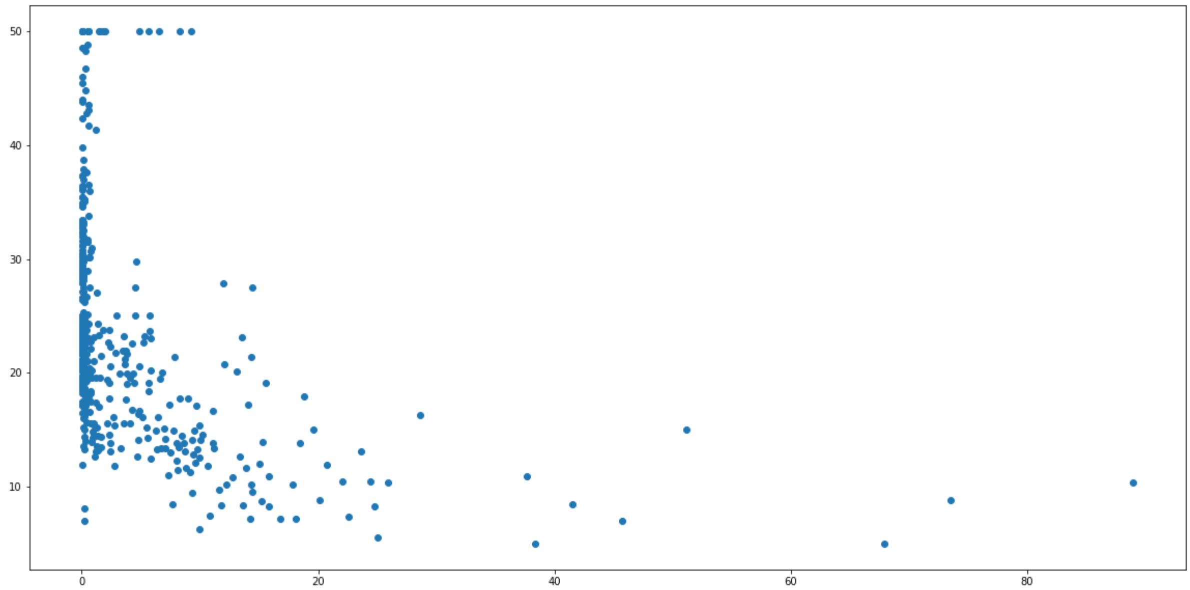
Explore the dependencies between dependent and independent features

In [20]:

```
plt.figure(figsize=(20,10))
plt.scatter(x=boston_data['CRIM'],y=boston_data['Price'])
```

Out[20]:

```
<matplotlib.collections.PathCollection at 0x138712d90>
```

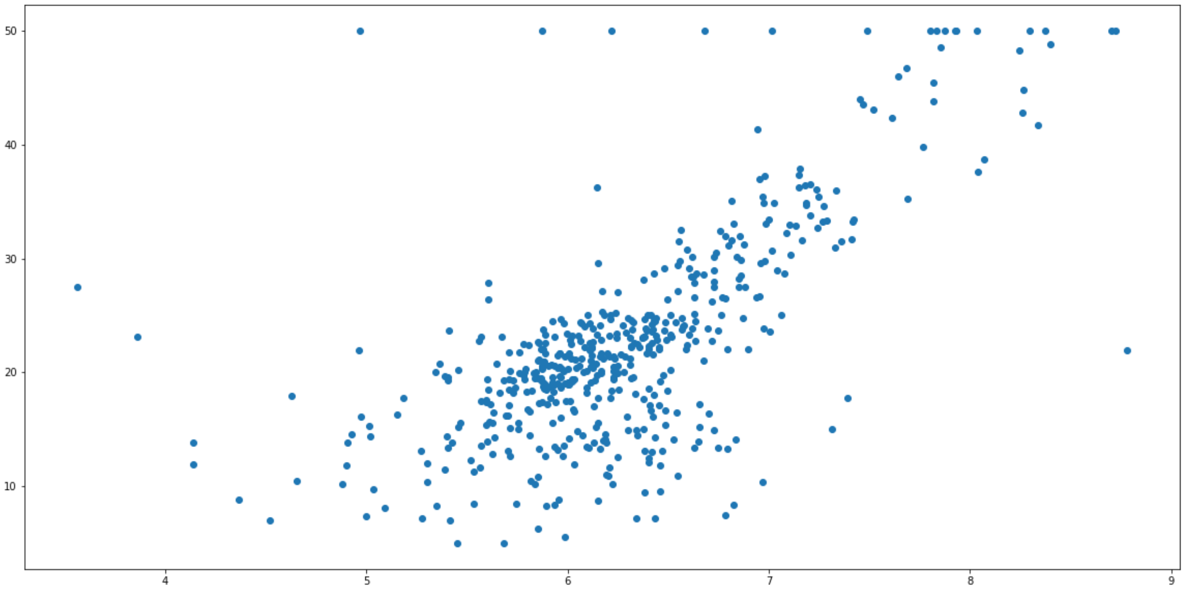


In [21]:

```
plt.figure(figsize=(20,10))
plt.scatter(x=boston_data['RM'],y=boston_data['Price'])
```

Out[21]:

```
<matplotlib.collections.PathCollection at 0x1387cddf0>
```



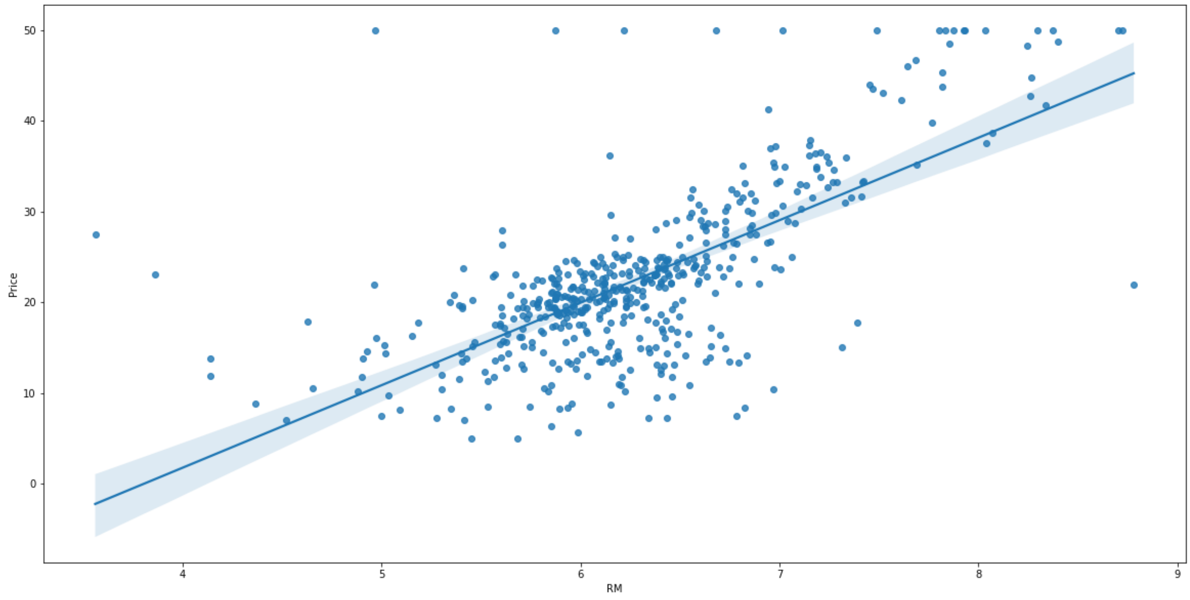
No of rooms (RM) is mostly correlated with Price

