Regression on Boaston dataset

- SHWETA KANHERE BANAIT
  - GitHub :https://github.com/shwetawin/PythonProject (https://github.com/shwetawin/PythonProject)
  - Linkdin: https://www.linkedin.com/in/shweta-kanhere-07810213/ (https://www.linkedin.com/in/shweta-kanhere-07810213/)
  - Email:shweta.kanhere@gmail.com (mailto:shweta.kanhere@gmail.com)

Types of Regression

1. Linear Regression
2. Ridge Regression
3. Lasso Regression
4. Elastic-Net Regression

In [1]:

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

In [2]:

```
boston= load_boston()
```

```
C:\Users\Shweta Kanhere\anaconda3\lib\site-packages\sklearn\utils\deprecatio
n.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 th
is
    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)
```

In [3]:

```
# to get keys of the data set
boston.keys()
```

Out[3]:

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

In [4]:

```python
# data set description
print(boston.DESCR)
```

```
.. _boston_dataset:

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

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,0
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 black p
eople 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/ (http
s://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 [5]:

```
# to get input feature data
print(boston.data)
```

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

In [6]:

```
# to get output feature means target (here output feature is  cost of house which is to
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 [7]:

```
# to get feature name
print(boston.feature_names)
```

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

In [8]:

```
print(boston.filename)
```

```
boston_house_prices.csv
```

In [9]:

```
print(boston.data_module)
```

```
sklearn.datasets.data
```

In [10]:

```
###Preapre the dataframe

```

In [11]:

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

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
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 [12]:

```
# to get the information about dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 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
dtypes: float64(13)
memory usage: 51.5 KB
```

In [13]:

```python
dataset.shape
```

Out[13]:

```
(506, 13)
```

In [14]:

```python
# Add target variable Price
dataset["Price"]=boston.target
```

In [15]:

```python
dataset.head()
```

Out[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
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 [16]:

```
dataset.info()
```

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

In [17]:

```
dataset.shape
```

Out[17]:

```
(506, 14)
```

In [18]:

```
dataset.describe().T
```

Out[18]:

	count	mean	std	min	25%	50%	75%	ma
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.97
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.00
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.74
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.00
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.871
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.78
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.00
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.12
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.00
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.00
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.00
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.90
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.97
Price	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.00

- see (25 -50- 75)%
- Chas has no outliers
- INDUS has few outliers
- Price has minute outlires

In [19]:

```
# check missing values
dataset.isnull().sum()
```

Out[19]:

```
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 [20]:

```
### no missing values
```

- Information
- In Linear regression the most important thing is the relation ship between dependent and Independent features

In [21]:

```python
# EDA
dataset.corr()
```

Out[21]:

	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

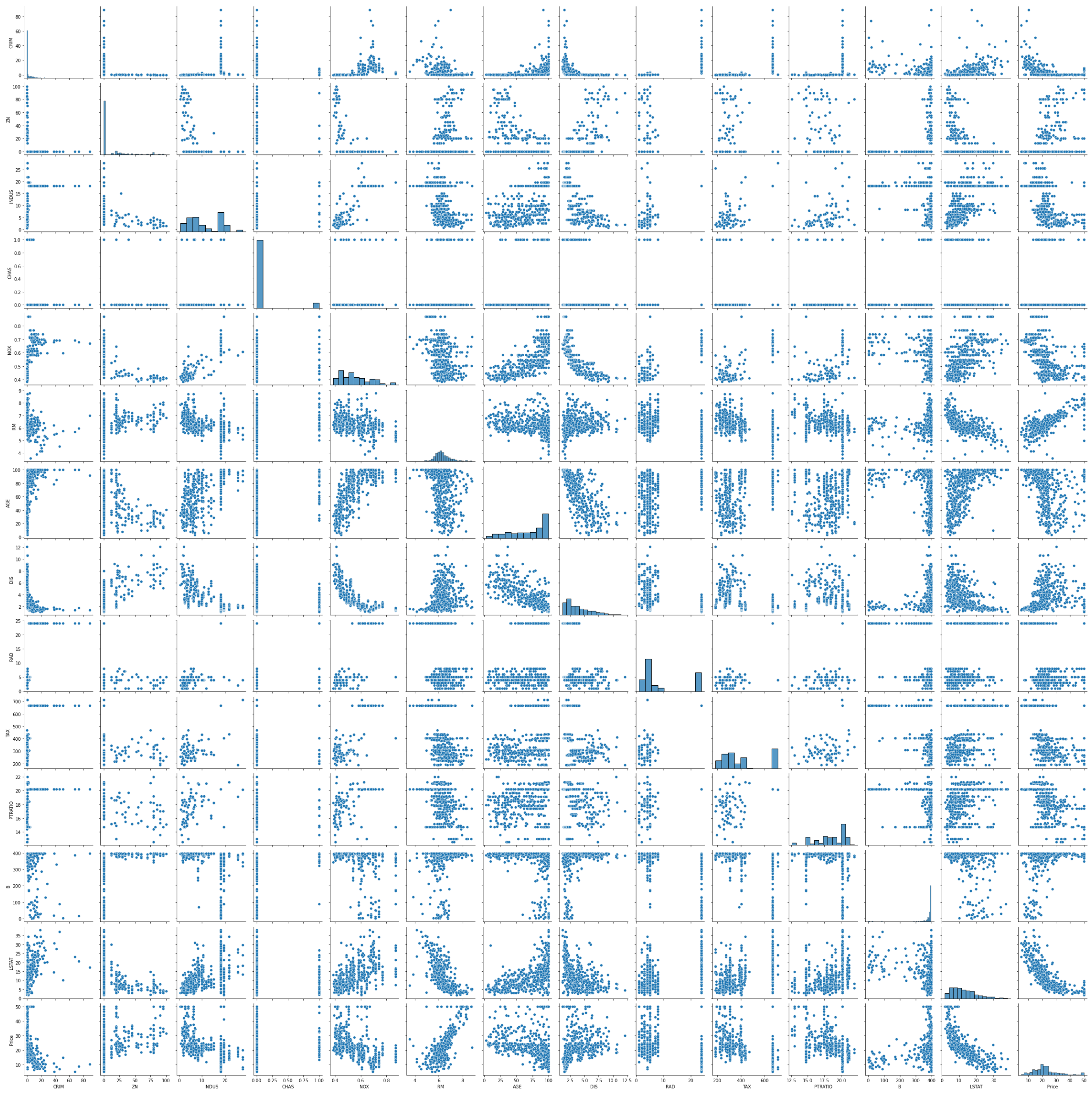
- Relationship betn CRIM and ZN is -0.2004
- CRIM increase Price decreases
- Tax and RAD has good coorelation
- model is good only if relation between dependent and indepent variable is good

In [22]:

```
sns.pairplot(dataset)
```

Out[22]:

```
<seaborn.axisgrid.PairGrid at 0x2069339b1c0>
```

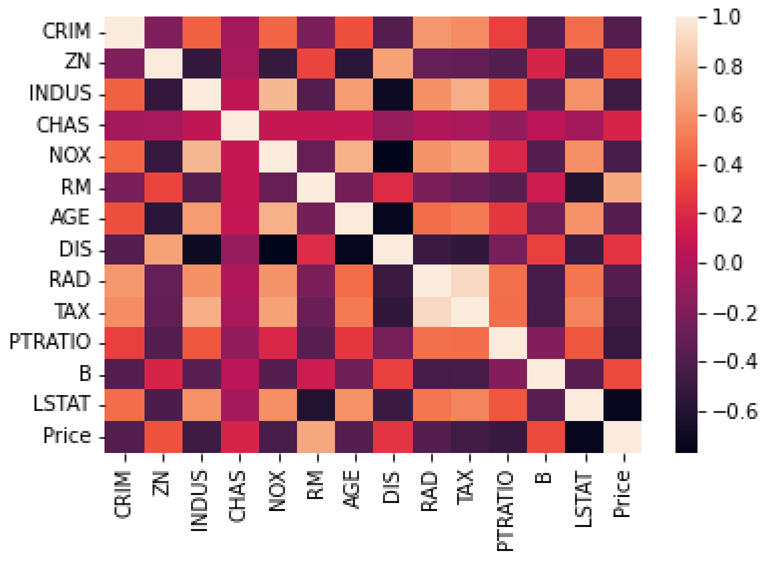


In [23]:

```
## can see this above in better way
sns.heatmap(dataset.corr())
```

Out[23]:

```
<AxesSubplot:>
```

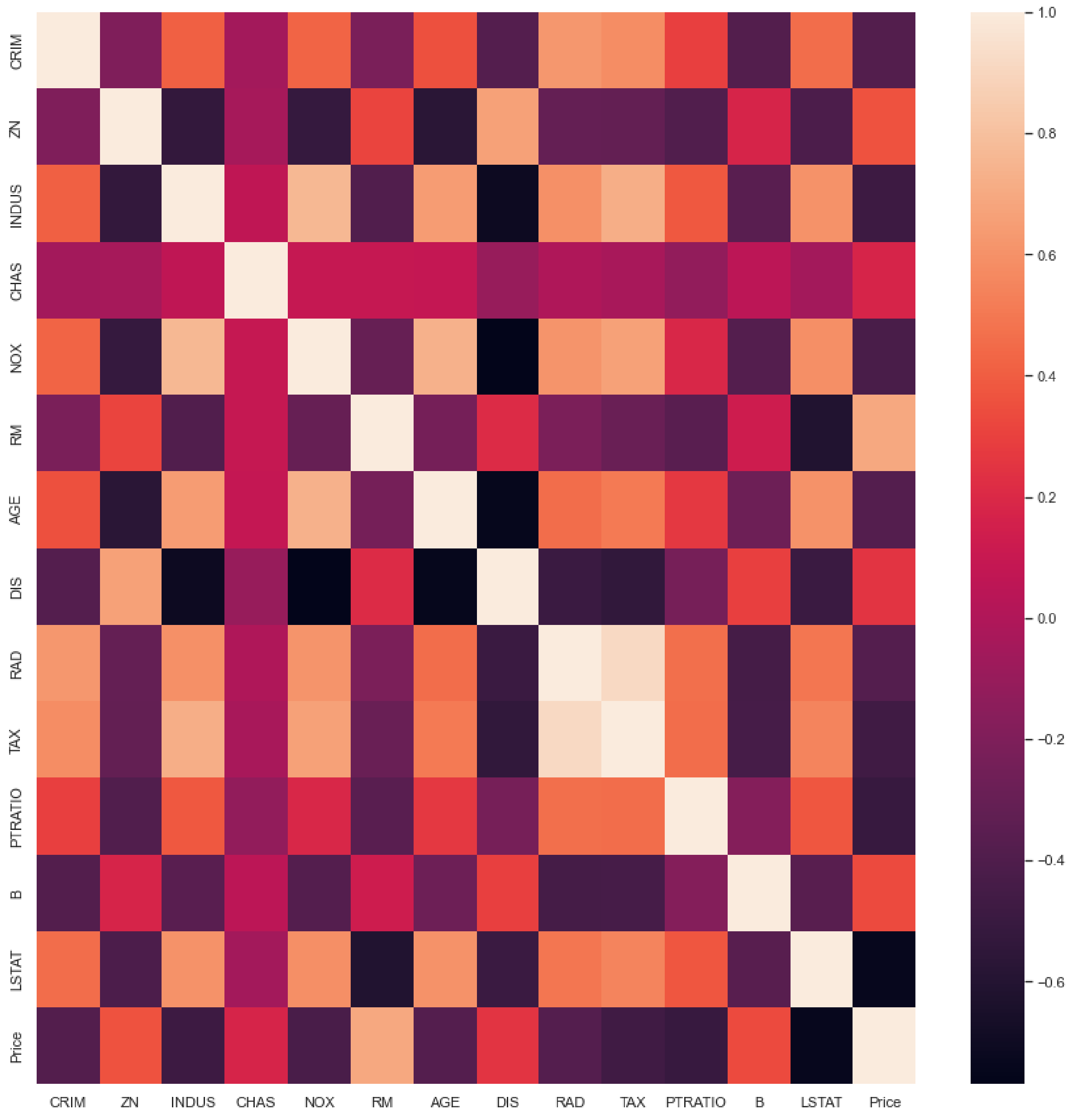


In [24]:

