Linear Regression on Boston Data

In [1]:

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

%matplotlib inline
```

In [2]:

```
# Let's Load the Boston Data

from sklearn.datasets import load_boston
```

In [3]:

```
boston=load_boston()
boston.keys()
```

```
C:\Users\anubh\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function load_boston is deprecated; `load_boston` is deprecat
ed 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)
```

Out[3]:

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

In [4]:

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

```
.. _boston_dataset:

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

**Data Set Characteristics:**

    :Number of Instances: 506

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

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river;
0 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 peo
ple 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/ (https://
archive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carne
gie 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 diagnostic
s
...', 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 th
at address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
```

```
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. I
n Proceedings on the Tenth International Conference of Machine Learning, 236
-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

Convert into Dataframe

In [5]:

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

In [6]:

```
df.head()
```

Out[6]:

	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

Add price Column into data set

In [7]:

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

In [8]:

```
df.head()
```

Out[8]:

	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

Mandatory EDA

In [9]:

```
df.info()
```

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

In [10]:

```
df.isnull().sum()
```

Out[10]:

```
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
```

We don't have missing value and data type for all feature is float

In [11]:

```
df.describe()
```

Out[11]:

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

Observation: Here we have CRIM, ZN have alots outlier. Because the max value is quite high according to 75%.

In [12]:

```
df.corr()
```

Out[12]:

	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

Univariate analysis

In [13]:

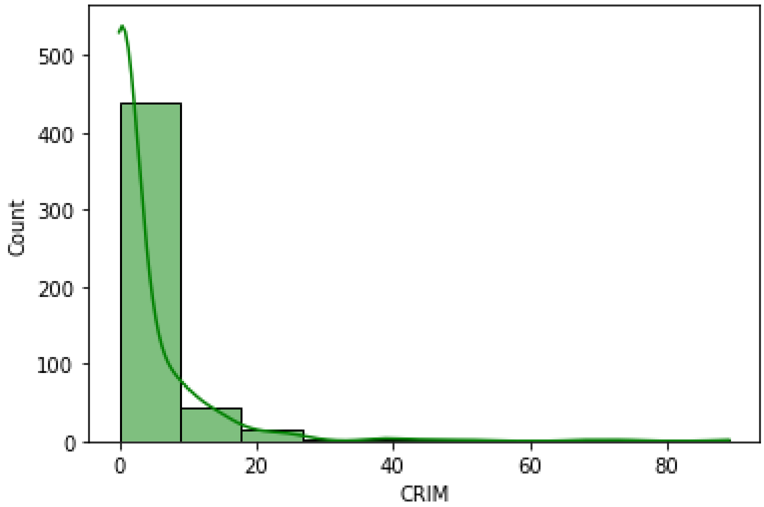
```
numerical_features=[feature for feature in df.columns if df[feature].dtype!='Object']

print('We have total {} numerical features and the feature is{}'.format(len(numerical_f
```

```
We have total 14 numerical features and the feature is['CRIM', 'ZN', 'INDU
S', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTA
T', 'Price']
```

In [14]:

```
for feature in numerical_features:
    sns.histplot(data=df,x=feature,kde=True,bins=10,color='g')
    plt.show()
```



Observation on univariate analysis

CRIM,ZN,CHAS,NOX,DIS,RAD,TAX and LSTAT are following log normal(distribution left skewed).

Price & RM following normal distribution.