In [22]:

```
plt.figure(figsize=(20,10))
sns.regplot(x='RM',y='Price',data=boston_data)
```

Out[22]:

```
<AxesSubplot:xlabel='RM', ylabel='Price'>
```

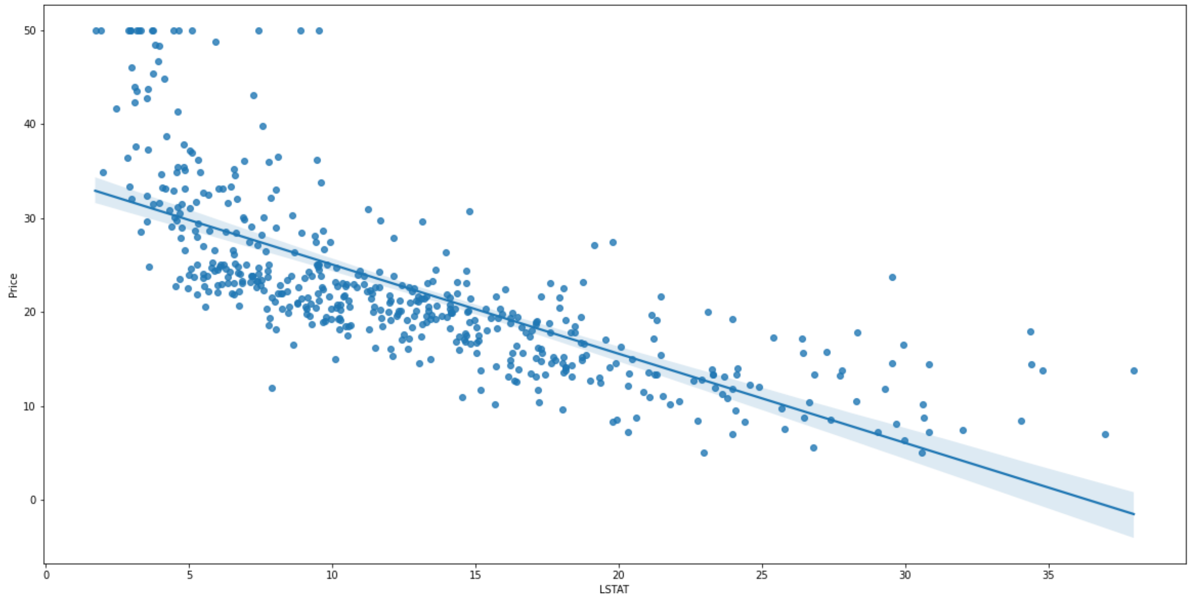


In [23]:

```
plt.figure(figsize=(20,10))
sns.regplot(x='LSTAT',y='Price',data=boston_data)
```

Out[23]:

```
<AxesSubplot:xlabel='LSTAT', ylabel='Price'>
```

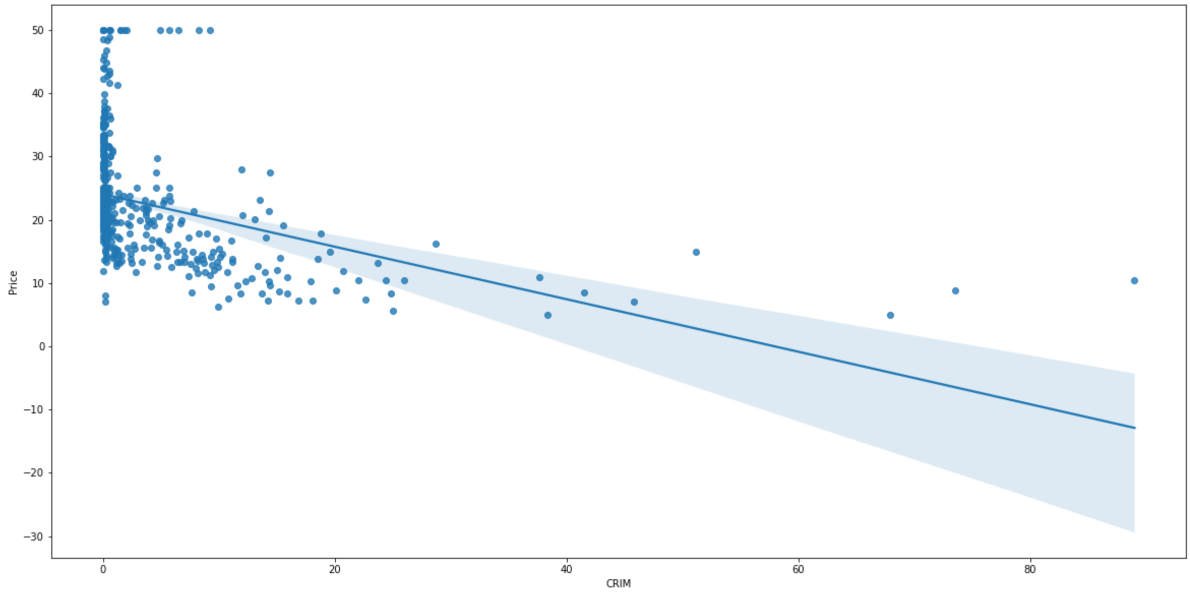


In [24]:

```
plt.figure(figsize=(20,10))
sns.regplot(x='CRIM',y='Price',data=boston_data)
```