```
## size of graph will increase by figure size
sns.set(rc={'figure.figsize':(15,15)})
sns.heatmap(dataset.corr())
```

Out[24]:

```
<AxesSubplot:>
```

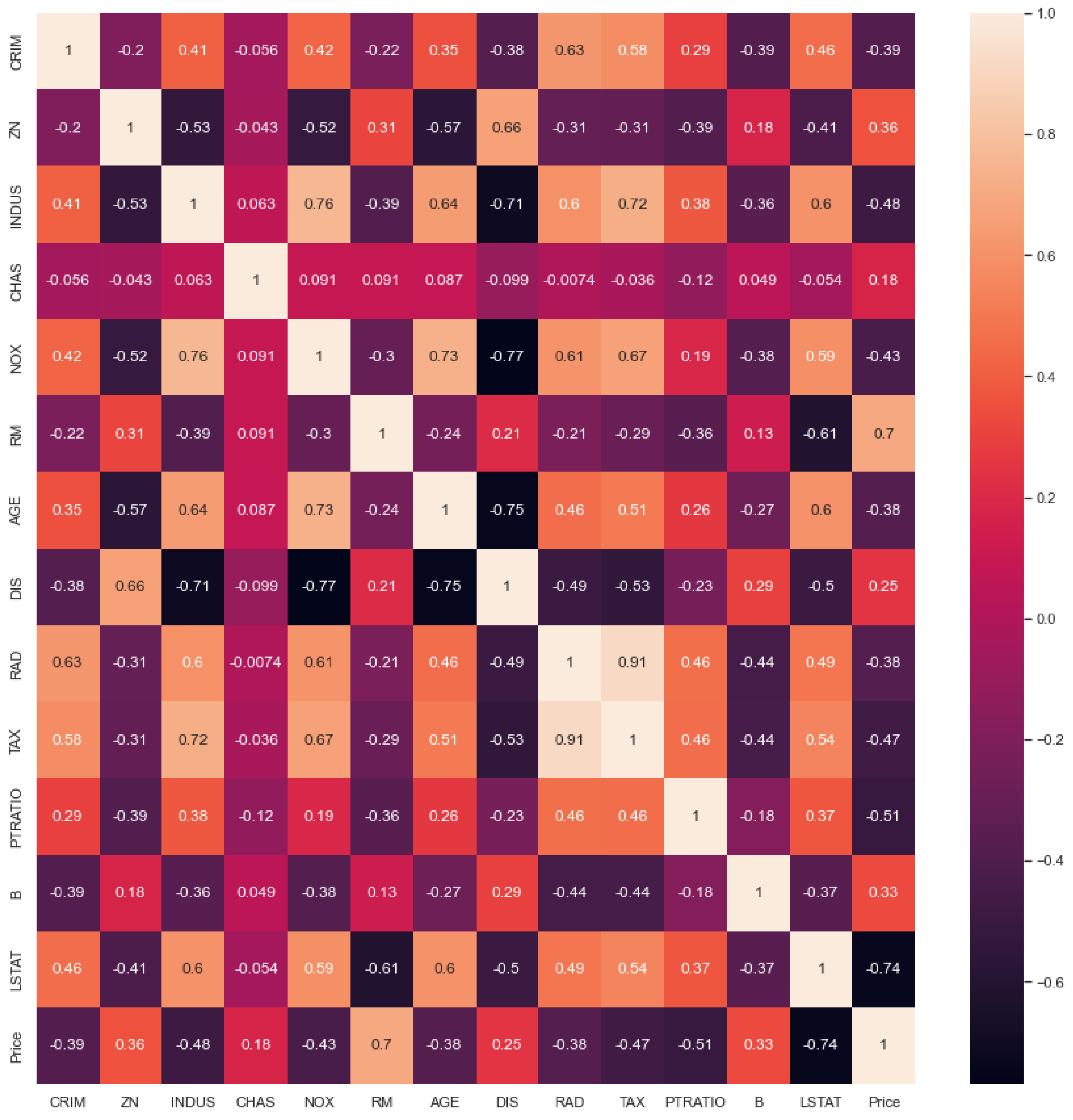


In [25]:

```
## chek the % by annot= True
sns.set(rc={'figure.figsize':(15,15)})
sns.heatmap(dataset.corr(),annot= True)
```

Out[25]:

```
<AxesSubplot:>
```

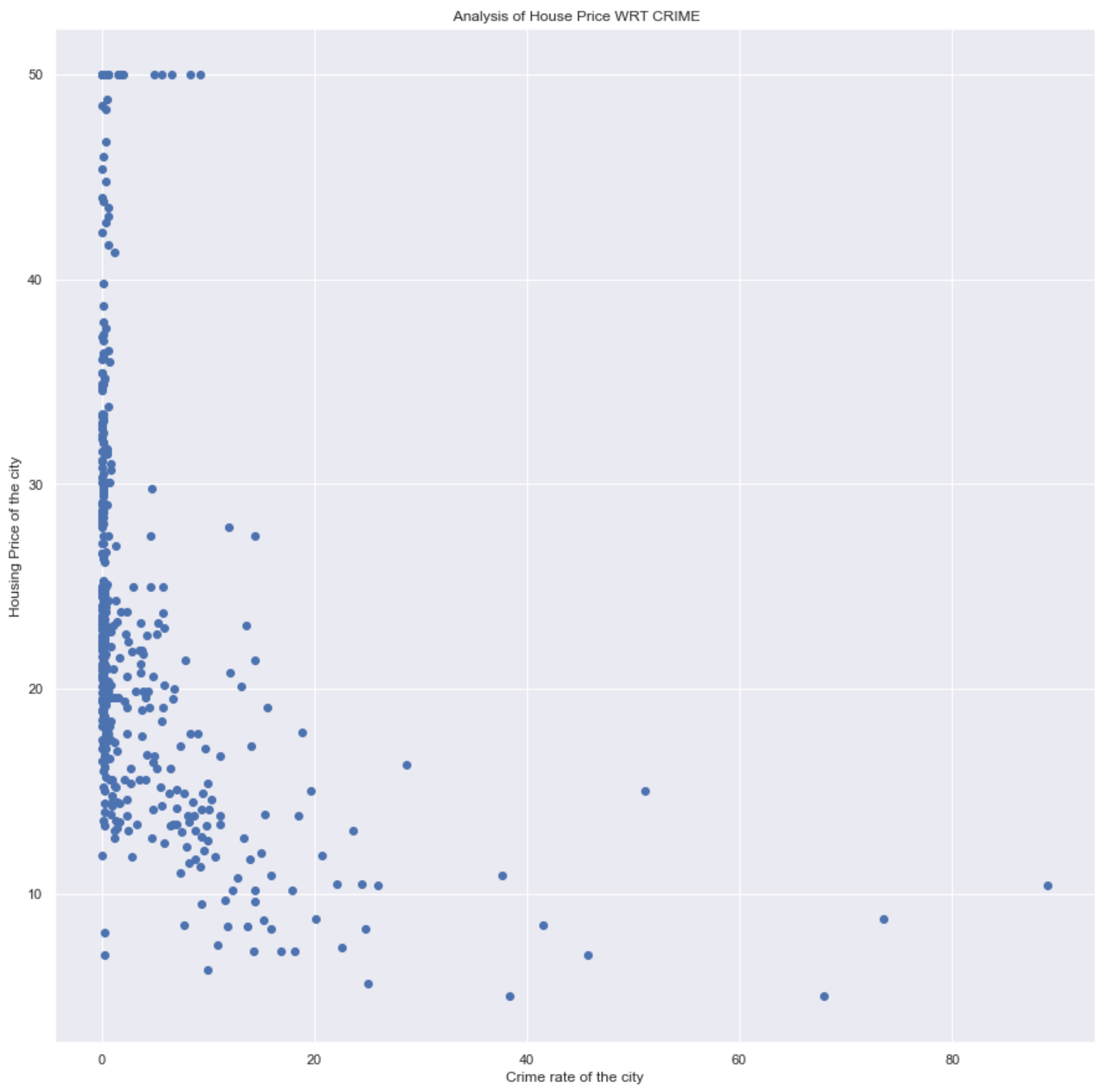


In [26]:

```
# to see some relationship betn crim and Price
# use scatter plot
plt.scatter(dataset["CRIM"],dataset["Price"])
plt.title(" Analysis of House Price WRT CRIME")
plt.xlabel("Crime rate of the city")
plt.ylabel("Housing Price of the city")

```

Out[26]:

```
Text(0, 0.5, 'Housing Price of the city')
```



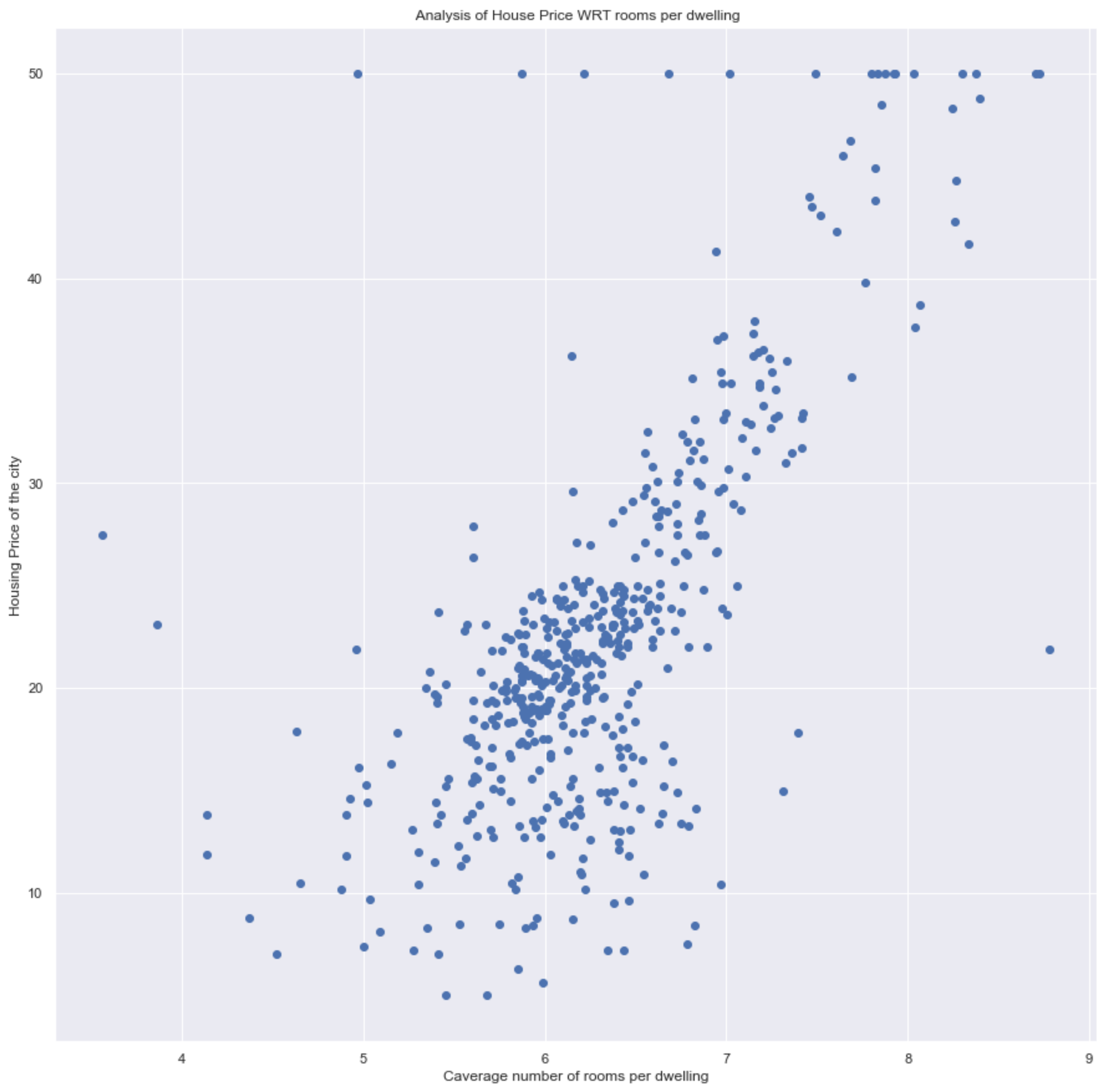
Observation : crime is high then price is low

In [27]:

```
plt.scatter(dataset["RM"],dataset["Price"])
plt.title(" Analysis of House Price WRT rooms per dwelling")
plt.xlabel("Caverage number of rooms per dwelling")
plt.ylabel("Housing Price of the city")
```

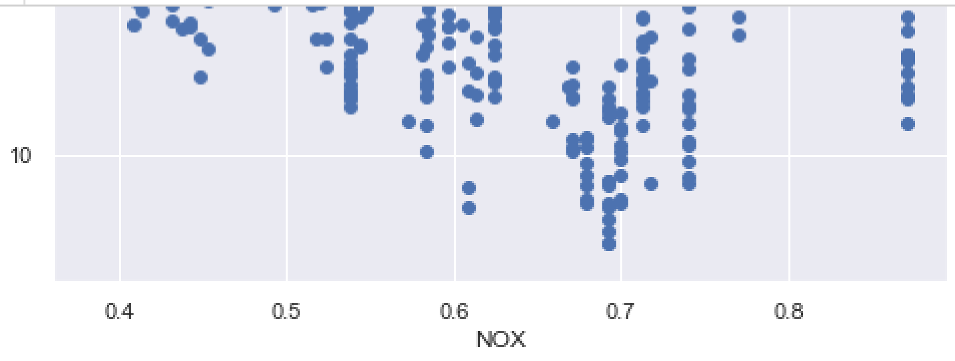
Out[27]:

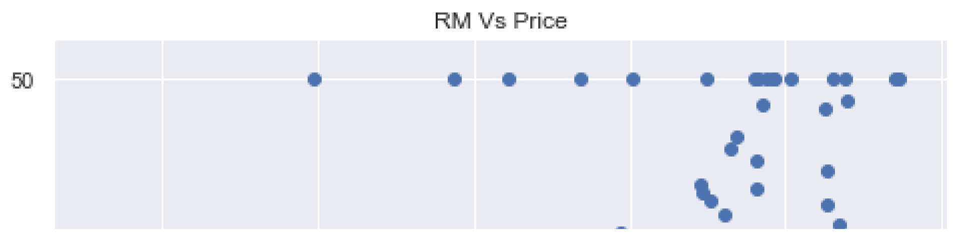
```
Text(0, 0.5, 'Housing Price of the city')
```



In [94]:

```python
### Visualising all the features with repeect to  price
for feature in [feature for feature in dataset.columns if feature not in ['Price']]:
 sns.set(rc={'figure.figsize':(8,8)})
 plt.scatter(x=dataset[feature], y=dataset['Price'])
 plt.xlabel(feature)
 plt.ylabel("Price")
 plt.title("{} Vs Price".format(feature))
 plt.show();
```





- good correlaton between them
- RM is good feature to find output
- model will create good feet line

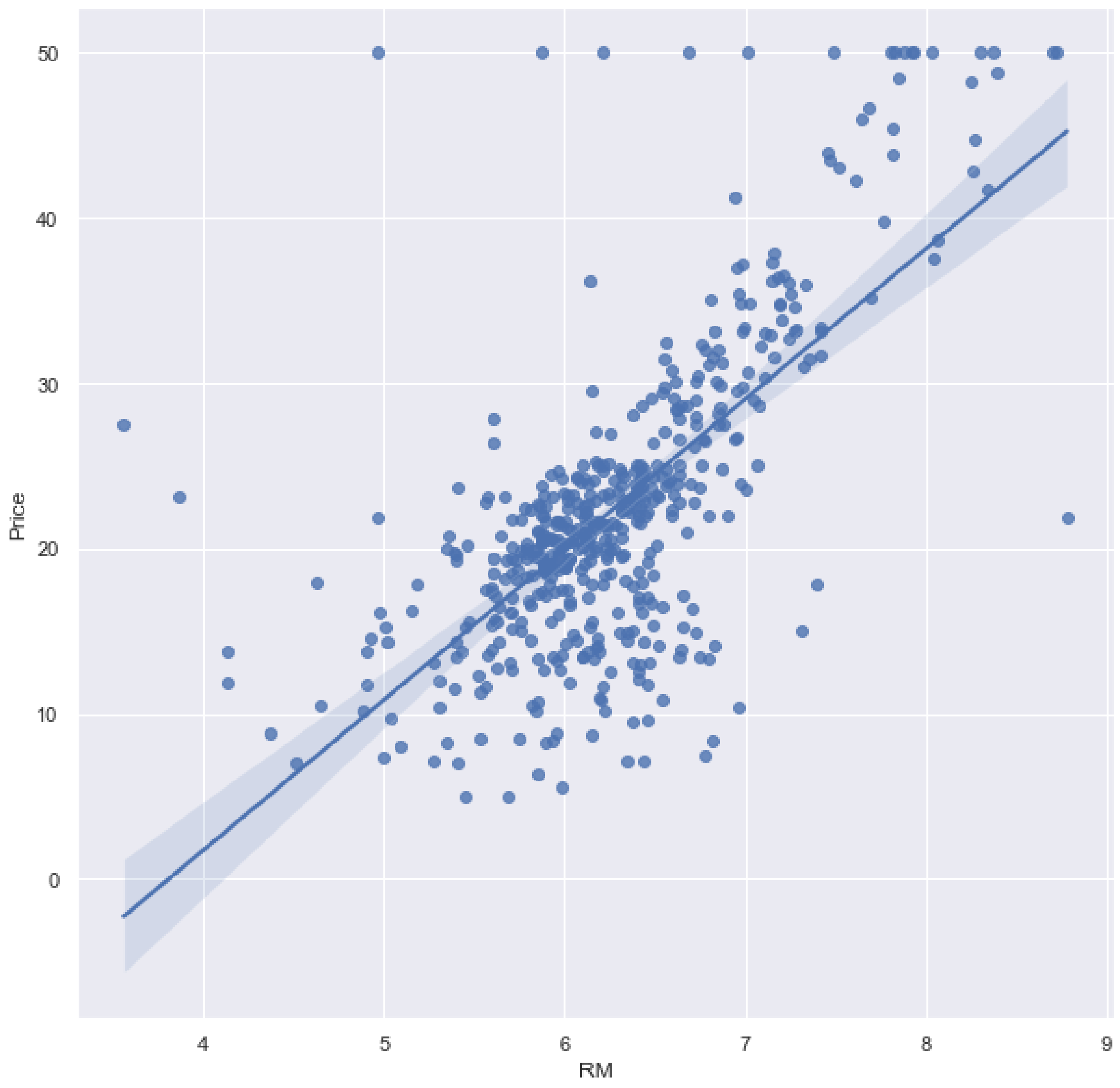
- The REGPLOT------- Plot data in a linear regression model fit. There are a number of mutually exclusive options for estimating the regression model

In [28]:

```
sns.set(rc={'figure.figsize':(10,10)}) # set the figure size
sns.regplot(data=dataset,x="RM",y="Price")
```

Out[28]:

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



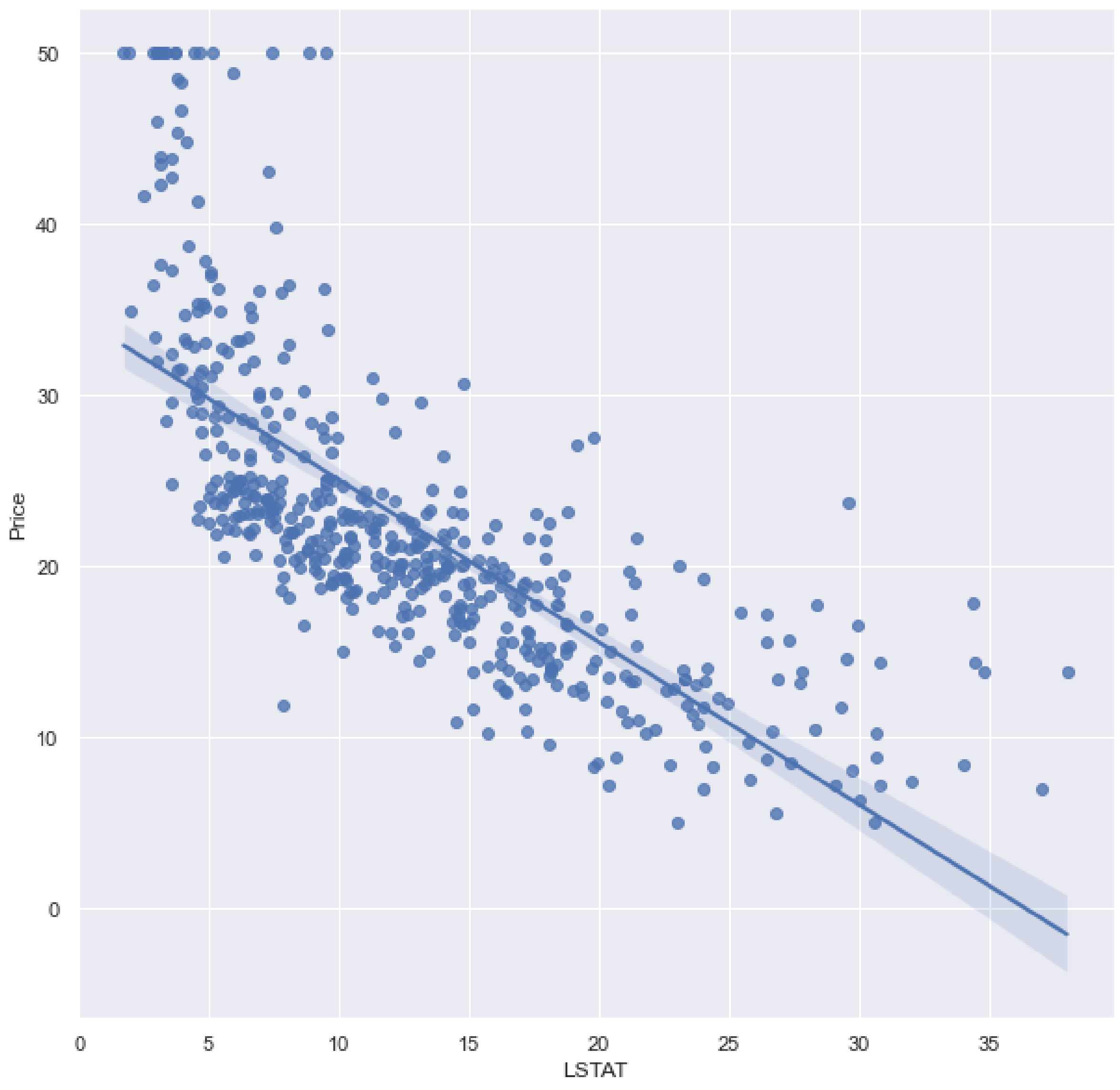
- shaded region shows region lasso
- with change in Lamda we get best fit line
- concentration of points are more then movement is less and concentration of points are less then movement is high

In [29]:

```
#LSTAT -    % lower status of the population
sns.regplot(data=dataset,x="LSTAT",y="Price")
```

Out[29]:

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

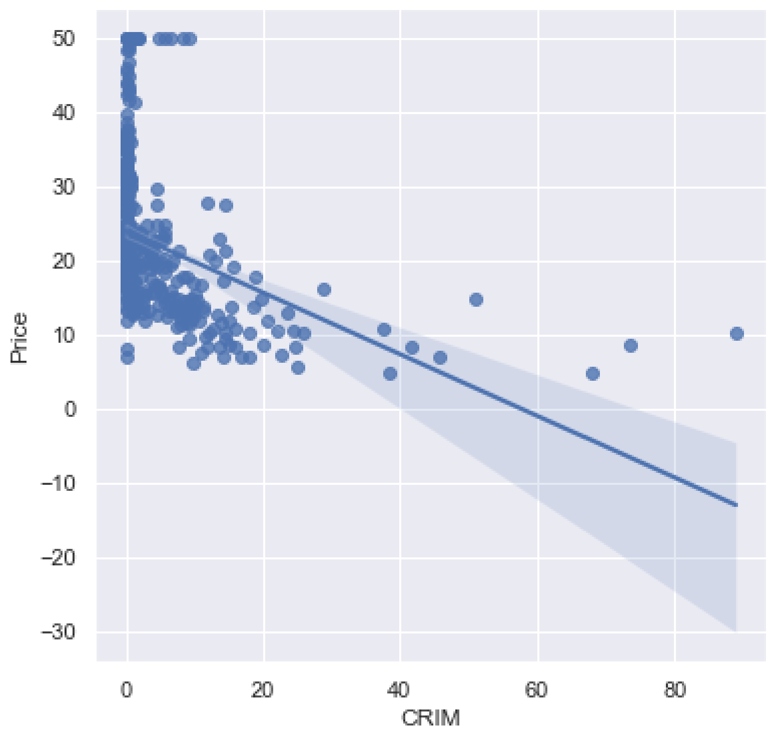


In [30]:

```
sns.set(rc={'figure.figsize':(6,6)}) # set the figure size
sns.regplot(data=dataset,x="CRIM",y="Price")  #----- negative correlation
```

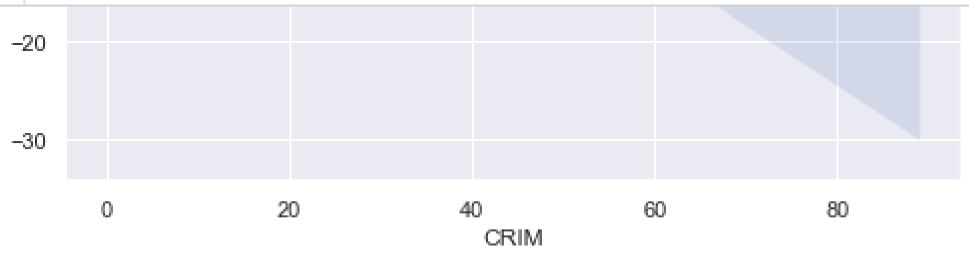
Out[30]:

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



In [95]:

```python
#### Regression plot for all other features wrt Price
for feature in [feature for feature in dataset.columns if feature not in ['Price']]:
 sns.set(rc={'figure.figsize':(8,8)})
 sns.regplot(x=dataset[feature], y=dataset['Price'])
 plt.xlabel(feature)
 plt.ylabel("Price")
 plt.title("{} Vs Price".format(feature))
 plt.show();
```



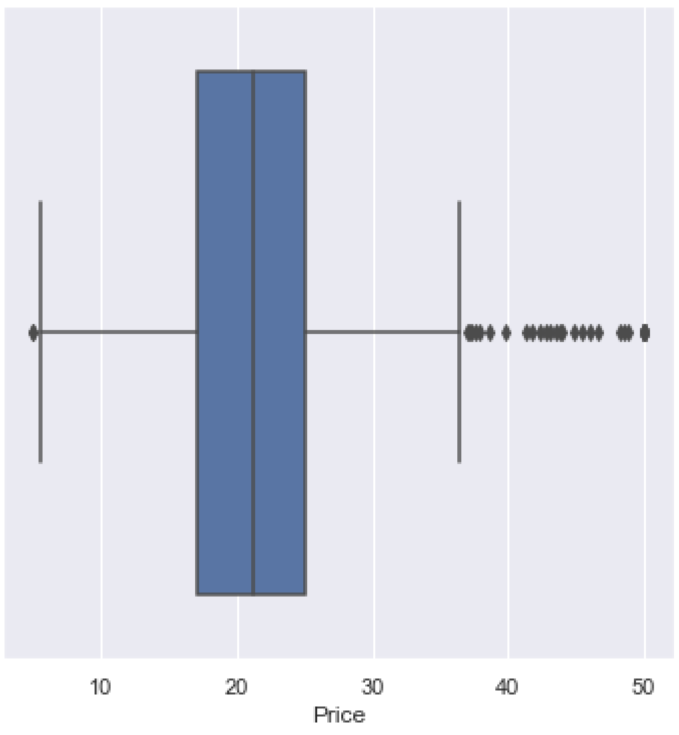
In [31]:

```
sns.boxplot(dataset['Price'])
```

```
C:\Users\Shweta Kanhere\anaconda3\lib\site-packages\seaborn\_decorators.py:3
6: 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 oth
er arguments without an explicit keyword will result in an error or misinter
pretation.
  warnings.warn(
```

Out[31]:

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



Less outliers in Price

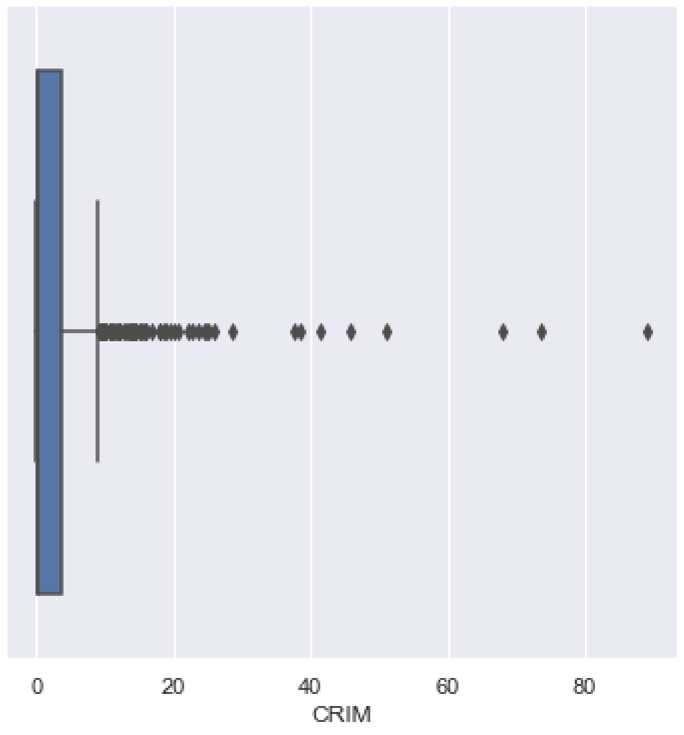
In [32]:

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

```
C:\Users\Shweta Kanhere\anaconda3\lib\site-packages\seaborn\_decorators.py:3
6: 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 oth
er arguments without an explicit keyword will result in an error or misinter
pretation.
  warnings.warn(
```

Out[32]:

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



In [33]:

```
### More outliers with crime ant this impact Price
```

In [96]:

```python
# Box plot for all the features
for feature in dataset.columns:
 sns.set(rc={'figure.figsize':(8,8)})
 sns.boxplot(dataset[feature])
 plt.xlabel(feature)
 plt.title("{}".format(feature))
 plt.show();
```

```
version 0.12, the only valid positional argument will be `data`, and passi
ng other arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

Training the Model

In [34]:

```python
dataset.head()
```

Out[34]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
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 [35]:

```
### Independent and Dependint features
## Independent features= 13
X=dataset.iloc[:,:-1]      # lock all the columns from zero index to last except last t
X.head()
```

Out[35]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
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 [36]:

```
y= dataset.iloc[:,-1]   # only last row
y.head()
```

Out[36]:

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

Note

- Dependent feature is series
- Independent feature is dataframe

  Training and testing
- Scaling will do after train and test of data set
- when we train the model then the model dont have any information about test data
- so train test split of the data is important
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

In [37]:

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

In [38]:

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

- X_train output is y_train

- X_test output is y_test
- X is independent feature
- y is dependent feature
- test_size decides how much % of data goes to train data 33% of total data
- random_state=42 my data is same with your if you select 42
- Size of X_train = y_train same for test

In [39]:

```
y_train.shape
```

Out[39]:

```
(339,)
```

In [40]:

```
X_train.shape
```

Out[40]:

```
(339, 13)
```

In [41]:

```
X_test.shape
```

Out[41]:

```
(167, 13)
```

In [42]:

```
y_test.shape
```

Out[42]:

```
(167,)
```

In [43]:

```
y_train
```

Out[43]:

```
478    14.6
26     16.6
7      27.1
492    20.1
108    19.8
       ...
106    19.5
270    21.1
348    24.5
435    13.4
102    18.6
Name: Price, Length: 339, dtype: float64
```

In [44]:

```
X_train
```

Out[44]:

	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

Feature Scaling or Standardization

- it use to reach globel minima (Gradent Decent)

In [45]:

```
from sklearn.preprocessing import StandardScaler
```

In [46]:

```
scaler= StandardScaler()
scaler     # StandardScaler() object is created
```

Out[46]:

```
StandardScaler()
```

In [47]:

```
#### apply scaler on X train data
scaler.fit_transform(X_train)
```

Out[47]:

```
array([[ 0.89624872, -0.51060139,  0.98278223, ...,  0.86442095,
         0.24040357,  0.77155612],
       [-0.34895881, -0.51060139, -0.44867555, ...,  1.22118698,
         0.20852839,  0.32248963],
       [-0.41764058,  0.03413008, -0.48748013, ..., -1.36536677,
         0.43481957,  0.92775316],
       ...,
       [-0.43451148,  2.97567999, -1.32968321, ..., -0.56264319,
         0.36745216, -0.90756208],
       [ 1.01703049, -0.51060139,  0.98278223, ...,  0.86442095,
        -2.80977992,  1.50233514],
       [-0.40667333, -0.51060139, -0.38831288, ...,  1.17659123,
        -3.25117205, -0.26046005]])
```

In [48]:

```
## after transfor store the data into variable called X_train or any new variable
X_train=scaler.fit_transform(X_train)
X_train
```

Out[48]:

```
array([[ 0.89624872, -0.51060139,  0.98278223, ...,  0.86442095,
         0.24040357,  0.77155612],
       [-0.34895881, -0.51060139, -0.44867555, ...,  1.22118698,
         0.20852839,  0.32248963],
       [-0.41764058,  0.03413008, -0.48748013, ..., -1.36536677,
         0.43481957,  0.92775316],
       ...,
       [-0.43451148,  2.97567999, -1.32968321, ..., -0.56264319,
         0.36745216, -0.90756208],
       [ 1.01703049, -0.51060139,  0.98278223, ...,  0.86442095,
        -2.80977992,  1.50233514],
       [-0.40667333, -0.51060139, -0.38831288, ...,  1.17659123,
        -3.25117205, -0.26046005]])
```

In [49]:

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

Out[49]:

```
array([[-0.42451319, -0.51060139, -1.03649306, ..., -0.74102621,
         0.41899501, -0.48220406],
       [-0.42911576,  1.2325393 , -0.6973123 , ..., -0.29506866,
         0.43481957, -1.25063772],
       [-0.42269508, -0.51060139,  2.36824941, ...,  0.8198252 ,
         0.35807046,  0.77713459],
       ...,
       [-0.33727525,  0.36096896, -1.04799071, ..., -2.34647337,
         0.38395492, -0.28556314],
       [-0.30591027, -0.51060139, -0.44867555, ...,  1.22118698,
         0.2463943 , -0.07218683],
       [-0.36872487,  0.36096896, -1.04799071, ..., -2.34647337,
         0.32133488, -0.91871901]])
```

Important

- for test data only transfor function is used not fit_transform to avoide data leakage.
- model should not need to know which techniques is used for training.