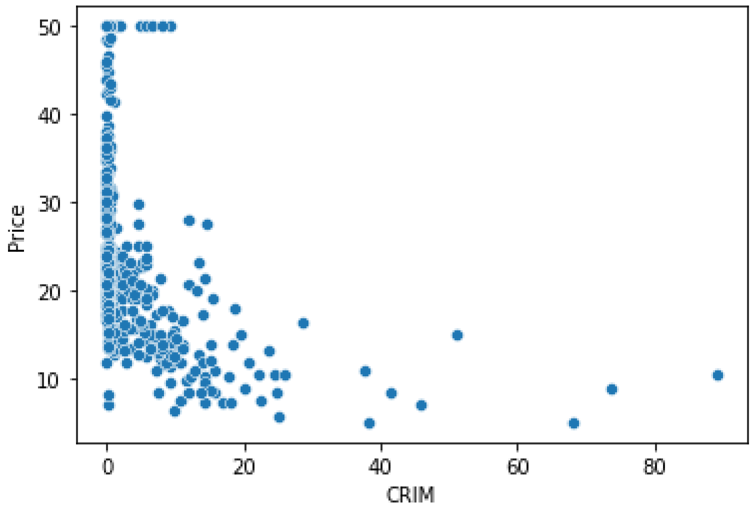
B & Age following right skewed disbribution.

Bivariate analysis

In [15]:

```
for feature in numerical_features:
    sns.scatterplot(data=df,x=feature,y='Price',legend='auto')

    plt.show()
```



Observation on bivariate analyis

1. Crime rate increase price decrease

2. ZN distributed normally.

3. Indus increase price decrease

4. Nox increase price decrease

5. Rm increase price increase

6. Age increase price decrease

7. Dis increase price increase

8. Rad decrease price increase

9. Ptration increase price increase

10. B decrease price decrease

11. Lstat increase price decrease

In [16]:

```
plt.figure(figsize=(12,10))
sns.heatmap(df.corr(),annot=True)
```

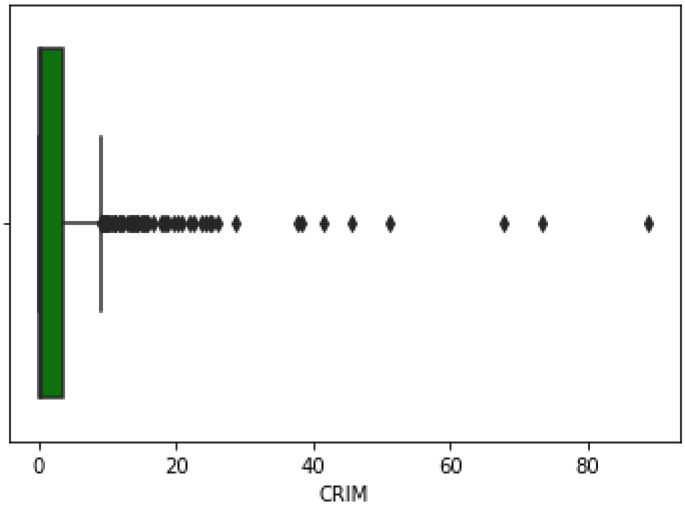
Out[16]:

```
<AxesSubplot:>
```

Find outlier in feature

In [17]:

```
for feature in numerical_features:
    sns.boxplot(data=df,x=feature,color='g')
    plt.show()
```

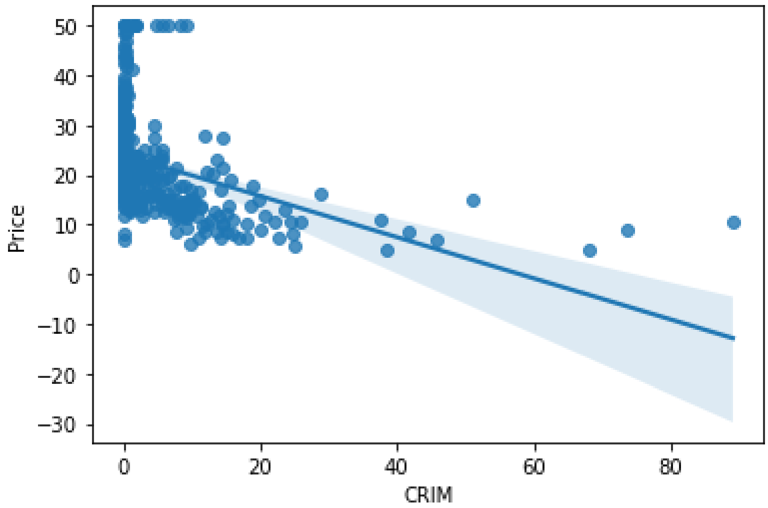


Observation: Crim, Zn,Chas,rm,dis,ptratio,B,lstat we have outlier

Check the Relation and best fit line.