Out[24]:

```
<AxesSubplot:xlabel='CRIM', ylabel='Price'>
```



Shaded Region is due to the Hyper parameter tuning in Ridge and Lasso Regression

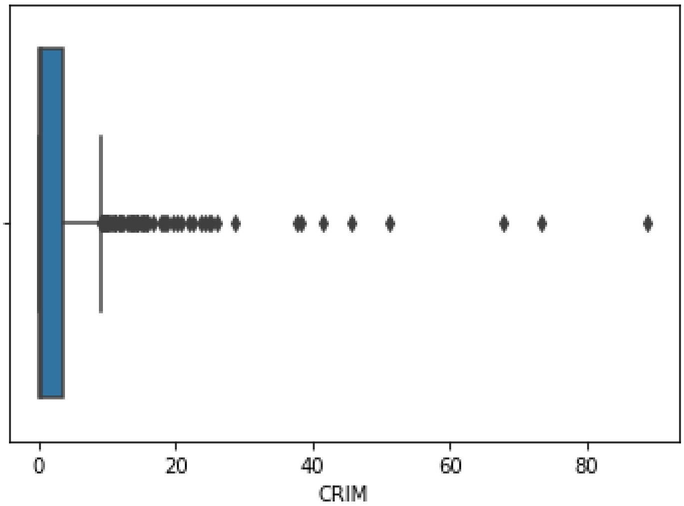
In [25]:

```
sns.boxplot(boston_data['CRIM'])
```

```
/usr/local/lib/python3.9/site-packages/seaborn/_decorators.py:36: Futu
reWarning: Pass the following variable as a keyword arg: x. From versi
on 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
```

Out[25]:

<AxesSubplot:xlabel='CRIM'>



In [26]:

```
## Independent and Dependent features

X=boston_data.iloc[:,:-1] ## Independent features will be in dataframe
Y=boston_data.iloc[:,-1]  ## Dependent feature will be in series
```

In [27]:

```
X.head()
```

Out[27]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.

In [28]:

```
Y.head()
```

Out[28]:

```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: Price, dtype: float64
```

In [29]:

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

In [30]:

```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.33, random_stat
```

In [31]:

```
X_train
```

Out[31]:

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

339 rows × 13 columns

In [32]:

```
X_train.shape
```

Out[32]:

```
(339, 13)
```

In [33]:

```
y_train.shape
```

Out[33]:

```
(339,)
```

In [34]:

```
X_test.shape
```

Out[34]:

```
(167, 13)
```

In [35]:

```
y_test.shape
```

Out[35]:

```
(167,)
```

Standardize the datasets or feature scaling

In [36]:

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

In [37]:

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

In [38]:

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

Linear Regression of the model

In [39]:

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

In [40]:

```
regression=LinearRegression()
```

In [41]:

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

Out[41]:

```
LinearRegression()
```

In [42]:

```
print(regression.coef_)
```

```
[-0.98858032  0.86793276  0.40502822  0.86183791 -1.90009974  2.808135
18
 -0.35866856 -3.04553498  2.03276074 -1.36400909 -2.0825356   1.041256
84
 -3.92628626]
```

In [43]:

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

```
22.970796460176988
```

In [44]:

```
## Prediction for the test data

reg_predict=regression.predict(X_test)
```

In [45]:

```
reg_predict
```

Out[45]:

```
array([28.53469469, 36.6187006 , 15.63751079, 25.5014496 , 18.7096734
,
        23.16471591, 17.31011035, 14.07736367, 23.01064388, 20.5422348
2,
        24.91632351, 18.41098052, -6.52079687, 21.83372604, 19.1490306
4,
        26.0587322 , 20.30232625,  5.74943567, 40.33137811, 17.4579144
6,
        27.47486665, 30.2170757 , 10.80555625, 23.87721728, 17.9949221
1,
        16.02608791, 23.268288  , 14.36825207, 22.38116971, 19.3092068
,
        22.17284576, 25.05925441, 25.13780726, 18.46730198, 16.6040571
2,
        17.46564046, 30.71367733, 20.05106788, 23.9897768 , 24.9432240
8,
        13.97945355, 31.64706967, 42.48057206, 17.70042814, 26.9250786
9,
        17.15897719, 13.68918087, 26.14924245, 20.2782306 , 29.9900349
2,
        21.21260347, 34.03649185, 15.41837553, 25.95781061, 39.1389727
4,
        22.96118424, 18.80310558, 33.07865362, 24.74384155, 12.8364095
8,
        22.41963398, 30.64804979, 31.59567111, 16.34088197, 20.9504304
,
        16.70145875, 20.23215646, 26.1437865 , 31.12160889, 11.8976276
8,
        20.45432404, 27.48356359, 10.89034224, 16.77707214, 24.0259371
4,
         5.44691807, 21.35152331, 41.27267175, 18.13447647,  9.8012101
,
        21.24024342, 13.02644969, 21.80198374,  9.48201752, 22.9918385
7,
        31.90465631, 18.95594718, 25.48515032, 29.49687019, 20.0728253
9,
        25.5616062 ,  5.59584382, 20.18410904, 15.08773299, 14.3456211
7,
        20.85155407, 24.80149389, -0.19785401, 13.57649004, 15.6440167
9,
        22.03765773, 24.70314482, 10.86409112, 19.60231067, 23.7342916
1,
        12.08082177, 18.40997903, 25.4366158 , 20.76506636, 24.6858823
7,
         7.4995836 , 18.93015665, 21.70801764, 27.14350579, 31.9376520
8,
        15.19483586, 34.01357428, 12.85763091, 21.06646184, 28.5847004
2,
        15.77437534, 24.77512495,  3.64655689, 23.91169589, 25.8229292
5,
        23.03339677, 25.35158335, 33.05655447, 20.65930467, 38.1891736
1,
        14.04714297, 25.26034469, 17.6138723 , 20.60883766,  9.8525544
,
        21.06756951, 22.20145587, 32.2920276 , 31.57638342, 15.2926593
```

```
8,
        16.7100235 , 29.10550932, 25.17762329, 16.88159225,  6.3262187
7,
        26.70210263, 23.3525851 , 17.24168182, 13.22815696, 39.4990750
7,
        16.53528575, 18.14635902, 25.06620426, 23.70640231, 22.2016777
2,
        21.22272327, 16.89825921, 23.15518273, 28.69699805,  6.6552648
2,
        23.98399958, 17.21004545, 21.0574427 , 25.01734597, 27.6546185
9,
        20.70205823, 40.38214871])
```