Mode Training

1.Linear Regression Model

Multiple regression used for?

- making a prediction or forecasting

- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linearregression#sklearn.linear_model.LinearRegression (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linearregression#sklearn.linear_model.LinearRegression)
- We can skip the transform and we can directly apply model training.
- in model training there is (normalizebool, default=False) if its true then transformation is done

In [50]:

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

In [51]:

```
reg = LinearRegression()
reg
```

Out[51]:

```
LinearRegression()
```

- fit means traning the data or to get the parameters.Data does not change
- fit_transform means to set the data and change the data

In [52]:

```
reg.fit(X_train,y_train)   # y_train - dependent feature and  X_train- independent feat
```

Out[52]:

```
LinearRegression()
```

In [53]:

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

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

In [54]:

```
### there are 13 coefficients as features are 13 like crime, ZN,INDUS
```

In [55]:

```
print(reg.intercept_)
```

```
22.970796460176988
```

In [56]:

```
# if all the features are zero then the price of the house is 22.97
# if one unit of increase in price then crime is decrease
```

Prediction for test Data

In [57]:

```
reg_predict = reg.predict(X_test)
```

In [58]:

```
reg_predict
```

Out[58]:

```
array([28.53469469, 36.6187006 , 15.63751079, 25.5014496 , 18.7096734 ,
       23.16471591, 17.31011035, 14.07736367, 23.01064388, 20.54223482,
       24.91632351, 18.41098052, -6.52079687, 21.83372604, 19.14903064,
       26.0587322 , 20.30232625,  5.74943567, 40.33137811, 17.45791446,
       27.47486665, 30.2170757 , 10.80555625, 23.87721728, 17.99492211,
       16.02608791, 23.268288  , 14.36825207, 22.38116971, 19.3092068 ,
       22.17284576, 25.05925441, 25.13780726, 18.46730198, 16.60405712,
       17.46564046, 30.71367733, 20.05106788, 23.9897768 , 24.94322408,
       13.97945355, 31.64706967, 42.48057206, 17.70042814, 26.92507869,
       17.15897719, 13.68918087, 26.14924245, 20.2782306 , 29.99003492,
       21.21260347, 34.03649185, 15.41837553, 25.95781061, 39.13897274,
       22.96118424, 18.80310558, 33.07865362, 24.74384155, 12.83640958,
       22.41963398, 30.64804979, 31.59567111, 16.34088197, 20.9504304 ,
       16.70145875, 20.23215646, 26.1437865 , 31.12160889, 11.89762768,
       20.45432404, 27.48356359, 10.89034224, 16.77707214, 24.02593714,
        5.44691807, 21.35152331, 41.27267175, 18.13447647,  9.8012101 ,
       21.24024342, 13.02644969, 21.80198374,  9.48201752, 22.99183857,
       31.90465631, 18.95594718, 25.48515032, 29.49687019, 20.07282539,
       25.5616062 ,  5.59584382, 20.18410904, 15.08773299, 14.34562117,
       20.85155407, 24.80149389, -0.19785401, 13.57649004, 15.64401679,
       22.03765773, 24.70314482, 10.86409112, 19.60231067, 23.73429161,
       12.08082177, 18.40997903, 25.4366158 , 20.76506636, 24.68588237,
        7.4995836 , 18.93015665, 21.70801764, 27.14350579, 31.93765208,
       15.19483586, 34.01357428, 12.85763091, 21.06646184, 28.58470042,
       15.77437534, 24.77512495,  3.64655689, 23.91169589, 25.82292925,
       23.03339677, 25.35158335, 33.05655447, 20.65930467, 38.18917361,
       14.04714297, 25.26034469, 17.6138723 , 20.60883766,  9.8525544 ,
       21.06756951, 22.20145587, 32.2920276 , 31.57638342, 15.29265938,
       16.7100235 , 29.10550932, 25.17762329, 16.88159225,  6.32621877,
       26.70210263, 23.3525851 , 17.24168182, 13.22815696, 39.49907507,
       16.53528575, 18.14635902, 25.06620426, 23.70640231, 22.20167772,
       21.22272327, 16.89825921, 23.15518273, 28.69699805,  6.65526482,
       23.98399958, 17.21004545, 21.0574427 , 25.01734597, 27.65461859,
       20.70205823, 40.38214871])
```

Assumptions for Linear Regression

- assumptions are used to check model is working good or not

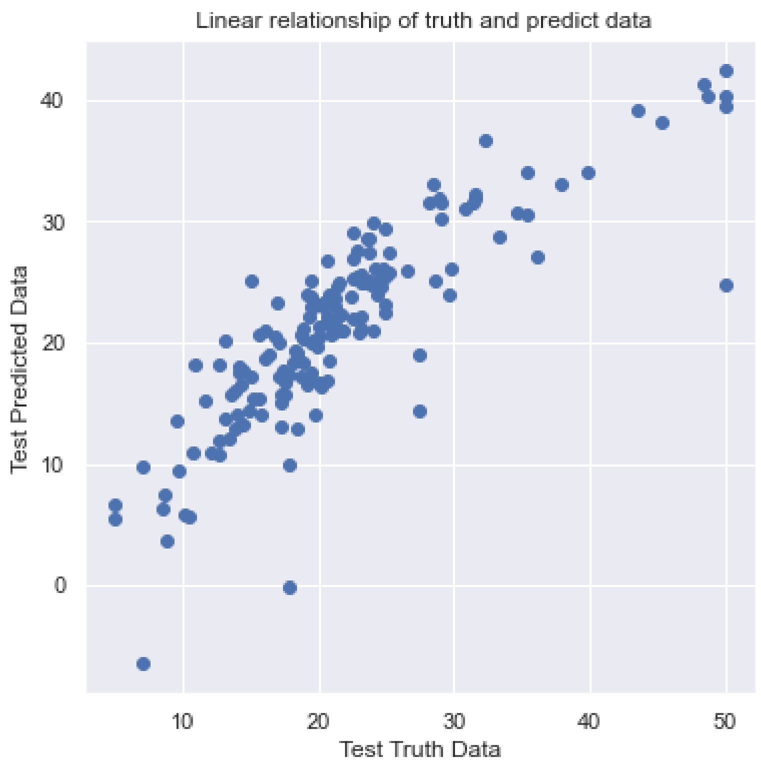
```
### 1. First Assumption
```

In [59]:

```

## check scatter plot for y_test and predict data
plt.scatter(y_test,reg_predict)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
plt.title(" Linear relationship of truth and predict data")
```

Out[59]:

```
Text(0.5, 1.0, ' Linear relationship of truth and predict data')
```



- It shows linear relationship
- so model is good

2.Second Assumption

In [60]:

```
### residuals means error
residuals= y_test- reg_predict
residuals
```

Out[60]:

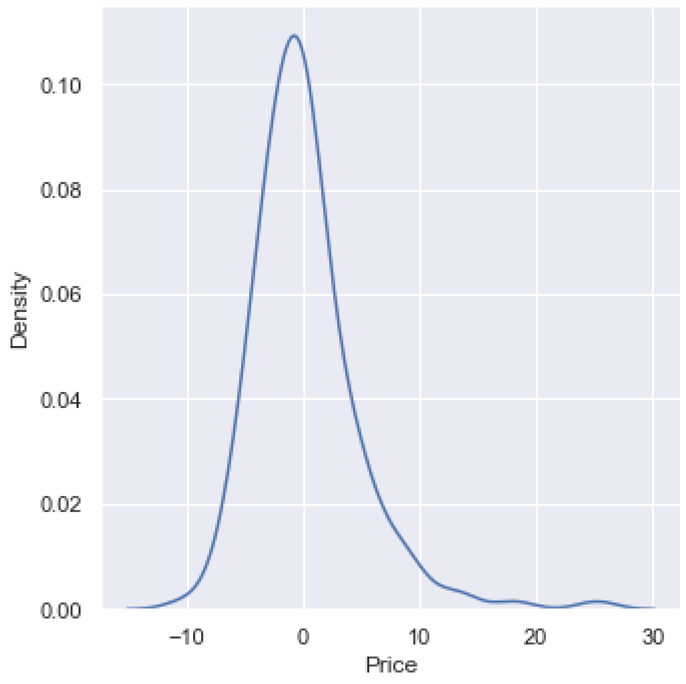
```
173   -4.934695
274   -4.218701
491   -2.037511
72    -2.701450
452   -2.609673
         ...
110    0.642557
321   -1.917346
265   -4.854619
29     0.297942
262    8.417851
Name: Price, Length: 167, dtype: float64
```

In [61]:

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

Out[61]:

```
<seaborn.axisgrid.FacetGrid at 0x206a059beb0>
```



Observation

- the error plot is gaussion distribution (normal distribution) with negative skewed (right skewed) due to little outliers

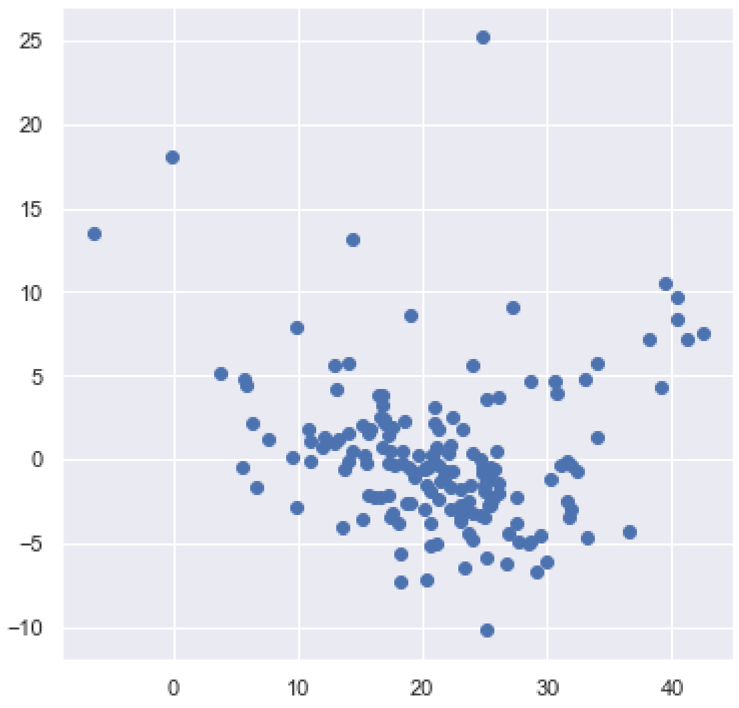
3.Third Assumption

In [62]:

```
### Scatter plot with preciction and residual
## Uniform distribution
plt.scatter(reg_predict,residuals)
```

Out[62]:

```
<matplotlib.collections.PathCollection at 0x206a35eeb20>
```



In [99]:

```
## This above distribution is uniform distribution
# doesn't have any define shape
```

4.Fourth Assumption

In [64]:

```
## performance Metric
```

In [65]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

In [66]:

```
print("mean_squared_error : ", mean_squared_error(y_test,reg_predict))
print( "mean_absolute_error:", mean_absolute_error(y_test,reg_predict))
print("Root mean square error : ",np.sqrt(mean_squared_error(y_test,reg_predict)))
```

```
mean_squared_error :  20.72402343733975
mean_absolute_error: 3.148255754816832
Root mean square error :  4.552364598463061
```