In [18]:

```
for feature in numerical_features:
    sns.regplot(data=df,x=feature,y='Price')
    plt.show()
```



obseration: If we have outlier so shaed region is high. or we can say in simple words, where the point satruation is high there is less movement in best fit line.

Let's implement the Linear Regression

Divede the data into Train and Test

Perform standardization

In [19]:

```

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

In [20]:

```
X  # alwasy note we get all independent feature as dataframe.
```

Out[20]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90
...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90
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
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90

506 rows × 13 columns

In [21]:

```
y    # We get dependent feature as a series
```

Out[21]:

```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: Price, Length: 506, dtype: float64
```

In [22]:

```
from sklearn.model_selection import train_test_split

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

In [23]:

```
X_train.shape
```

Out[23]:

```
(339, 13)
```

In [24]:

```
y_train.shape
```

Out[24]:

```
(339,)
```

In [25]:

```
X_test.shape
```

Out[25]:

```
(167, 13)
```

In [26]:

```
y_test.shape
```

Out[26]:

```
(167,)
```

Perform standardization

In [27]:

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

In [28]:

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

In [29]:

```
X_train
```

Out[29]:

```
array([[-0.13641471, -0.47928013,  1.16787606, ..., -1.77731527,
         0.39261401,  2.36597873],
       [-0.41777807, -0.47928013, -1.18043314, ..., -0.75987458,
         0.14721899, -0.54115799],
       [ 1.31269177, -0.47928013,  0.95517731, ...,  0.76628645,
         0.19334986,  2.52100705],
       ...,
       [-0.13520965, -0.47928013,  0.95517731, ...,  0.76628645,
         0.17012536,  0.06331026],
       [-0.40281114, -0.47928013,  2.04022838, ...,  0.25756611,
         0.32166792,  0.27238516],
       [-0.33104058,  0.34161649, -1.07552092, ..., -2.56351944,
         0.39993132, -0.34772815]])
```

In [30]:

```
X_test
```

Out[30]:

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

Super important ! Interview question: why we don't perform fit_transform on X_test data.

Ans: Because: To avoid Data leakage.

Model training

In [34]:

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

In [35]:

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

Out[35]:

```
LinearRegression()
```

In [36]:

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

In [37]:

```
reg_pred
```

Out[37]:

```
array([31.43849583, 31.98794389, 30.99895559, 22.31396689, 18.89492791,
       16.21371128, 35.9881236 , 14.81264582, 25.04500847, 37.12806894,
       21.49110158, 30.88757187, 28.05752881, 34.05600093, 33.75791114,
       40.63880011, 24.24023412, 23.41351375, 25.54158122, 21.34135664,
       32.71699711, 17.88341061, 25.49549436, 25.01006418, 32.54102925,
       20.48979076, 19.48816948, 16.92733183, 38.38530857,  0.36265208,
       32.42715816, 32.15306983, 26.10323665, 23.79611814, 20.67497128,
       19.69393973,  3.50784614, 35.26259797, 27.04725425, 27.66164435,
       34.35132103, 29.83057837, 18.40939436, 31.56953795, 17.91877807,
       28.50042742, 19.49382421, 21.69553078, 38.0954563 , 16.44490081,
       24.58507284, 19.67889486, 24.53954813, 34.30610423, 26.74699088,
       34.87803562, 21.06219662, 19.87980936, 18.68725139, 24.71786624,
       19.96344041, 23.56002479, 39.57630226, 42.81994338, 30.37060855,
       17.03737245, 23.83719412,  3.2425022 , 31.5046382 , 28.63779884,
       18.49288659, 27.14115768, 19.67125483, 25.34222917, 25.05430467,
       10.29463949, 38.96369453,  8.26774249, 18.52214761, 30.34082002,
       22.87681099, 20.96680268, 20.04604103, 28.73415756, 30.81726786,
       28.23002473, 26.28588806, 31.59181918, 22.13093608, -6.48201197,
       21.53000756, 19.90826887, 24.96686716, 23.44746617, 19.28521216,
       18.75729874, 27.40013804, 22.17867402, 26.82972   , 23.39779064,
       23.9260607 , 19.16632572, 21.09732823, 11.01452286, 13.7692535 ,
       20.74596484, 23.54892211, 14.04445469, 28.88171403, 15.77611741,
       15.25195598, 22.429474  , 26.60737213, 28.88742175, 24.29797261,
       18.26839956, 16.26943281, 17.40100292, 15.53131616, 21.27868825,
       33.78464602, 30.00899396, 21.16115702, 13.95560661, 16.18475215,
       29.30998858, 13.1866784 , 22.08393725, 24.34499386, 31.86829501,
       33.45923602,  5.90671516, 35.20153265, 24.17614831, 17.54200544,
       24.25032915, 28.44671354, 34.50123773,  6.33164665,  1.93565618,
       28.40727267, 12.56461105, 18.31045646, 19.71015745,  5.50105857,
       14.51366874, 37.193992  , 25.81821367, 23.31632083, 26.43254504,
       11.38255141, 20.46224115, 35.27645709, 20.57841598, 11.48799917,
       16.23913171, 24.56511742, 10.53131603, 15.07115005, 25.98488217,
       11.2136222 , 11.695686  , 19.40437966, 19.58768384, 32.43800883,
       22.66170871, 25.68576052])
```

In [42]:

```
print(regression.coef_)

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

```
[-1.29099218  1.60949999 -0.14031574  0.37201867 -1.76205329  2.22752218
  0.32268871 -3.31184248  2.70288107 -2.09005699 -1.7609799   1.25191514
 -3.83392028]
13
```

In [40]:

```
regression.intercept_
```

Out[40]:

```
22.077286135693246
```

Obeservation : If the value of all coefficient is zero, then price value is 22.077 as per model

if there is -1.29099 movement in Crim then price will change 22.077. similalry with all coefficient.

Assumption of Linear Regression

1. there is linear realtionship between truth value and pridicted value

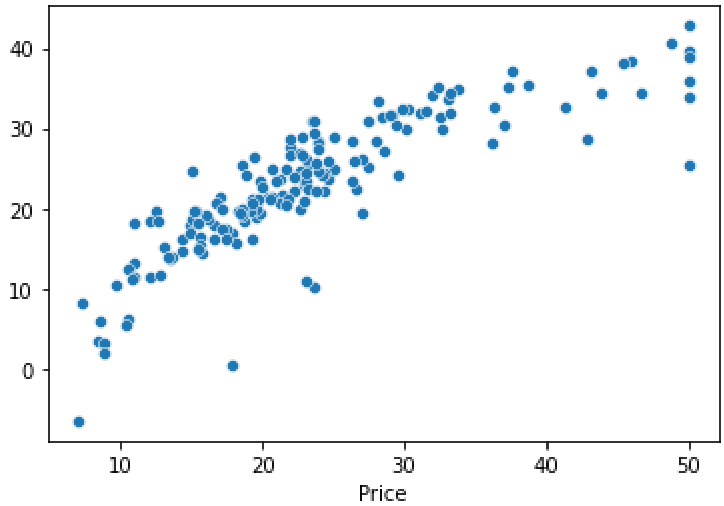
In [45]:

```
sns.scatterplot(y_test,reg_pred)
```

```
C:\Users\anubh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
```

Out[45]:

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



2. if we calculate the residual(error) and plot. it should follow the normal distribution.

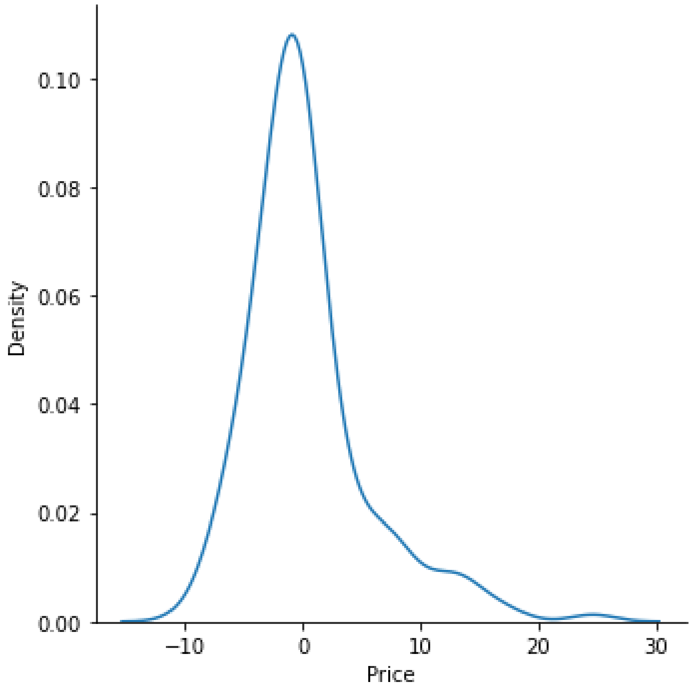
In [46]:

```
residual=y_test-reg_pred

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

Out[46]:

```
<seaborn.axisgrid.FacetGrid at 0x195d22baca0>
```



3. if we plot on graph the pridicted value and residual. they should be follow unifrom distribution.

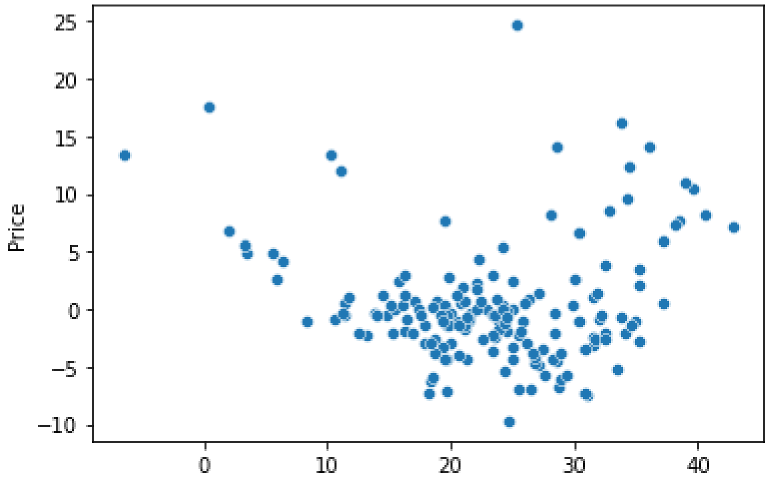
In [47]:

```
1  sns.scatterplot(reg_pred,residual)
```

```
C:\Users\anubh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
```

Out[47]:

<AxesSubplot:ylabel='Price'>



Performance matrix

1. MSE (mean square error)

2. MAE (mean absoulte error)

3. RSME (root mean square error)

In [48]:

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

In [52]:

```
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)))
```

```
27.100991709962475
3.520658529879791
5.205861284164463
```

R square and Adjusted R Square

In [56]:

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

```
0.7165219393967556
```

In [57]:

```
# adjusted R square

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

Out[57]:

```
0.6905827704526679
```

In [ ]:

```

```