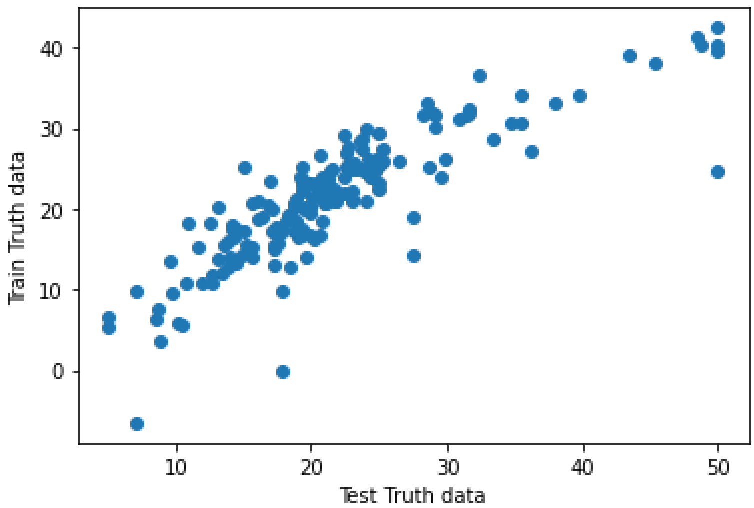
Assumptions on Linear Regression

In [46]:

```
plt.scatter(y_test,reg_predict)
plt.xlabel("Test Truth data")
plt.ylabel("Train Truth data")
```

Out[46]:

```
Text(0, 0.5, 'Train Truth data')
```



Linear Relationship so the model is good

In [47]:

```
residual=y_test-reg_predict
```

In [48]:

```
residual
```

Out[48]:

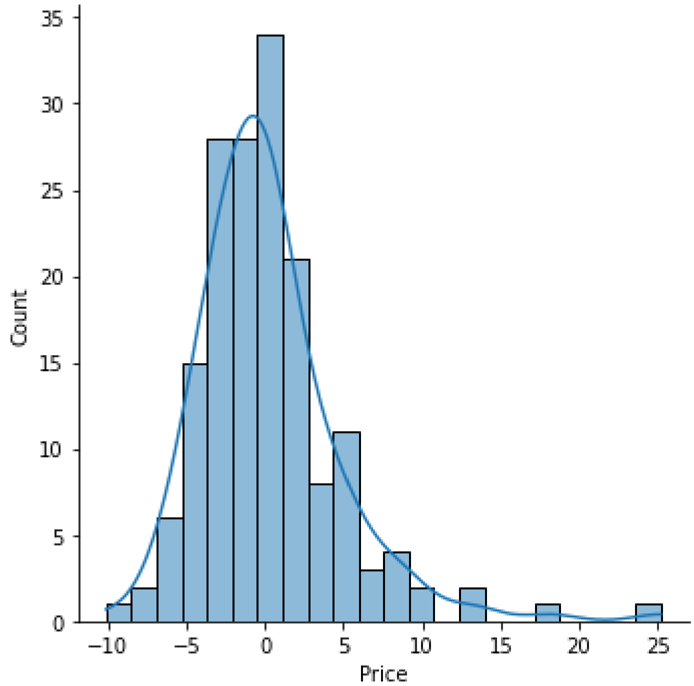
```
173   -4.934695
274   -4.218701
491   -2.037511
72    -2.701450
452   -2.609673
         ...
110    0.642557
321   -1.917346
265   -4.854619
29     0.297942
262    8.417851
Name: Price, Length: 167, dtype: float64
```

In [49]:

```
sns.displot(residual,kde=True)
```

Out[49]:

```
<seaborn.axisgrid.FacetGrid at 0x138891310>
```



Distribution plot should be approximately like a normal distribution

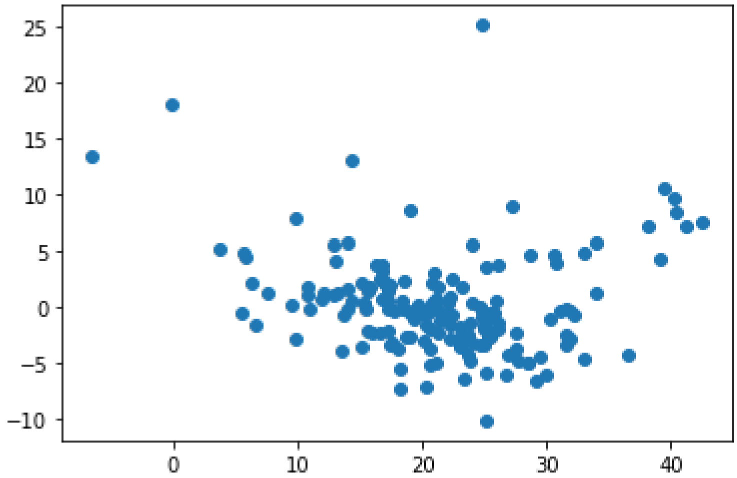
In [50]:

```
## Scatter plot with prediction and residuals

plt.scatter(x=reg_predict,y=residual)
```

Out[50]:

```
<matplotlib.collections.PathCollection at 0x13afe8880>
```



No shapes for the result it is called as uniform distribution

In [51]:

```
### Performance Metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

In [52]:

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

```
20.724023437339753
3.1482557548168324
4.552364598463062
```

In [88]:

```
## R Squared error and adjusted R Square

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

Out[88]:

```
0.7261570836552476
```

In [89]:

```
###Adjuested R square

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

Out[89]:

```
0.7028893848808568
```

Ridge Regression

In [55]:

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

In [56]:

```
ridge.fit(X_test,y_test)
```

Out[56]:

```
Ridge()
```

In [57]:

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

```
22.915881340648l3
```

In [58]:

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

```
[-0.81895266  1.28146367 -0.6351135   0.13033418 -2.32624176  2.608731
69
  0.58445046 -3.36326457  3.49484598 -3.02841669 -2.16311653  0.396169
6
 -3.21808869]
```

In [59]:

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

In [60]:

```
ridge_predict
```

Out[60]:

```
array([30.13546806, 33.69074127, 10.54760167, 22.90382486, 18.1794841
3,
       22.57378855, 18.46486352, 12.94387181, 22.47756466, 20.6475581
3,
       23.64761515, 19.96657389, -1.09423529, 20.35289663, 20.1559944
3,
       24.36816595, 20.50269304,  7.66311848, 41.34848691, 17.3002048
,
       26.73235288, 30.44014572, 13.17320375, 23.02439572, 18.0811209
3,
       16.70577326, 20.99488517, 16.77653781, 21.32631104, 19.0913124
5,
       24.71247364, 24.65763849, 24.69697677, 17.47831213, 17.4663802
1,
       16.95179946, 30.19158864, 20.60983885, 21.56562326, 22.5436453
7,
       14.66096614, 34.44852133, 43.80677765, 17.2968139 , 27.7341768
1,
       17.03644827, 15.74762414, 23.58433421, 19.70482784, 30.1347845
,
       22.26591651, 34.67134962, 16.97526754, 25.11146121, 39.4036138
5,
       20.59134522, 18.39589852, 33.63003863, 25.10230671, 14.7031864
8,
       22.81474509, 31.68365205, 30.59683424, 16.33574052, 19.9206825
8,
       16.63852152, 20.07321706, 25.90236435, 31.54573985, 14.1871276
,
       20.68969917, 25.78443805, 12.48578167, 16.31897139, 22.2328353
8,
        7.59482875, 21.85363697, 42.76888169, 18.58895769,  5.7890739
4,
       20.20273359, 14.9135559 , 21.47434704, 10.50068162, 23.8375553
8,
       30.15144323, 19.0757619 , 24.54468401, 27.91360889, 18.0010627
5,
       26.01215273,  8.74511938, 17.8295459 , 14.76182475, 13.7429416
6,
       18.43795068, 24.51128012,  5.14299999, 15.898517  , 18.9770022
9,
       22.78068194, 22.67742684, 12.42727666, 18.96129877, 22.7413329
8,
       15.21582374, 18.65583337, 23.44779519, 20.58997357, 25.0419446
2,
       11.12558185, 21.03240092, 21.45244163, 29.08536876, 33.0863223
6,
       16.88077737, 36.02108432, 14.74603774, 20.1790386 , 28.1126862
7,
       17.39037148, 23.25371808,  8.22577034, 23.42684788, 24.2318000
4,
       22.68398502, 24.94485069, 35.28025531, 17.94126472, 39.5459327
7,
       15.4603437 , 23.00165651, 19.03001212, 21.22119801,  9.8872638
5,
       22.02625499, 22.34372185, 34.00467362, 31.37834923, 16.6974073
```

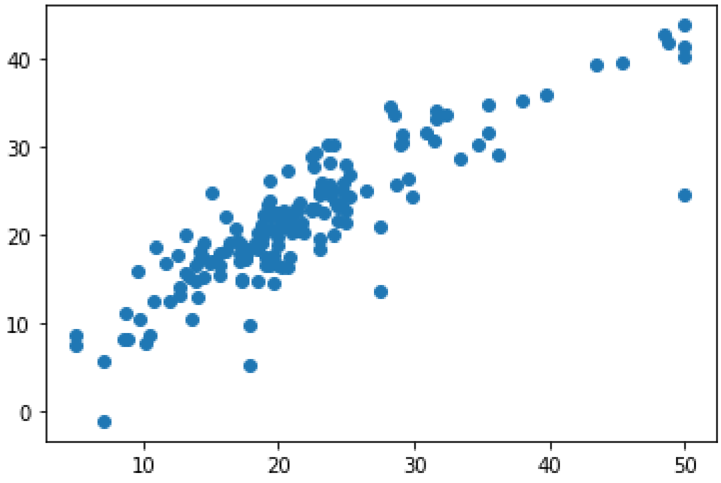
```
9,
       17.59352966, 28.96497792, 26.06830368, 16.54665221,  8.1865867
8,
       27.18010951, 19.35714454, 17.12907526, 15.26550829, 40.1267016
5,
       17.82708456, 17.71827252, 25.72484091, 23.02013087, 22.0932510
6,
       19.52130904, 16.60597912, 21.36800047, 28.53448396,  8.7375810
2,
       26.4818624 , 19.13502094, 20.35553391, 24.66488255, 29.3730161
3,
       21.37475628, 41.825846  ])
```

In [61]:

```
plt.scatter(y_test,ridge_predict)
```

Out[61]:

<matplotlib.collections.PathCollection at 0x13b0593d0>



In [62]:

```
ridge_residuals=y_test-ridge_predict
```

In [63]:

```
ridge_residuals
```

Out[63]:

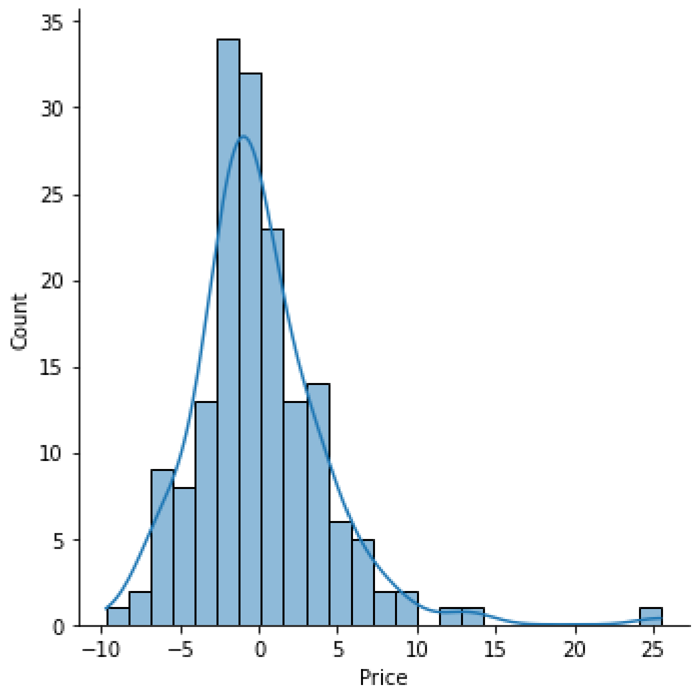
```
173   -6.535468
274   -1.290741
491    3.052398
72    -0.103825
452   -2.079484
         ...
110    1.344466
321   -1.564883
265   -6.573016
29    -0.374756
262    6.974154
Name: Price, Length: 167, dtype: float64
```

In [64]:

```
sns.displot(ridge_residuals,kde=True)
```

Out[64]:

```
<seaborn.axisgrid.FacetGrid at 0x13af04970>
```

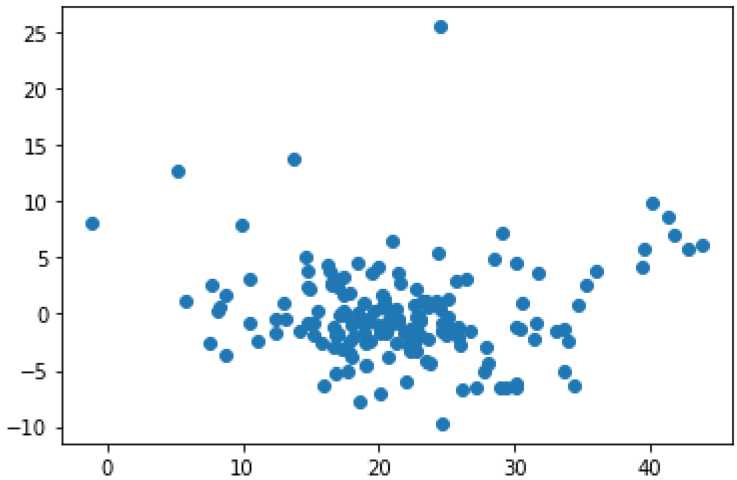


In [65]:

```
plt.scatter(x=ridge_predict,y=ridge_residuals)
```

Out[65]:

```
<matplotlib.collections.PathCollection at 0x13b146b50>
```



In [66]:

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

```
17.812451949233274
2.9562384184655244
4.220480061466145
```

In [85]:

```
## R Squared error and adjusted R Square

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

Out[85]:

```
0.7646299810566635
```

In [86]:

```
###Adjuested R square

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

Out[86]:

```
0.7446312212771643
```

Lasso Regression

In [67]:

```
from sklearn.linear_model import Lasso
```

In [68]:

```
lasso=Lasso()
```

In [70]:

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

Out[70]:

```
Lasso()
```

In [72]:

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

```
[-0.          0.         -0.          0.27140271 -0.          2.629321
47
 -0.         -0.         -0.         -0.         -1.21106809  0.298726
25
 -3.81788375]
```

In [74]:

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

22.970796460176988

In [75]:

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

In [76]:

```
lasso_predict
```

Out[76]:

```
array([26.08015466, 30.7480057 , 17.78164882, 25.25224684, 19.2838727
4,
       22.81161765, 18.31125182, 14.6359243 , 21.41277818, 20.4427665
9,
       20.7857368 , 21.00978479,  1.29101416, 22.48591111, 20.4207989
,
       24.73115299, 18.16643043,  6.95747132, 35.82658816, 18.4566435
8,
       25.66618031, 26.77096265, 13.79601995, 24.00317031, 18.8367757
5,
       15.53225538, 22.93567982, 18.81410882, 19.96419904, 19.7139455
4,
       19.9929271 , 25.48086778, 25.07506471, 19.62299031, 15.8716444
2,
       20.47826644, 30.90020658, 21.73740698, 21.69357896, 24.7879514
1,
       14.48946282, 27.49872616, 36.28097645, 19.68302782, 25.5469591
8,
       17.26691093, 16.01035524, 25.87512519, 19.3705841 , 29.5296518
3,
       23.10173719, 31.37342903, 17.55332715, 25.82107048, 34.9885719
9,
       22.91267519, 19.3967501 , 29.34678421, 24.65125376, 16.7297165
8,
       25.42537393, 30.6751849 , 28.90511192, 18.42571639, 27.5642663
9,
       14.62706882, 20.02272756, 25.60745002, 28.32959623, 15.9197130
7,
       20.36020491, 26.04012236, 13.70562148, 23.19186499, 23.2538407
,
        9.14791655, 21.08680468, 35.13203126, 18.20120981, 12.4057912
6,
       23.03574753, 11.70030485, 24.10234373, 10.23869501, 22.2478844
6,
       28.20852115, 20.77401763, 26.01572261, 25.97666619, 20.7747168
8,
       24.05595237,  9.79658092, 21.55718522, 20.96232324, 14.5894139
7,
       22.29462592, 23.04513053,  2.87810564, 18.26028545, 17.3140528
4,
       21.55660947, 24.48282506, 11.46772233, 21.88129799, 25.0434920
7,
       14.07796126, 19.97841644, 26.61705358, 23.30429098, 27.3273603
5,
       12.59741065, 19.28050072, 24.94727892, 24.22470232, 29.7251931
4,
       19.11634391, 31.14895846, 16.43050137, 20.50890111, 27.6902697
8,
       19.80307948, 26.66386801, 15.01321139, 23.31466084, 26.1544601
8,
       23.80801526, 27.15999771, 30.37432077, 22.93935948, 34.9115986
5,
       11.97264266, 26.45153342, 20.25377754, 19.96681079, 12.2167763
5,
       21.57200937, 23.11587937, 31.05309711, 29.72484228, 18.0366938
```

```
7,
        19.12012649, 28.8679226 , 23.41788443, 14.36679948, 10.9143384
9,
        23.78530314, 23.58885901, 18.48704182, 15.72569049, 36.3867166
,
        19.38880373, 19.56932184, 27.03174387, 22.95041998, 22.0780790
6,
        23.22702436, 17.41528186, 24.66493926, 30.31735718, 13.9998893
,
        22.25446895, 19.75004517, 20.8350658 , 25.47389672, 24.1357763
3,
        23.02944605, 36.90324597])
```

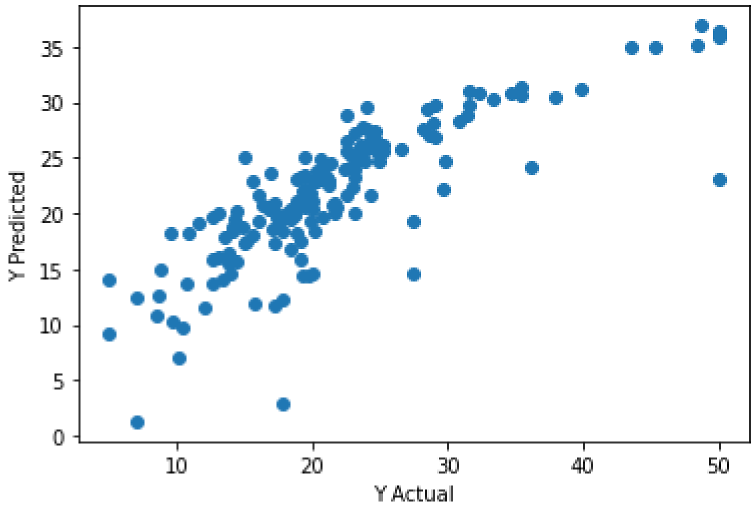
Assumptions on Lasso Regression

In [78]:

```
# Scatter Plot for Actual and Predicted Values
plt.scatter(x=y_test,y=lasso_predict)
plt.xlabel("Y Actual")
plt.ylabel("Y Predicted")
```

Out[78]:

```
Text(0, 0.5, 'Y Predicted')
```



In [79]:

```
# Residuals for Actual and Predicted Values
lasso_residuals=y_test-lasso_predict
```

In [80]:

```
lasso_residuals
```

Out[80]:

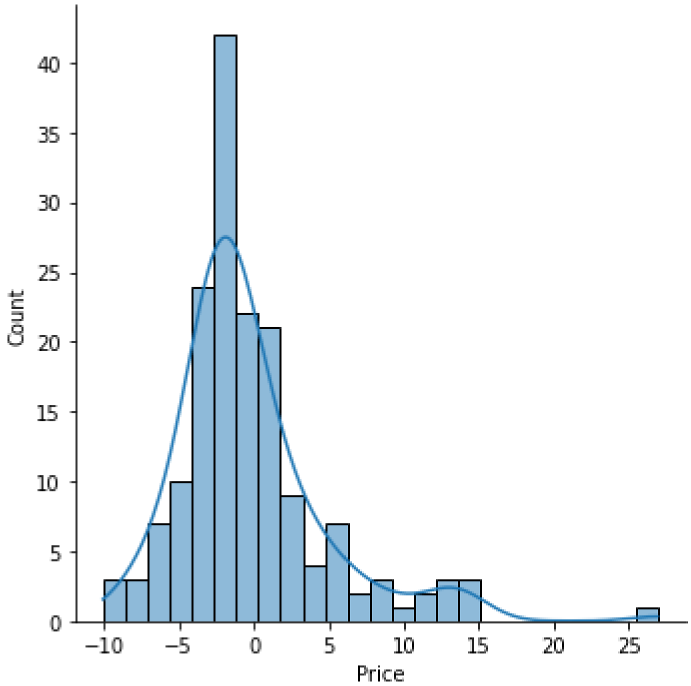
```
173    -2.480155
274     1.651994
491    -4.181649
72     -2.452247
452    -3.183873
          ...
110     0.864934
321    -2.373897
265    -1.335776
29     -2.029446
262    11.896754
Name: Price, Length: 167, dtype: float64
```

In [81]:

```
sns.displot(lasso_residuals,kde=True)
```

Out[81]:

```
<seaborn.axisgrid.FacetGrid at 0x13bf0dcd0>
```

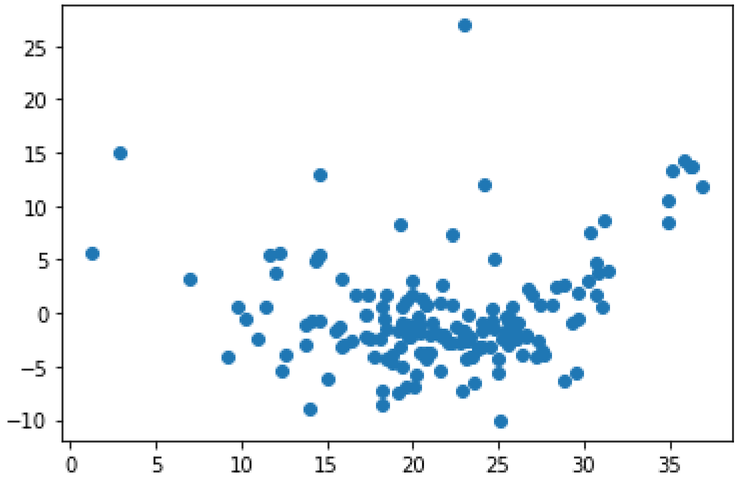


In [82]:

```
plt.scatter(x=lasso_predict,y=lasso_residuals)
```

Out[82]:

```
<matplotlib.collections.PathCollection at 0x13c27a7c0>
```



In [84]:

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

```
26.16637721498099
3.6464026430077423
5.11530812512609
```

In [90]:

```
## R Squared error and adjusted R Square

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

Out[90]:

```
0.6542429577734992
```

In [91]:

```
###Adjuested R square

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

Out[91]:

```
0.6248649084339926
```

Elastic Net Regression

In [92]:

```
from sklearn.linear_model import ElasticNet
```

In [93]:

```
elastic=ElasticNet()
```

In [94]:

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

Out[94]:

```
ElasticNet()
```

In [95]:

```
elastic.coef_
```

Out[95]:

```
array([-0.36520114,  0.        , -0.14336748,  0.63145824, -0.2519314
8,
        2.34999448, -0.        , -0.        , -0.        , -0.2564996
9,
       -1.23951556,  0.56384945, -2.56053213])
```

In [97]:

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

In [98]:

```
elastic_pred
```

Out[98]:

```
array([26.04802695, 31.11448131, 18.09845158, 24.74715491, 19.1302971
3,
       23.07195028, 19.8492127 , 16.42921582, 20.98280883, 21.0304090
5,
       23.59247585, 22.4067143 ,  2.50342106, 22.86968897, 21.0583647
7,
       23.53088819, 19.32942155,  9.24659633, 34.51755093, 18.3311198
2,
       25.39963891, 26.53220506, 16.04212388, 23.68595117, 18.2230960
9,
       15.9070075 , 22.91791506, 17.40135861, 22.80881602, 20.3496007
2,
       21.28107265, 25.0664737 , 23.29041734, 18.52289666, 16.6894671
9,
       20.17099878, 29.78000437, 22.08911412, 24.00624402, 24.5210960
1,
       16.51539744, 27.25142517, 34.8940966 , 20.75229792, 25.5494436
2,
       17.27877681, 17.51067948, 25.422475  , 19.45141801, 28.7244543
1,
       23.85816391, 30.64335445, 19.05778782, 25.10137208, 33.4367358
7,
       21.9368327 , 19.10068361, 28.38705767, 24.91075492, 18.6882115
8,
       25.41735754, 29.96236233, 27.77368373, 18.66077461, 26.8345677
6,
       18.72984267, 19.66634919, 25.37569386, 27.64862833, 15.0932688
7,
       21.6230625 , 24.42218348, 14.00935207, 22.80340884, 23.3105703
7,
       10.15388121, 21.41270277, 33.98445117, 18.23197774, 13.8363012
7,
       23.23840504, 13.33915878, 24.55297788, 11.80396525, 22.6039322
,
       29.04777925, 19.93864988, 25.40796591, 25.4253774 , 20.7962663
4,
       24.35937771,  9.98031645, 21.1883148 , 21.7010251 , 13.6415836
6,
       21.70329066, 21.48780024,  4.89921219, 16.60651403, 16.4869136
4,
       22.52662793, 24.23810699, 12.67234464, 21.73288769, 24.9486962
2,
       14.02785155, 20.34560161, 25.81816846, 23.27665603, 26.9896808
3,
       12.4461064 , 18.53919342, 24.57036836, 24.5991159 , 28.7891728
9,
       17.35695925, 30.55890904, 17.49669771, 20.81635985, 27.2666873
5,
       20.67056474, 26.10145269, 11.29182557, 23.22360335, 25.7679046
4,
       23.6220014 , 26.68354582, 29.53505955, 23.09099441, 33.6906031
5,
       14.86234562, 26.00700282, 20.72314768, 21.04261631, 14.5858345
3,
       19.80159048, 23.16541552, 30.31837307, 29.06727633, 19.2166498
```