R square and Adjusted R square

R square

In [67]:

```
from sklearn.metrics import r2_score
score=r2_score(y_test,reg_predict)
print("R square Value is: ", score)
```

```
R square Value is:  0.7261570836552477
```

Adjusted R2

- Formula
-
  - Adjusted R2 = 1 – [(1-R2)*(n-1)/(n-k-1)]

In [68]:

```
## Adjusted R square  using formula
#display adjusted R-squared
Adjusted_R = 1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("Adjusted R square is: ", Adjusted_R)
```

```
Adjusted R square is:  0.702889384880857
```

```
##### Note :
* Adjusted R square is less than R square
```

2.Ridge Regression

- Ref : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html)
- sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, normalize='deprecated', copy_X=True, max_iter=None, tol=0.001, solver='auto', positive=False, random_state=None)

In [71]:

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

Out[71]:

```
Ridge()
```

In [72]:

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

Out[72]:

```
Ridge()
```

In [129]:

```
## print the coefficients and the intercept
print("***Ridge.coef are-- ",ridge.coef_)
print("***Ridge.intercept :",ridge.intercept_)
```

```
***Ridge.coef are--  [-0.97541551  0.84608896  0.37564928  0.86738391 -1.860
77739  2.81535042
 -0.36108635 -3.00177053  1.95063015 -1.29462251 -2.06972563  1.03867858
 -3.91121554]
***Ridge.intercept : 22.970796460176988
```

In [103]:

```
# prediction for test data
ridge_predict=ridge.predict(X_test)
```

```
### Assumptions for Ridge Regression
```

1.First Assumption

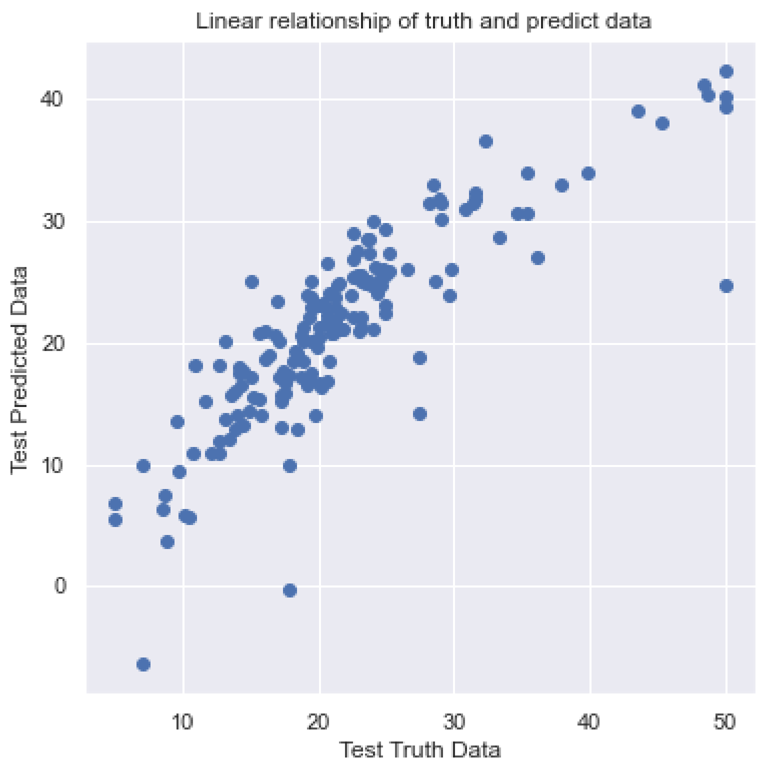
- 1.Test truth data and Test predicted data should follow linear relationship.
- 2.If we get linear relationship then this indicates that the model is good.

In [76]:

```
plt.scatter(y_test,ridge_predict)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
plt.title(" Linear relationship of truth and predict data")
```

Out[76]:

```
Text(0.5, 1.0, ' Linear relationship of truth and predict data')
```



2.Second Assumption

- 1. Residuals should follow normal distribution.
- 2. If residuals follow normal distribution, it indicates that the model is good model.

In [110]:

```python
### residuals means error
residuals1= y_test- ridge_predict
residuals1.head()
```

Out[110]:

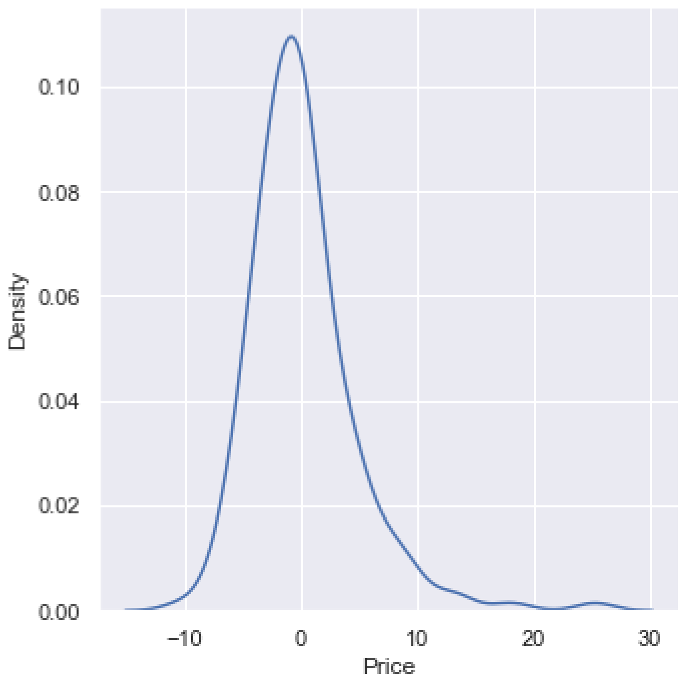
```
173   -4.907420
274   -4.166232
491   -2.143852
72    -2.691076
452   -2.604498
Name: Price, dtype: float64
```

In [108]:

```python
sns.displot(residuals1,kind="kde")
```

Out[108]:

```
<seaborn.axisgrid.FacetGrid at 0x206a55e3370>
```



3.Third Assumption

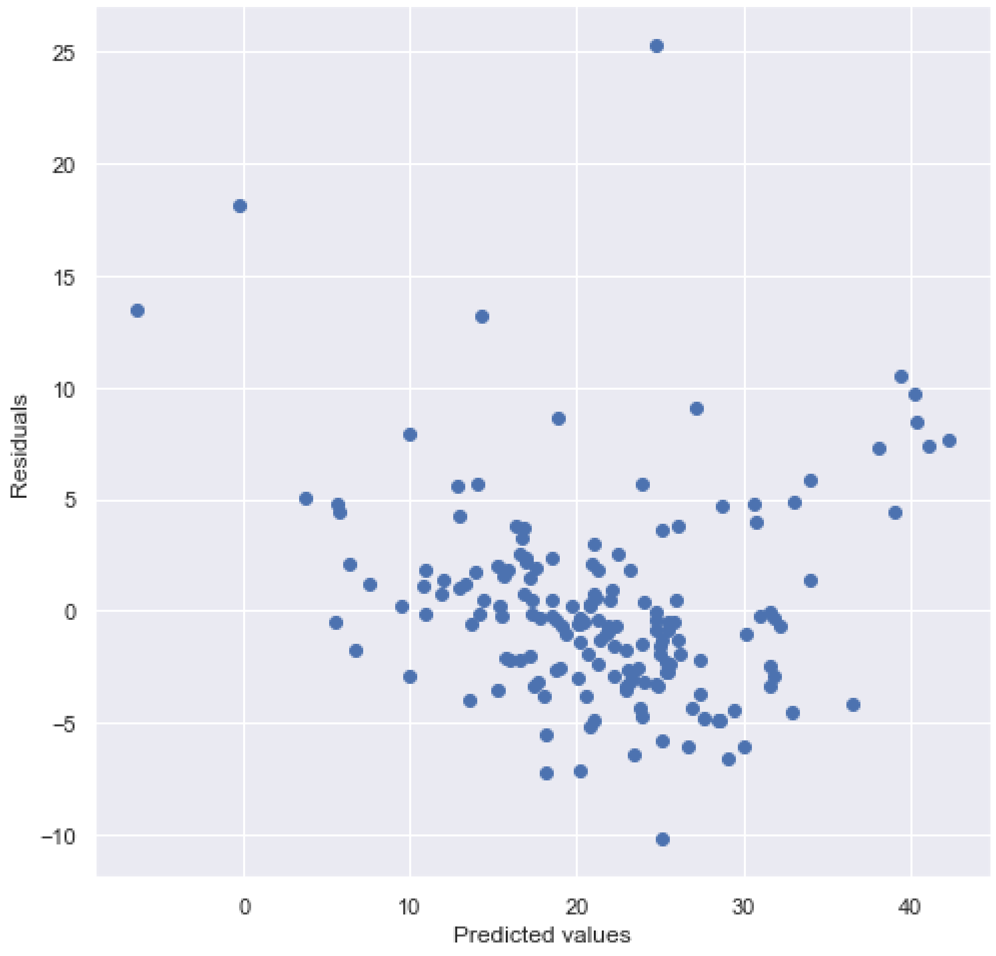
- 1. Residuals vs Predictions should follow a uniform distribution.
- 2. If Residuals vs Predictions follow uniform distribution, it indicates this is good model.

In [107]:

```
## Uniform distribution
### Scatter plot with preciction and residual
plt.scatter(ridge_predict,residuals1)
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
```

Out[107]:

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



4.Fourth Assumption

In [118]:

```
print("mean_squared_error : ", format(round(mean_squared_error(y_test,ridge_predict),4)
print( "mean_absolute_error:", format(round(mean_absolute_error(y_test,ridge_predict),4
print("Root mean square error : ",format(round(np.sqrt(mean_squared_error(y_test,ridge_
```

```
mean_squared_error :  20.7524
mean_absolute_error: 3.146
Root mean square error :  4.5555
```

R square

In [121]:

```
from sklearn.metrics import r2_score
score1=r2_score(y_test,ridge_predict)
print("R square Value for ridge regression is: ", format(round(score1*100,4)))
```

```
R square Value for ridge regression is:  72.5782
```

Adjusted R square

In [126]:

```
#adjusted R-squared
Adjusted_R_ridge = 1 - (1-score1)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("Adjusted R square for ridge regression is: ", format(round(Adjusted_R_ridge*100,
```

```
Adjusted R square for ridge regression is:  70.2482
```

```
## 3.Lasso Regression
* REF : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html?highlight=lasso
* lass sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, normalize='deprecated', precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic'
```

In [160]:

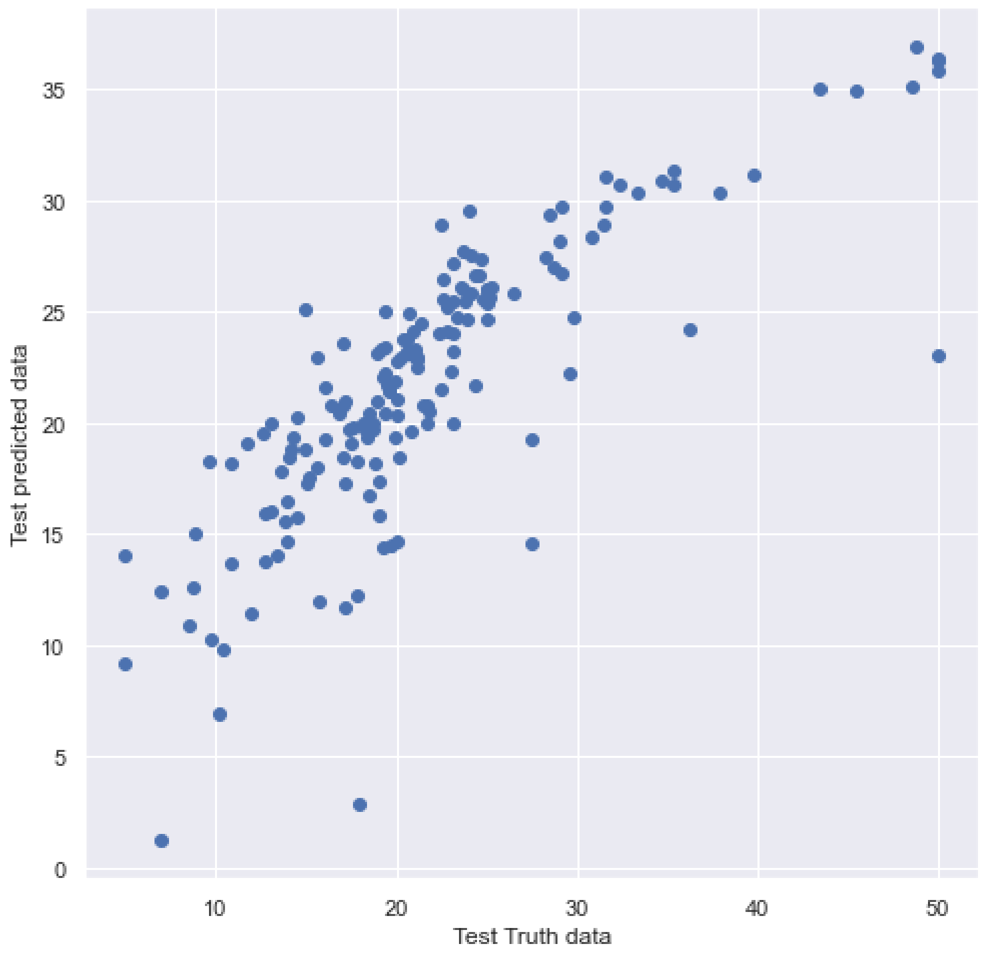
```python
from sklearn.linear_model import Lasso
lasso = Lasso()
lasso
lasso.fit(X_train,y_train)
## print the coefficients and the intercept
print("***Lasso.coef are-- ",lasso.coef_)
print("***Lasso.intercept :",lasso.intercept_)
print("________________________________________________________________________________
# prediction for test data
lasso_predict=lasso.predict(X_test)
print("***lasso_predict",lasso_predict)
print("________________________________________________________________________________
### Assumptions of Lasso regression
### 1.First Assumption ---- Linear relationship
plt.scatter(y_test, lasso_predict)
plt.xlabel('Test Truth data')
plt.ylabel('Test predicted data')

## 2.second Assumption ---- Residual distribution
residuals_lasso=y_test-lasso_predict
print("***residuals_lasso.head()",residuals_lasso.head())
print("________________________________________________________________________________

sns.displot(residuals_lasso, kind='kde')


```

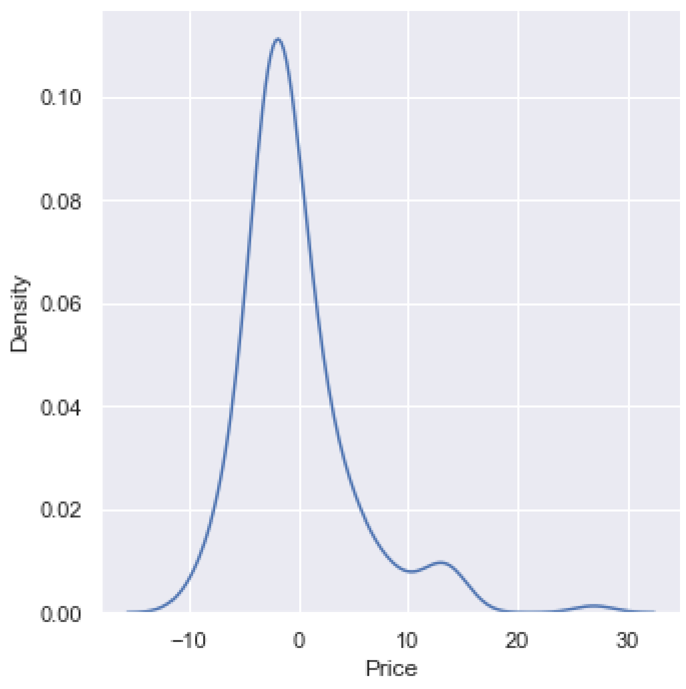
```
***Lasso.coef are--  [-0.          0.         -0.          0.27140271 -0.
2.62932147
 -0.         -0.         -0.         -0.         -1.21106809  0.29872625
 -3.81788375]
***Lasso.intercept : 22.970796460176988
____________________________________________________________________________
________
***lasso_predict [26.08015466 30.7480057  17.78164882 25.25224684 19.2838727
4 22.81161765
 18.31125182 14.6359243  21.41277818 20.44276659 20.7857368  21.00978479
  1.29101416 22.48591111 20.4207989  24.73115299 18.16643043  6.95747132
 35.82658816 18.45664358 25.66618031 26.77096265 13.79601995 24.00317031
 18.83677575 15.53225538 22.93567982 18.81410882 19.96419904 19.71394554
 19.9929271  25.48086778 25.07506471 19.62299031 15.87164442 20.47826644
 30.90020658 21.73740698 21.69357896 24.78795141 14.48946282 27.49872616
 36.28097645 19.68302782 25.54695918 17.26691093 16.01035524 25.87512519
 19.3705841  29.52965183 23.10173719 31.37342903 17.55332715 25.82107048
 34.98857199 22.91267519 19.3967501  29.34678421 24.65125376 16.72971658
 25.42537393 30.6751849  28.90511192 18.42571639 27.56426639 14.62706882
 20.02272756 25.60745002 28.32959623 15.91971307 20.36020491 26.04012236
 13.70562148 23.19186499 23.2538407   9.14791655 21.08680468 35.13203126
 18.20120981 12.40579126 23.03574753 11.70030485 24.10234373 10.23869501
 22.24788446 28.20852115 20.77401763 26.01572261 25.97666619 20.77471688
 24.05595237  9.79658092 21.55718522 20.96232324 14.58941397 22.29462592
 23.04513053  2.87810564 18.26028545 17.31405284 21.55660947 24.48282506
 11.46772233 21.88129799 25.04349207 14.07796126 19.97841644 26.61705358
 23.30429098 27.32736035 12.59741065 19.28050072 24.94727892 24.22470232
 29.72519314 19.11634391 31.14895846 16.43050137 20.50890111 27.69026978
 19.80307948 26.66386801 15.01321139 23.31466084 26.15446018 23.80801526
 27.15999771 30.37432077 22.93935948 34.91159865 11.97264266 26.45153342
 20.25377754 19.96681079 12.21677635 21.57200937 23.11587937 31.05309711
 29.72484228 18.03669387 19.12012649 28.8679226  23.41788443 14.36679948
 10.91433849 23.78530314 23.58885901 18.48704182 15.72569049 36.3867166
 19.38880373 19.56932184 27.03174387 22.95041998 22.07807906 23.22702436
 17.41528186 24.66493926 30.31735718 13.9998893  22.25446895 19.75004517
 20.8350658  25.47389672 24.13577633 23.02944605 36.90324597]
```

```
***residuals_lasso.head() 173   -2.480155
274    1.651994
491   -4.181649
72    -2.452247
452   -3.183873
Name: Price, dtype: float64
```

Out[160]:

```
<seaborn.axisgrid.FacetGrid at 0x206a9268970>
```

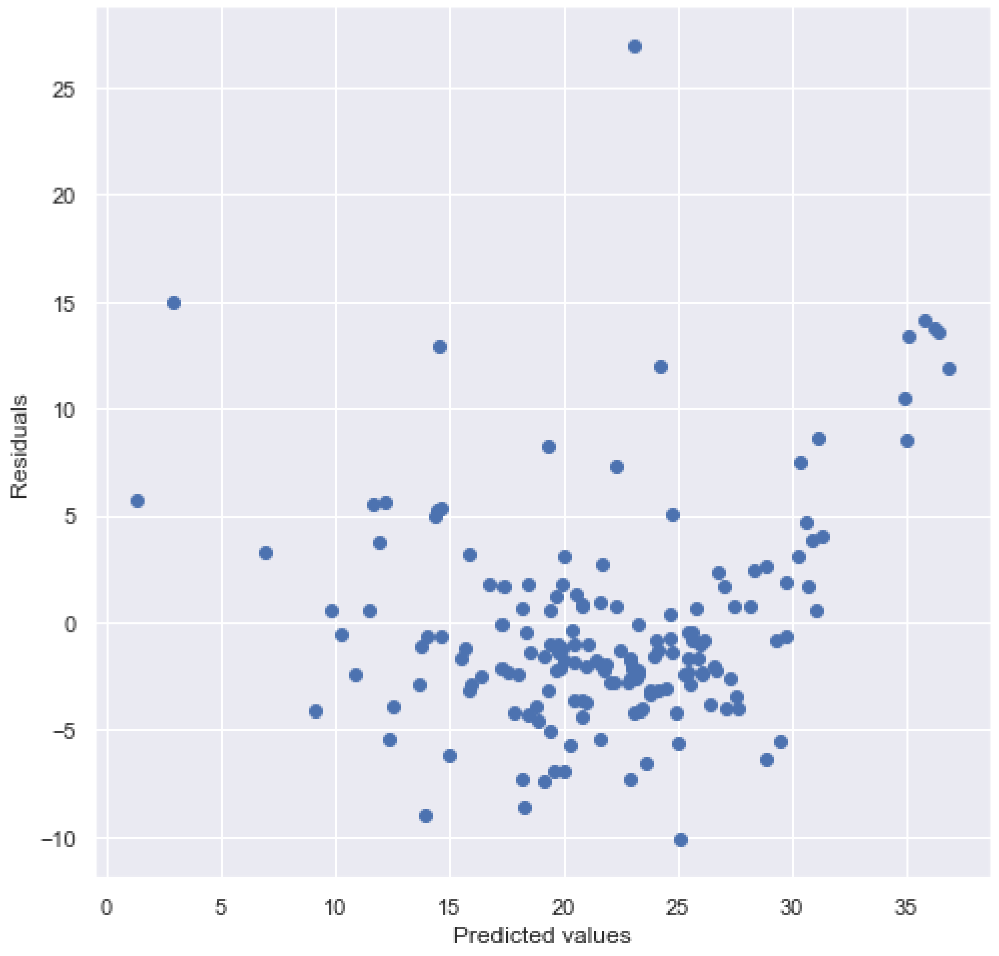



Density
0.10
0.08
0.06
0.04
0.02
0.00
−10
0
10
20
30
Price

In [171]:

```
## 3. Third Assumption-------Uniform distribution
plt.scatter(lasso_predict, residuals_lasso)
plt.xlabel('Predicted values')
plt.ylabel('Residuals')

### 4.Fourth Assumption
print("mean_squared_error : ", format(round(mean_squared_error(y_test,lasso_predict),4)
print( "mean_absolute_error:", format(round(mean_absolute_error(y_test,lasso_predict),4
print("Root mean square error : ",format(round(np.sqrt(mean_squared_error(y_test,lasso_
print("________________________________________________________________________________

## R square
score2=r2_score(y_test,lasso_predict)
print("R square Value for lasso regression is: ", format(round(score2*100,4)))
print("________________________________________________________________________________

#adjusted R-squared
Adjusted_R_lasso = 1 - (1-score2)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("Adjusted R square for lasso regression is: ", format(round(Adjusted_R_lasso*100,

```

```
mean_squared_error :  26.1664
mean_absolute_error: 3.6464
Root mean square error :  5.1153
____________________________________________________________________________
________
R square Value for lasso regression is:  65.4243
____________________________________________________________________________
________
Adjusted R square for lasso regression is:  62.4865
```



4.Elastic-net Regression

In [170]:

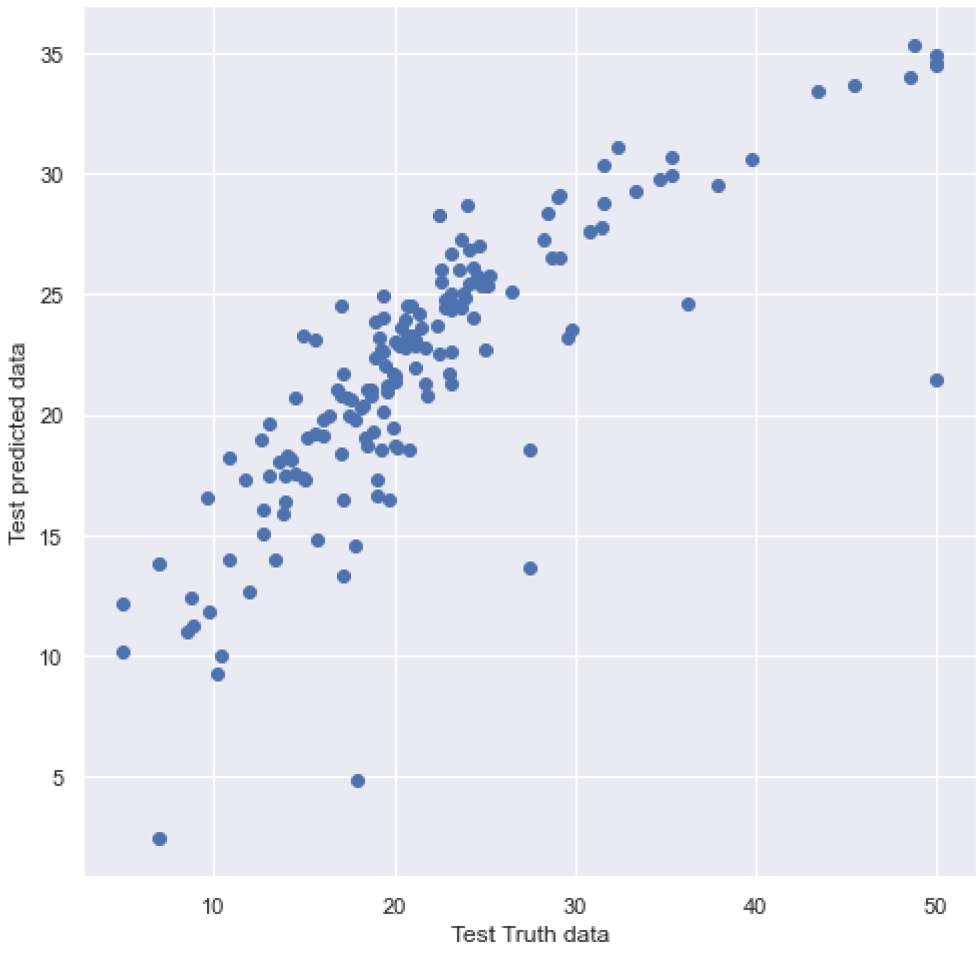
```python
from sklearn.linear_model import ElasticNet
Elesticnet = ElasticNet()
Elesticnet
Elesticnet.fit(X_train,y_train)
## print the coefficients and the intercept
print("***ElasticNet.coef are-- ",Elesticnet.coef_)
print("***ElasticNet.intercept :",Elesticnet.intercept_)
print("______________________________________________________________________
# prediction for test data
Elesticnet_predict=Elesticnet.predict(X_test)
print("***Elesticnet_predict",Elesticnet_predict)
print("______________________________________________________________________
### Assumptions of Lasso regression
### 1.First Assumption ---- Linear relationship
plt.scatter(y_test, Elesticnet_predict)
plt.xlabel('Test Truth data')
plt.ylabel('Test predicted data')

## 2.second Assumption ---- Residual distribution
residuals_Elesticnet=y_test-Elesticnet_predict
print("***residuals_Elesticnet.head()",residuals_Elesticnet.head())
print("______________________________________________________________________

sns.displot(residuals_Elesticnet, kind='kde')

```

```
***ElasticNet.coef are--  [-0.36520114  0.          -0.14336748  0.63145824
-0.25193148  2.34999448
 -0.          -0.          -0.          -0.25649969 -1.23951556  0.56384945
 -2.56053213]
***ElasticNet.intercept : 22.970796460176988
__________________________________________________________________________

__________
***Elesticnet_predict [26.04802695 31.11448131 18.09845158 24.74715491 19.
13029713 23.07195028
 19.8492127  16.42921582 20.98280883 21.03040905 23.59247585 22.4067143
  2.50342106 22.86968897 21.05836477 23.53088819 19.32942155  9.24659633
 34.51755093 18.33111982 25.39963891 26.53220506 16.04212388 23.68595117
 18.22309609 15.9070075  22.91791506 17.40135861 22.80881602 20.34960072
 21.28107265 25.0664737  23.29041734 18.52289666 16.68946719 20.17099878
 29.78000437 22.08911412 24.00624402 24.52109601 16.51539744 27.25142517
 34.8940966  20.75229792 25.54944362 17.27877681 17.51067948 25.422475
 19.45141801 28.72445431 23.85816391 30.64335445 19.05778782 25.10137208
 33.43673587 21.9368327  19.10068361 28.38705767 24.91075492 18.68821158
 25.41735754 29.96236233 27.77368373 18.66077461 26.83456776 18.72984267
 19.66634919 25.37569386 27.64862833 15.09326887 21.6230625  24.42218348
 14.00935207 22.80340884 23.31057037 10.15388121 21.41270277 33.98445117
 18.23197774 13.83630127 23.23840504 13.33915878 24.55297788 11.80396525
 22.6039322  29.04777925 19.93864988 25.40796591 25.4253774  20.79626634
 24.35937771  9.98031645 21.1883148  21.7010251  13.64158366 21.70329066
 21.48780024  4.89921219 16.60651403 16.48691364 22.52662793 24.23810699
 12.67234464 21.73288769 24.94869622 14.02785155 20.34560161 25.81816846
 23.27665603 26.98968083 12.4461064  18.53919342 24.57036836 24.5991159
 28.78917289 17.35695925 30.55890904 17.49669771 20.81635985 27.26668735
 20.67056474 26.10145269 11.29182557 23.22360335 25.76790464 23.6220014
 26.68354582 29.53505955 23.09099441 33.69060315 14.86234562 26.00700282
 20.72314768 21.04261631 14.58583453 19.80159048 23.16541552 30.31837307
 29.06727633 19.21664987 19.96431131 28.26947089 24.04394758 18.60022435
 10.97392042 23.98834146 24.56611841 18.41001674 17.5657926  34.56534242
 18.12003269 19.01050052 26.55906015 23.10161055 22.69122071 22.66142713
 17.29393495 22.68213782 29.22756893 12.14001806 23.24525465 20.77586134
 21.27148575 25.02037796 24.44501814 23.10652629 35.30973171]
```

```
***residuals_Elesticnet.head() 173   -2.448027
274    1.285519
491   -4.498452
72    -1.947155
452   -3.030297
Name: Price, dtype: float64
```

Out[170]:

```
<seaborn.axisgrid.FacetGrid at 0x206a943d250>
```

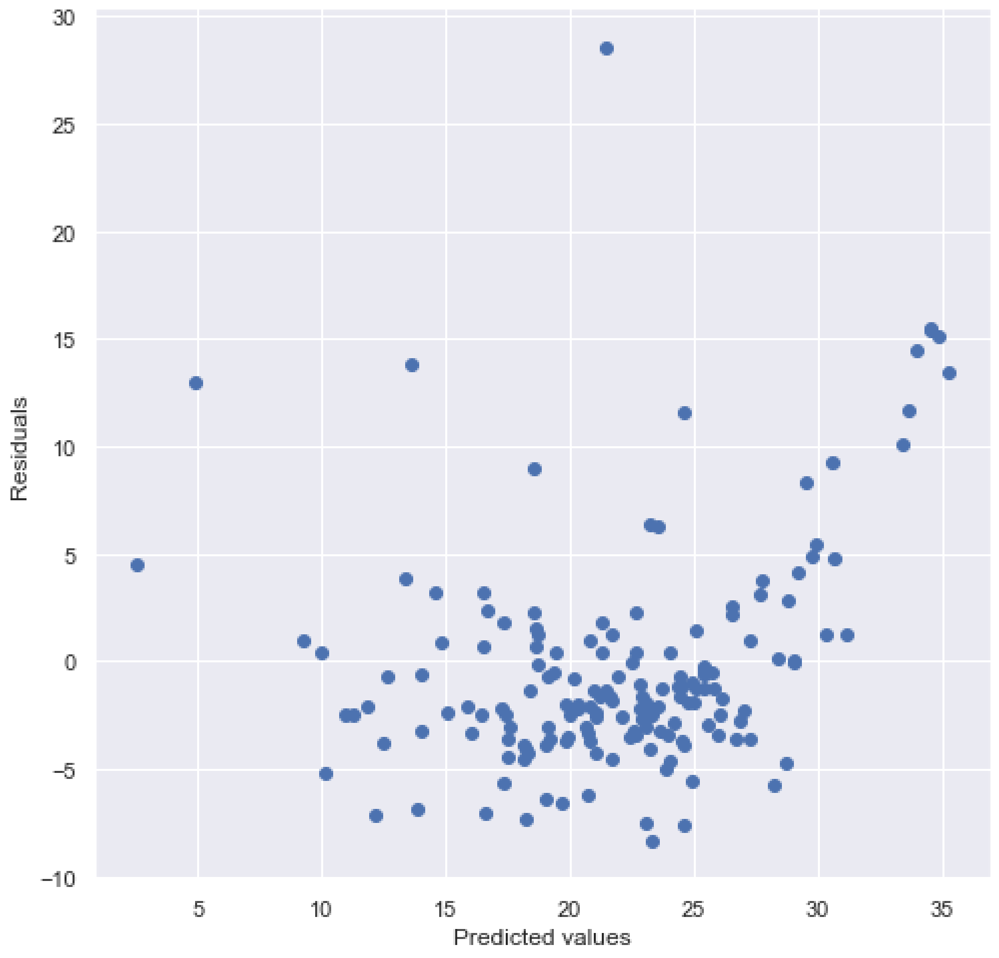


0.10
0.08

In [168]:

```
## 3. Third Assumption-------Uniform distribution
plt.scatter(Elesticnet_predict, residuals_Elesticnet)
plt.xlabel('Predicted values')
plt.ylabel('Residuals')

### 4.Fourth Assumption
print("mean_squared_error : ", format(round(mean_squared_error(y_test,Elesticnet_predic
print( "mean_absolute_error:", format(round(mean_absolute_error(y_test,Elesticnet_predi
print("Root mean square error : ",format(round(np.sqrt(mean_squared_error(y_test,Elesti
print("______________________________________________________________________________

## R square
score3=r2_score(y_test,Elesticnet_predict)
print("R square Value for Elasticnet regression is: ", format(round(score2*100,4)))
print("______________________________________________________________________________

#adjusted R-squared
Adjusted_R_Elastic = 1 - (1-score2)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("Adjusted R square for Elastnet regression is: ", format(round(Adjusted_R_Elastic

```

```
mean_squared_error :  27.1402
mean_absolute_error: 3.6277
Root mean square error :  5.2096
____________________________________________________________________________
________
R square Value for Elasticnet regression is:  65.4243
____________________________________________________________________________
________
Adjusted R square for Elastnet regression is:  62.4865
```



In [ ]:

```

```

END

In [169]:

```
## Hope You Like The Presentation Skills
```