```
7,
        19.96431131, 28.26947089, 24.04394758, 18.60022435, 10.9739204
2,
        23.98834146, 24.56611841, 18.41001674, 17.5657926 , 34.5653424
2,
        18.12003269, 19.01050052, 26.55906015, 23.10161055, 22.6912207
1,
        22.66142713, 17.29393495, 22.68213782, 29.22756893, 12.1400180
6,
        23.24525465, 20.77586134, 21.27148575, 25.02037796, 24.4450181
4,
        23.10652629, 35.30973171])
```

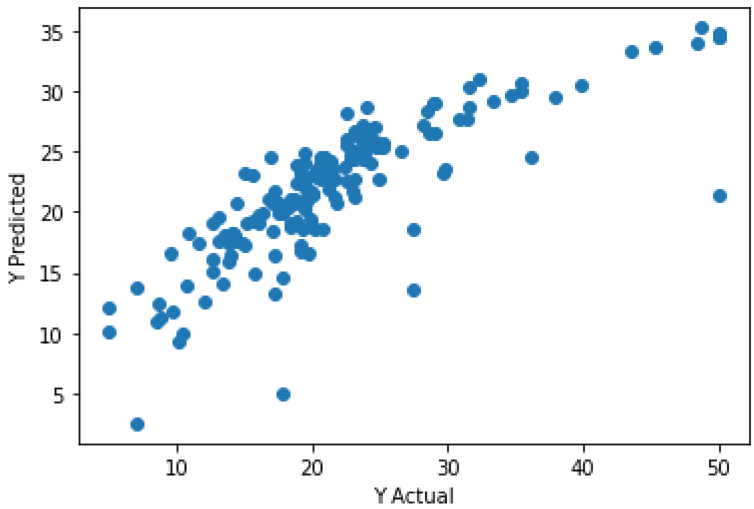
Assumptions on Elastic Net Regression

In [100]:

```
# Scatter Plot for Actual and Predicted Values
plt.scatter(x=y_test,y=elastic_pred)
plt.xlabel("Y Actual")
plt.ylabel("Y Predicted")
```

Out[100]:

```
Text(0, 0.5, 'Y Predicted')
```



In [101]:

```
# Residuals for Actual and Predicted Values
elastic_residuals=y_test-elastic_pred
```

In [102]:

```
elastic_residuals
```

Out[102]:

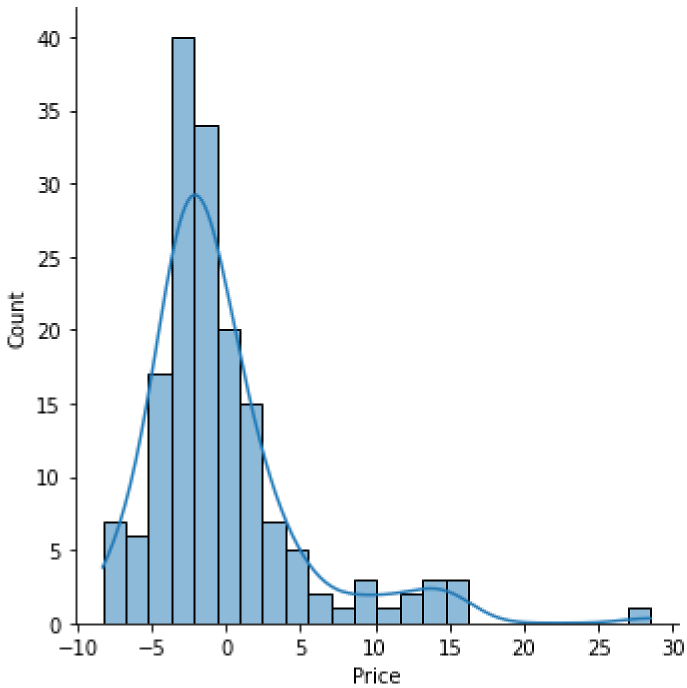
```
173    -2.448027
274     1.285519
491    -4.498452
72     -1.947155
452    -3.030297
          ...
110     0.428514
321    -1.920378
265    -1.645018
29     -2.106526
262    13.490268
Name: Price, Length: 167, dtype: float64
```

In [103]:

```
sns.displot(elastic_residuals,kde=True)
```

Out[103]:

```
<seaborn.axisgrid.FacetGrid at 0x13c437e80>
```

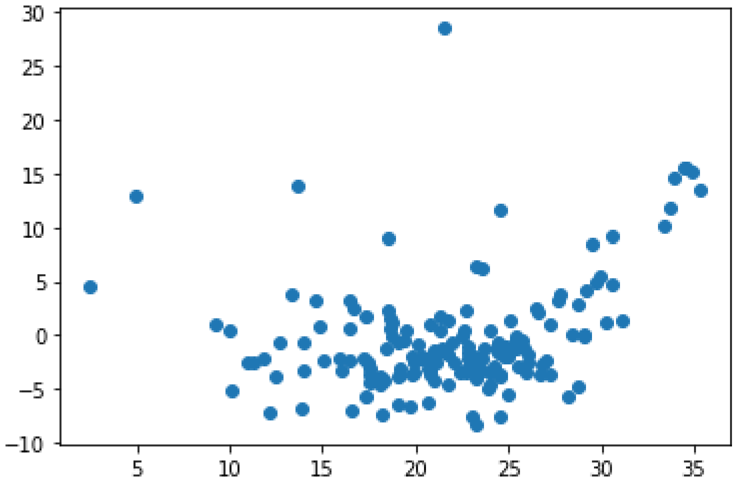


In [104]:

```
plt.scatter(x=elastic_pred,y=elastic_residuals)
```

Out[104]:

```
<matplotlib.collections.PathCollection at 0x13c665730>
```



In [105]:

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

```
27.140175406489988
3.627745135070299
5.209623345932985
```

In [106]:

```
## R Squared error and adjusted R Square

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

Out[106]:

```
0.641375391902405
```

In [107]:

```
###Adjuested R square

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

Out[107]:

```
0.610904019972544
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: