First Linear Regression Model

Importing the basic libraries

In [59]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings(action='ignore', category=FutureWarning)
```

Importing the dataset from sklearn The name of the dataset is boston

In [2]:
```
from sklearn.datasets import load_boston
```

In [60]:
```
df=load_boston()
```

Checking the keys of dataset

In [6]:
```
df.keys()
```

Out[6]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename', 'data_modul
e'])
```

Description of the dataset

In [8]:
```
1 print(df.DESCR)
```

```
.. _boston_dataset:

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

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (a
ttribute 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 peop
le 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://a
rchive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carneg
ie 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 diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers tha
t 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. In
Proceedings on the Tenth International Conference of Machine Learning, 236-24
3, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [12]:

```python
print(df.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 [11]:
```
print(df.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]
```

Getting the name of the features

In [13]:
```
print(df.feature_names)
```

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

Creating the dataset from array of boston data frame

In [18]:
```
data=pd.DataFrame(df.data,columns=df.feature_names)
```

Adding the price columns to our dataset

In [20]:

```python
data['Price']=df.target
```

Viewing the top 5 rows of dataset

In [21]:

```python
data.head()
```

Out[21]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Checking the dytpes of the columns

In [24]:

```python
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
```

Checking the statistics of the numerical columns

In [25]:
```
data.describe()
```

Out[25]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.79
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.10
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.12
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.10
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.20
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.18
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.12

In [34]:
```
data.shape
```

Out[34]:
```
(506, 14)
```

EDA

Checking the null values in the dataset

In [29]:
```
data.isna().sum()
```

Out[29]:
```
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
```

Checking the correlation of the dataset

In [37]:

```
data.corr()
```

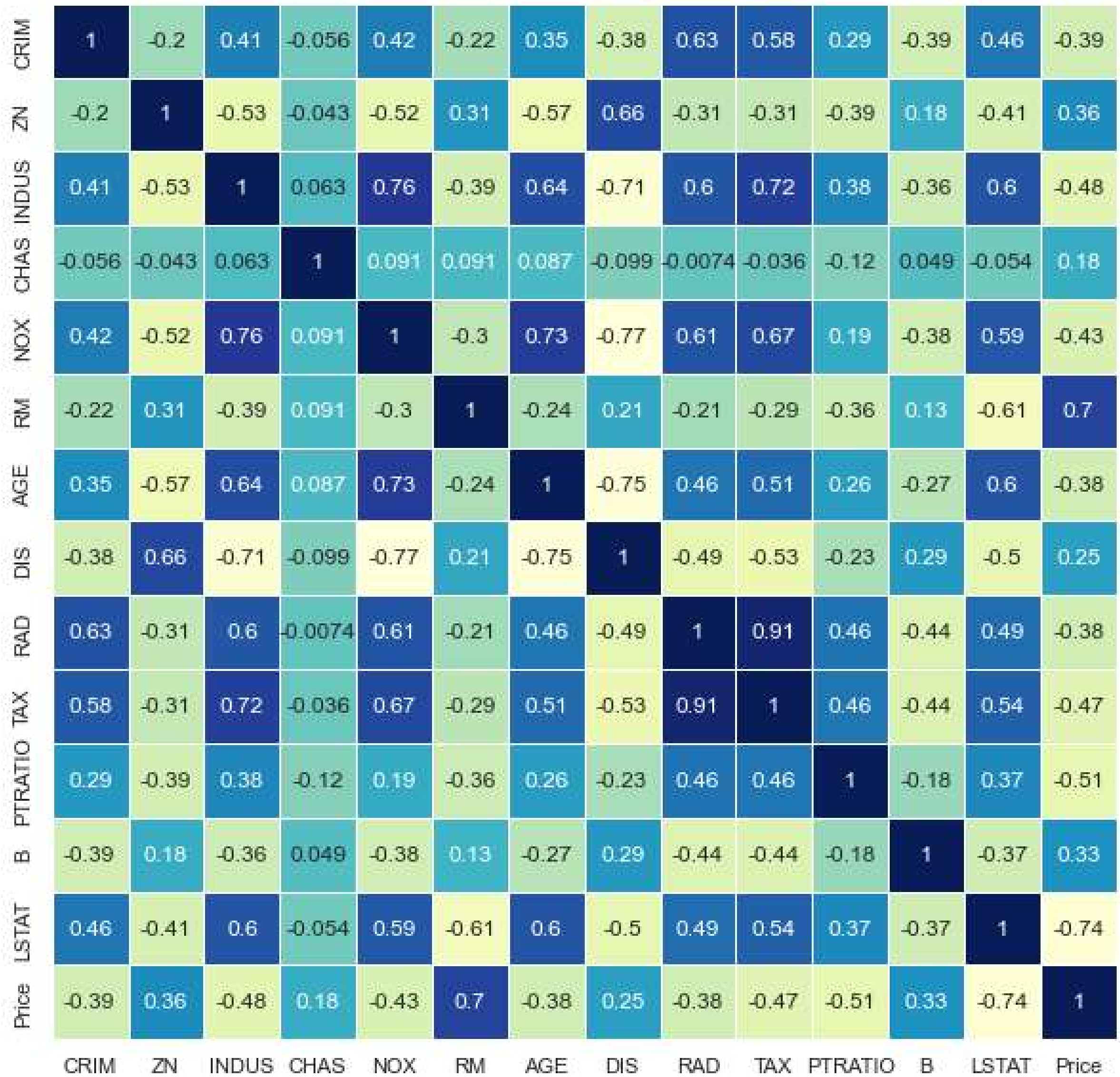
Out[37]:

	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	0.
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.
Price	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.

In [42]:

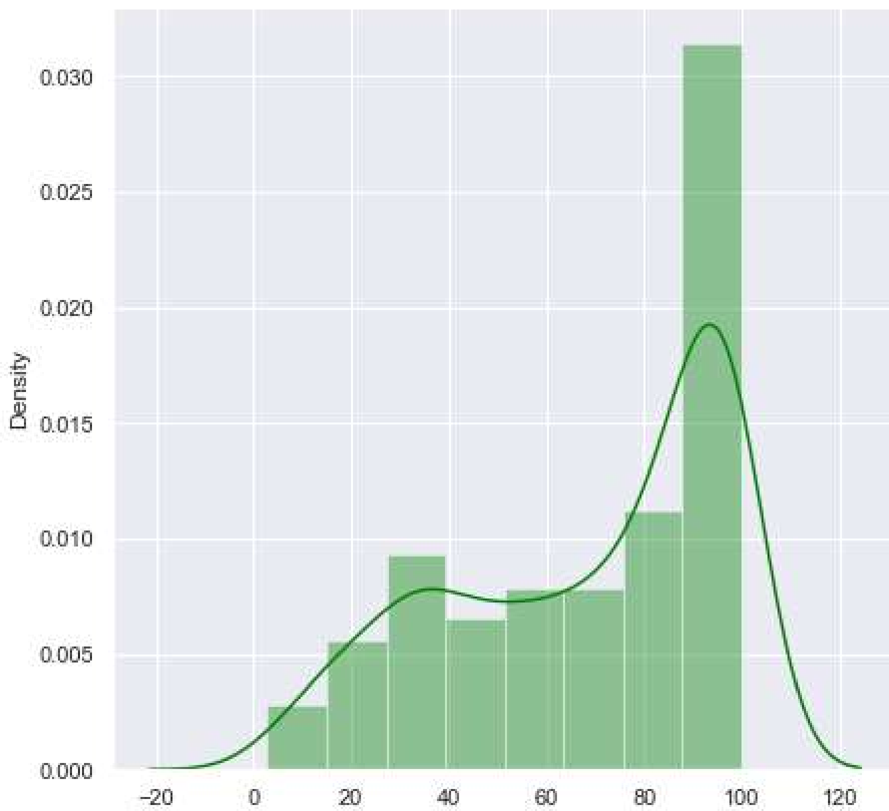
```
sns.set(rc={'figure.figsize':(10,10)})
sns.heatmap(data.corr(),annot=True,cmap="YlGnBu",linewidths=.5,cbar=False)
```

Out[42]: <AxesSubplot:>

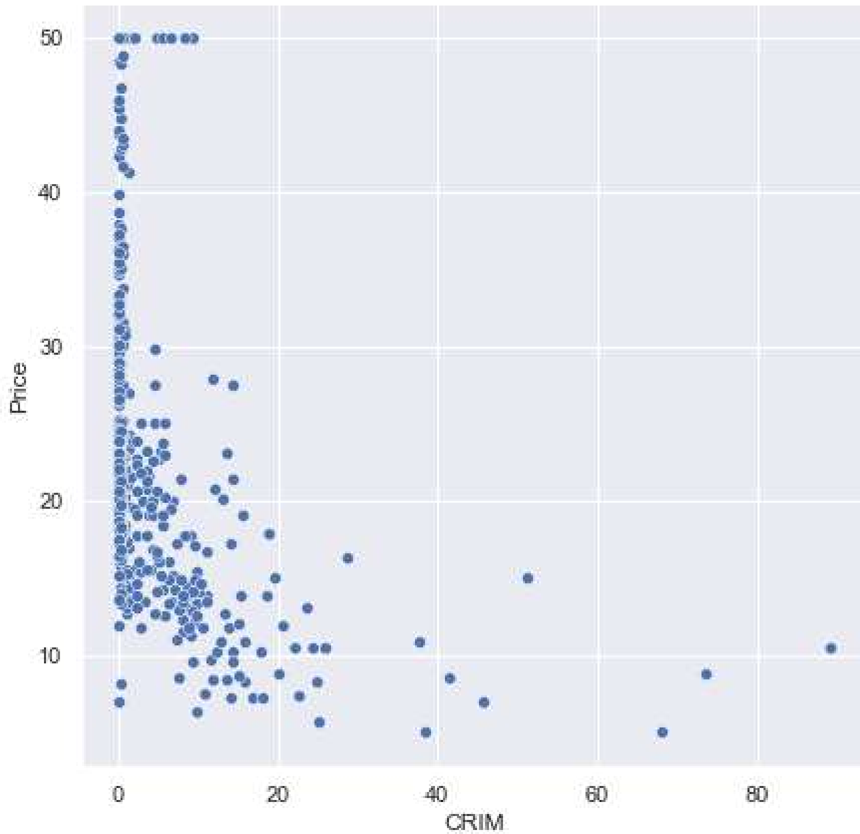


In [61]:
```
sns.set(rc={'figure.figsize':(7,7)})
sns.distplot(x=data["AGE"],hist=True,kde=True,color="GREEN")
```

Out[61]: <AxesSubplot:ylabel='Density'>



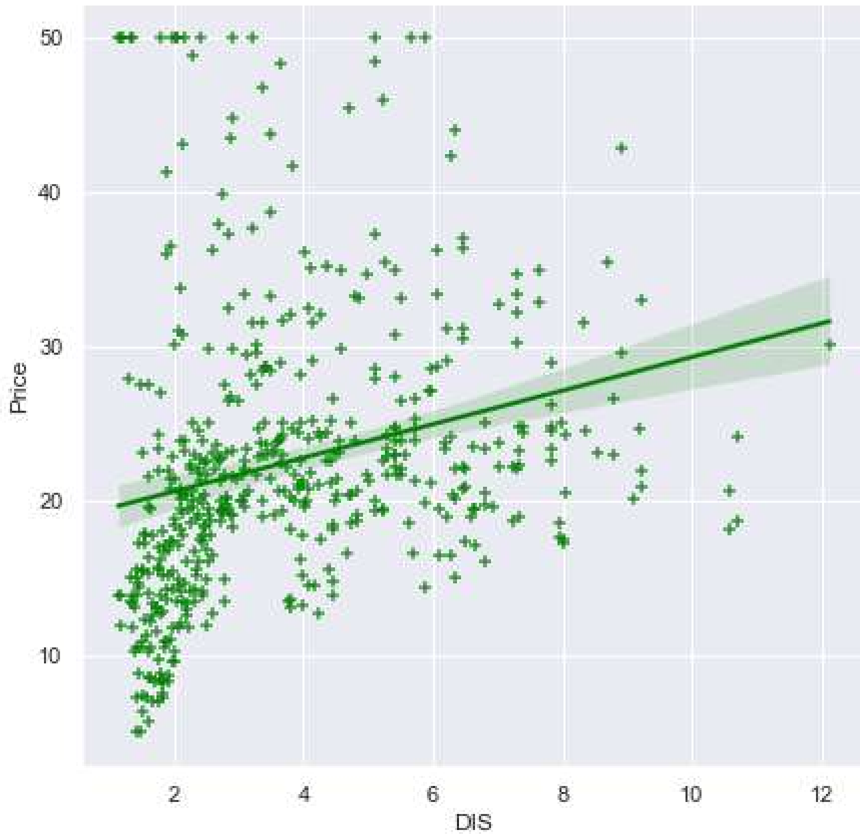
In [57]:
```
sns.scatterplot(data=data,x="CRIM",y="Price")
```
Out[57]: <AxesSubplot:xlabel='CRIM', ylabel='Price'>



In [67]:

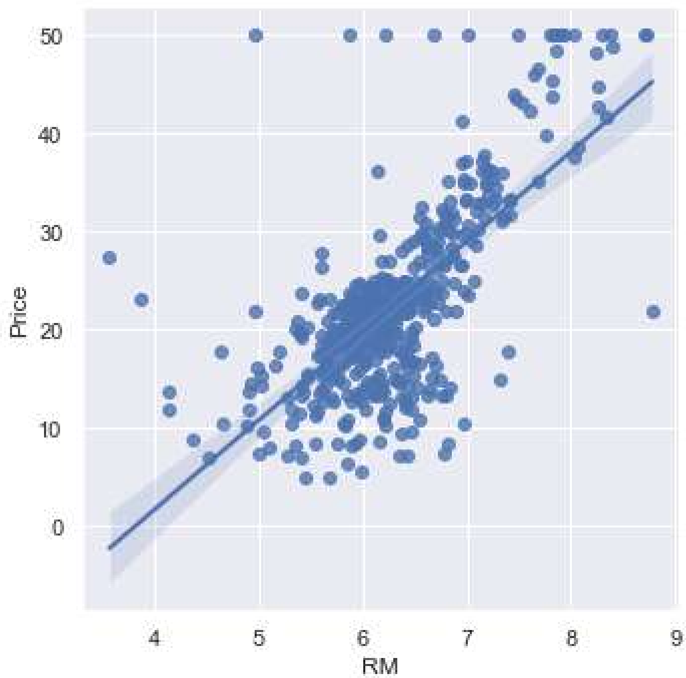
```
1 sns.regplot(data=data,x="DIS",y="Price",scatter=True,marker="+",color="green
```

Out[67]: <AxesSubplot:xlabel='DIS', ylabel='Price'>



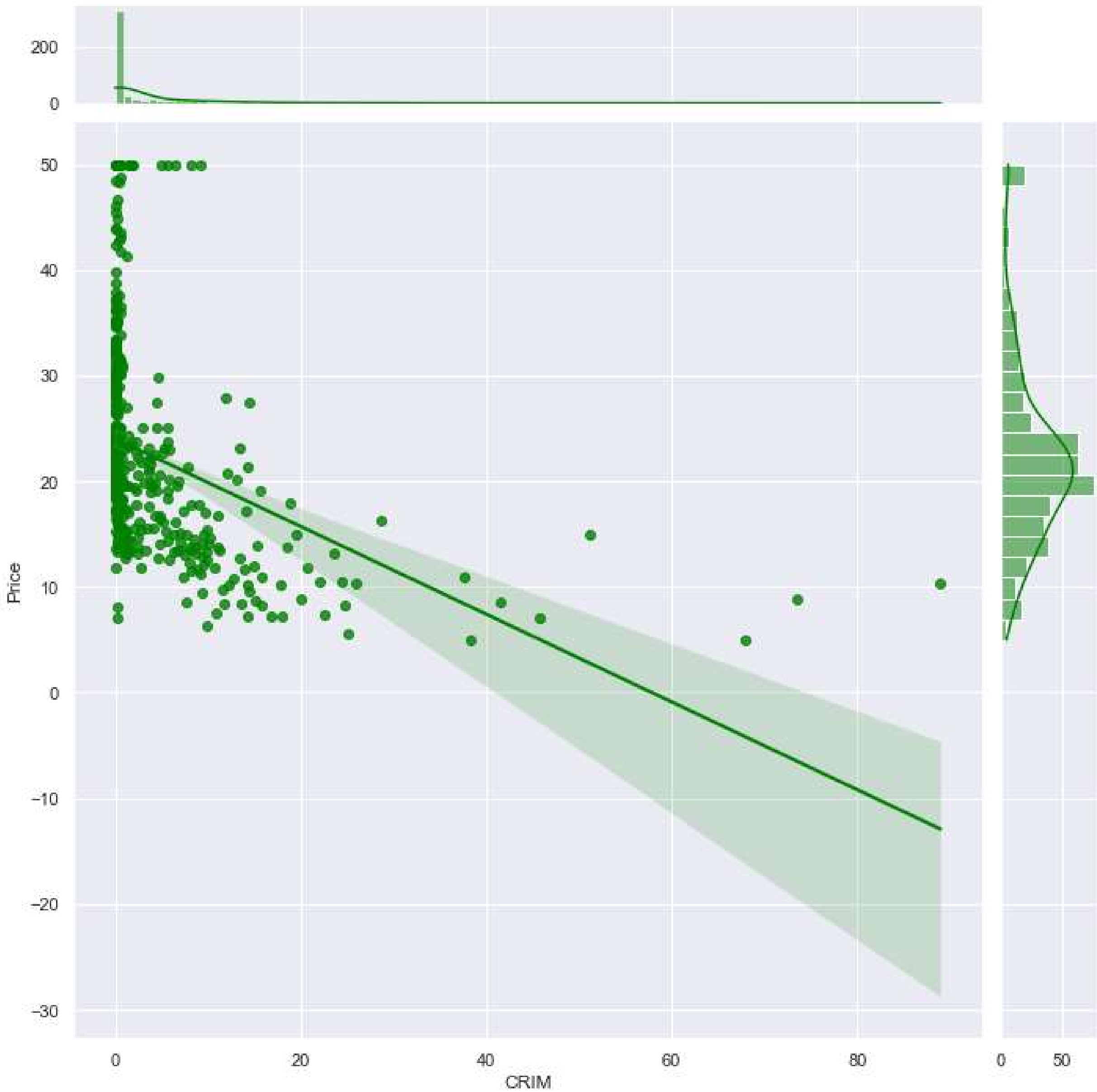
In [72]:
```
sns.lmplot(data=data,x="RM",y="Price")
```

Out[72]: <seaborn.axisgrid.FacetGrid at 0x1b3bf0d0d00>



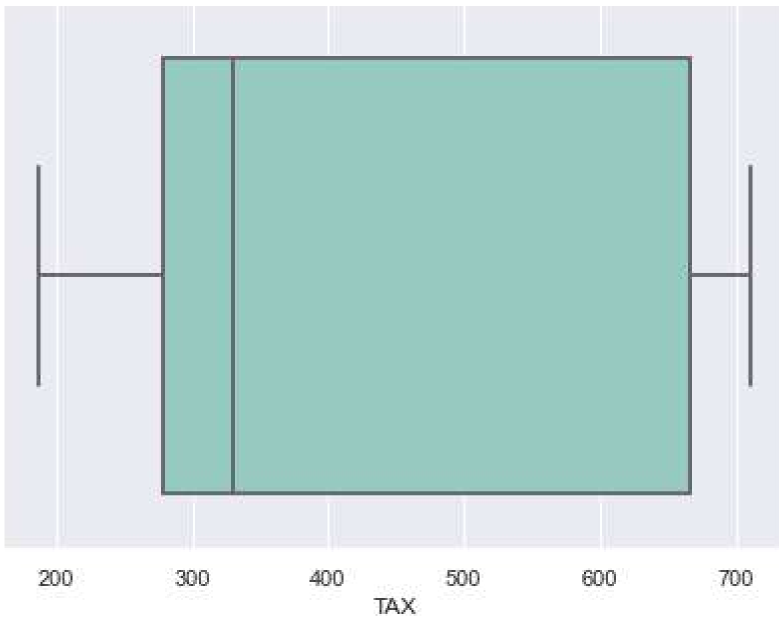
In [82]:
```
1 sns.jointplot(data=data,x="CRIM",y="Price",kind="reg",color="green",height=1
```

Out[82]: <seaborn.axisgrid.JointGrid at 0x1b3c05a5700>



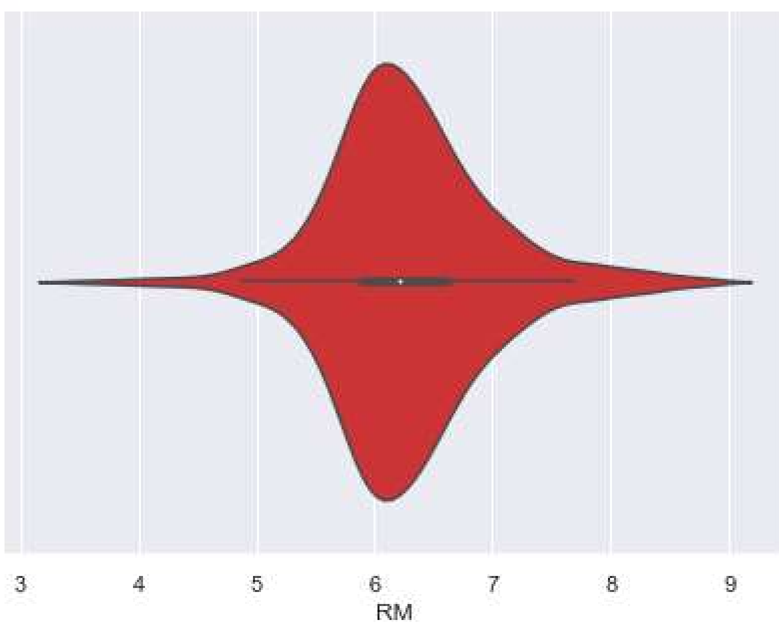
In [84]:
```
sns.set(rc={'figure.figsize':(7,5)})
sns.boxplot(data['TAX'],color="green",linewidth=2,palette="Set3")
```

Out[84]: <AxesSubplot:xlabel='TAX'>



In [90]:
```
sns.violinplot(x=data["RM"],palette="Set1")
```

Out[90]: <AxesSubplot:xlabel='RM'>



Spliting the features and target in x and y variables

In [92]:
```
x=data.iloc[:,:-1]
y=data.iloc[:,-1]
```

In [93]:

```
x
```

Out[93]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
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.
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.

506 rows × 13 columns

In [96]:

```
y

```

Out[96]:

```
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
```

Spliting the dataset into train and test part

In [97]:

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

In [125]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_sta
```

In [152]:
```
x_train
```

Out[152]:
```
array([[-0.45098379,  0.59163708, -0.84258861, ...,  0.55738858,
         0.37807205, -0.82757883],
       [ 7.93012362, -0.50215615,  1.06054133, ...,  0.80335661,
        -3.79668193, -0.32645016],
       [ 0.46202474, -0.50215615,  1.06054133, ...,  0.80335661,
         0.40378237, -0.6918886 ],
       ...,
       [-0.45261131,  0.04474047, -0.44053957, ..., -1.65632372,
         0.44498343,  1.06745852],
       [-0.45759249, -0.50215615,  0.44279445, ..., -1.06600044,
         0.41098987, -0.26631472],
       [-0.42648556, -0.50215615, -0.51243885, ...,  0.50819497,
         0.44498343, -0.93705617]])
```

In [101]:
```
x_train.shape
```

Out[101]:
```
(354, 13)
```

In [102]:
```
x_test.shape
```

Out[102]:
```
(152, 13)
```

In [105]:
```
y_test.shape
```

Out[105]:
```
(152,)
```

In [106]:
```
y_train.shape
```

Out[106]:
```
(354,)
```

Standardize or feature scaling the dataset

In [107]:
```
from sklearn.preprocessing import StandardScaler
```

In [122]:
```
scaler=StandardScaler()
```

In [126]:
```
x_train=scaler.fit_transform(x_train)
```

In [127]:
```
x_test=scaler.transform(x_test)
```

In [128]:
```
x_train
```

Out[128]:
```
array([[-0.45098379,  0.59163708, -0.84258861, ...,  0.55738858,
         0.37807205, -0.82757883],
       [ 7.93012362, -0.50215615,  1.06054133, ...,  0.80335661,
        -3.79668193, -0.32645016],
       [ 0.46202474, -0.50215615,  1.06054133, ...,  0.80335661,
         0.40378237, -0.6918886 ],
       ...,
       [-0.45261131,  0.04474047, -0.44053957, ..., -1.65632372,
         0.44498343,  1.06745852],
       [-0.45759249, -0.50215615,  0.44279445, ..., -1.06600044,
         0.41098987, -0.26631472],
       [-0.42648556, -0.50215615, -0.51243885, ...,  0.50819497,
         0.44498343, -0.93705617]])
```

In [129]:
```
x_test
```

Out[129]:
```
array([[ 0.37877319, -0.50215615,  1.06054133, ...,  0.80335661,
         0.42916997, -0.11520515],
       [-0.46870399,  2.99798219, -1.37229555, ..., -2.9353575 ,
        -0.01317667, -0.55774031],
       [-0.43463958, -0.50215615, -0.58140346, ..., -0.32809634,
         0.44498343,  2.865354  ],
       ...,
       [-0.44178915,  0.37287844, -1.10670841, ..., -1.80390454,
         0.44498343, -1.13750764],
       [-0.4670962 ,  1.24791302, -0.65477008, ..., -0.47567716,
         0.44498343, -1.34104298],
       [-0.31388118, -0.50215615, -0.4009216 , ...,  1.19690546,
         0.41959583,  1.18001973]])
```

Model Building

In [130]:
```python
from sklearn.linear_model import LinearRegression
```

In [131]:
```python
lr=LinearRegression()
```

In [132]:
```python
lr
```

Out[132]:
```
LinearRegression()
```

In [134]:
```python
lr_model=lr.fit(x_train,y_train)
```

In [135]:
```python
lr_model.score(x_train,y_train)
```

Out[135]:
```
0.7431215456774967
```

In [157]:
```python
Linear_regression_coefficent=lr_model.coef_
```

In [158]:
```
Linear_regression_coefficent.transpose
```

Out[158]: `<function ndarray.transpose>`

In [159]:
```
Linear_regression_coefficent
```

Out[159]:
```
array([-0.6208519 ,  0.89604528, -0.4181019 ,  0.85794528, -1.98345156,
        2.34054146, -0.14708338, -2.8644969 ,  2.15413705, -1.58410776,
       -1.74439973,  0.6305477 , -3.22010917])
```

In [181]:
```
Lr_coefficent=pd.DataFrame(data=df.feature_names,columns=["Independent facto

```

In [182]:
```
Lr_coefficent["coefficent"]=Linear_regression_coefficent
```

In [183]:
```
Lr_coefficent
```

Out[183]:

	Independent factors	coefficent
0	CRIM	-0.620852
1	ZN	0.896045
2	INDUS	-0.418102
3	CHAS	0.857945
4	NOX	-1.983452
5	RM	2.340541
6	AGE	-0.147083
7	DIS	-2.864497
8	RAD	2.154137
9	TAX	-1.584108
10	PTRATIO	-1.744400
11	B	0.630548
12	LSTAT	-3.220109

In [137]:
```
lr_model.intercept_
```

Out[137]: 22.331355932203394

In [184]:
```
predicted_values=lr_model.predict(x_test)
```

In [185]:
```
predicted_values
```

Out[185]:
```
array([21.90897572, 32.36829283,  9.38919345, 16.40673353, 17.80964232,
       31.83838312, 25.10363218, 15.4942598 , 21.82825591, -3.63190569,
       26.12960431, 15.57300292,  5.61225053,  5.58756072, 25.41154332,
       34.70503462, 26.17912943, 19.13532445, 23.91967422, 14.91252997,
       39.53465438, 11.07641307, 36.58914352, 26.00446715, 38.64469005,
       25.17973575, 21.75528189, 18.96547913, 18.27571802, 18.60093947,
       24.62357132, 23.66620392, 29.6987949 , 24.08585329,  0.50581275,
       24.63764742, 25.21913509, 12.19902726, 39.4812705 , 32.23454473,
       23.75474746,  7.056712  , 20.39810217, 21.0026853 , 31.32729178,
        7.46193071, 12.70824342, 31.32832609, 22.40993904, 35.817382  ,
       12.81513925, 20.71658302, 18.48252207,  7.65314991,  6.48378445,
       40.45412148, 24.95009747, 24.17943728, 23.04271387,  7.56345617,
       22.86100568,  9.73479018, 32.957889  , 14.06778493, 28.52717573,
       17.20171167,  3.61911076, 28.62629983, 19.42447388, 18.72979294,
       19.36051351, 27.88976576, 21.11155756, 27.95264386, 34.21722447,
       20.00321067, 13.39987071, 24.70136312, 16.70346939, 22.70991552,
       18.90523529, 17.48644847, 18.59401509, 10.09315724, 16.78988769,
       10.9323577 , 17.072616  , 20.4084587 , 20.34209532, 19.17179969,
       27.40674633,  7.73791552, 20.11982554,  5.18969717, 20.2172659 ,
        5.18968724, 17.55397823, 27.00736521, 23.00701384, 20.51409442,
       24.44825342, 16.23006422, 24.80902733,  5.95769186, 31.26003941,
       21.69010225, 30.42222709, 31.82079134, 21.90982483, 17.40866927,
       29.98143286, 39.96512278, 27.59995934, 22.03687566, 22.24622018,
       14.94470564, 21.28339441, 19.3278602 , 41.8844113 , 21.35566086,
       22.53418169, 28.98276109, 25.20250497, 16.47346987, 41.69742145,
       18.17821943, 13.59812528, 24.6032745 , 16.28450795, 28.54125121,
       13.22639708, 26.69967006, 30.09184714, 23.06307131, 33.64391574,
       34.87452978, 19.21096706, 20.22625076, 13.38442001, 19.58013994,
       12.97469066, 35.1171442 , 16.93050452, 25.13534002, 27.26167899,
       22.67914379, 14.24375693, 23.22069174, 14.87303988, 34.55202866,
       36.1305225 , 13.89590962])
```

In [187]:
```
Comparsion_table=pd.DataFrame(data=predicted_values,columns=['Predicted valu
```

In [189]:
```
Comparsion_table["Actual Value"]=y_test
```

In [191]:
```
Comparsion_table["Difference"]=Comparsion_table["Actual Value"]-Comparsion_t
```

In [200]:
```
Comparsion_table.dropna().head()
```

Out[200]:

	Predicted values	Actual Value	Difference
0	21.908976	24.0	2.091024
3	16.406734	33.4	16.993266
4	17.809642	36.2	18.390358
6	25.103632	22.9	-2.203632
9	-3.631906	18.9	22.531906

Performance Metrics

In [210]:

```python
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print("Mean squared error:",mean_squared_error(y_test,predicted_values))
print("Mean absolute error:",mean_absolute_error(y_test,predicted_values))
print("Root mean squared error:",np.sqrt(mean_squared_error(y_test,predicted_
```

```
Mean squared error: 31.829631155557482
Mean absolute error: 3.9079661456255192
Root mean squared error: 5.641775532184658
```

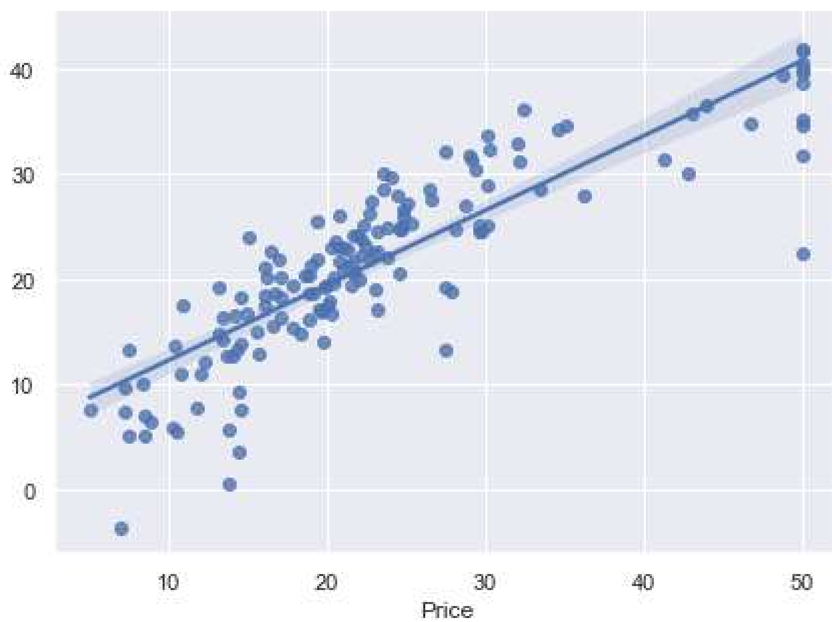
Assumptions Of Linear Regression

In [211]:

```python
sns.regplot(x=y_test,y=predicted_values)
```

Out[211]:

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



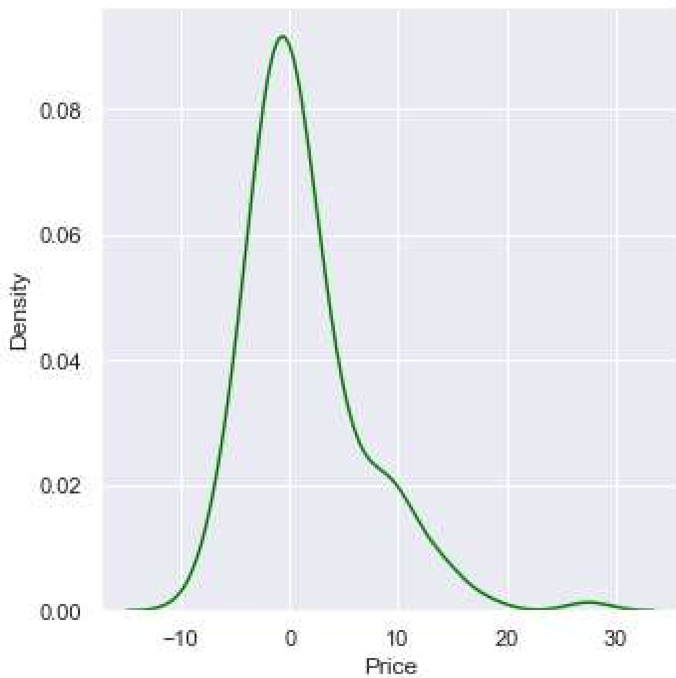
In [212]:

```python
residuals=y_test-predicted_values
```

In [216]:

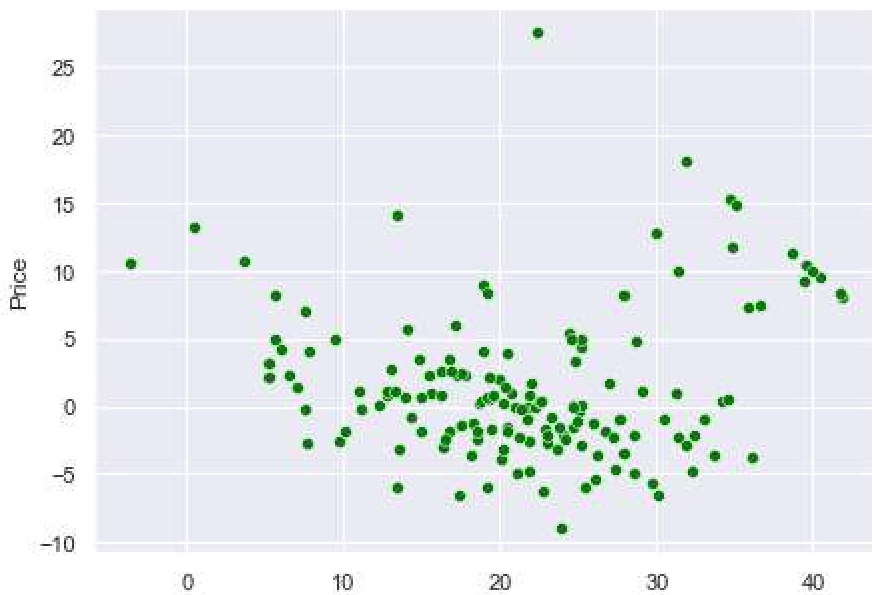
```
sns.displot(residuals,kind="kde",palette="Set1",color="green")
```

Out[216]: <seaborn.axisgrid.FacetGrid at 0x1b3c6af2ac0>



In [220]:
```
sns.scatterplot(x=predicted_values,y=residuals,color='green',)
```

Out[220]: <AxesSubplot:ylabel='Price'>



R square and adjusted R square

In [221]:
```
from sklearn.metrics import r2_score
R_score=r2_score(y_test,predicted_values)
print("R score value is",R_score)
```

R score value is 0.7215519718844172

In [223]:
```
Adjusted_r_score=1 - (1-R_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1
print("Adjusted R score value is",Adjusted_r_score)
```

Adjusted R score value is 0.6953213605401957

Ridge regression model

In [225]:
```
from sklearn.linear_model import Ridge
ridge=Ridge()
```

In [229]:
```
ridge_model=ridge.fit(X_train,y_train)
```

In [228]:
```
ridge_predict=ridge.predict(X_test)
```

In [230]:
```
ridge_model.coef_
```

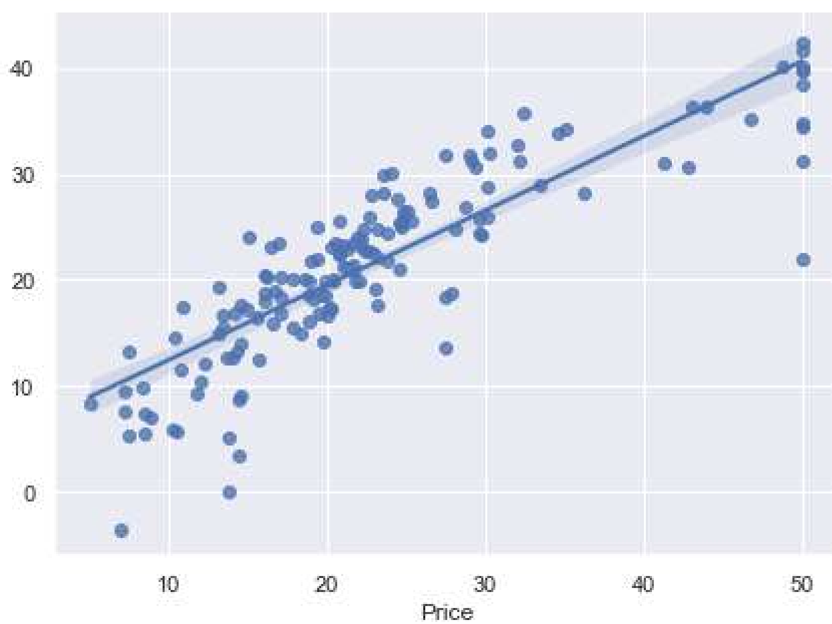
Out[230]:
```
array([-8.96773369e-02,  3.87293612e-02, -9.46326932e-02,  3.27853539e+00,
       -8.83310616e+00,  3.70498171e+00, -1.32155805e-02, -1.23740801e+00,
        2.25516417e-01, -1.00845466e-02, -7.84860767e-01,  7.35041802e-03,
       -5.07459245e-01])
```

In [231]:
```
ridge_model.intercept_
```

Out[231]:
```
30.29949040952099
```

In [232]:
```
sns.regplot(x=y_test,y=ridge_predict)
```

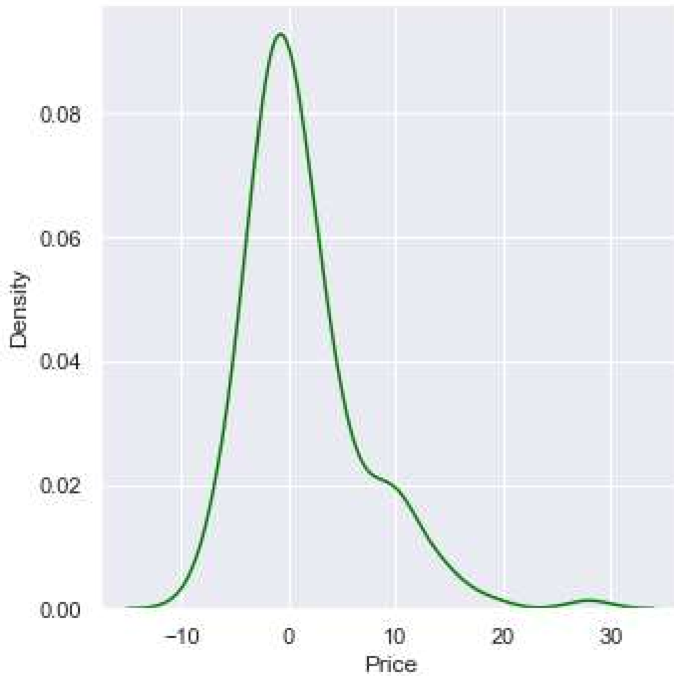
Out[232]:
```
<AxesSubplot:xlabel='Price'>
```



In [233]:

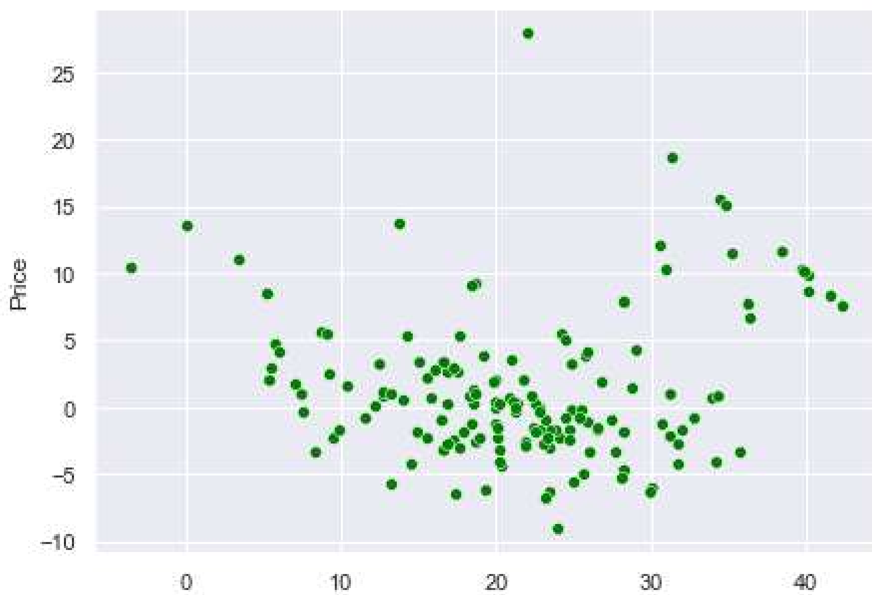
```
ridge_resuidals=y_test-ridge_predict
sns.displot(ridge_resuidals,kind="kde",palette="Set1",color="green")
```

Out[233]: <seaborn.axisgrid.FacetGrid at 0x1b3c7eed6d0>



In [236]:
```
sns.scatterplot(x=ridge_predict,y=ridge_resuidals,color='green',)
```

Out[236]: <AxesSubplot:ylabel='Price'>



Performance Metrics

In [237]:
```
print("Mean squared error:",mean_squared_error(y_test,ridge_predict))
print("Mean absolute error:",mean_absolute_error(y_test,ridge_predict))
print("Root mean squared error:",np.sqrt(mean_squared_error(y_test,ridge_pre
```

```
Mean squared error: 32.0345368225147
Mean absolute error: 3.9014189572909572
Root mean squared error: 5.659906078948192
```

Lasso regression model

In [238]:
```
from sklearn.linear_model import Lasso
```

In [239]:
```
lasso_model=Lasso()
```

In [242]:
```
lasso_model=lasso_model.fit(x_train,y_train)
```

In [243]:
```
lasso_predict=lasso_model.predict(x_test)
```

In [244]:
```
lasso_predict
```

Out[244]:
```
array([21.34943756, 30.45318044,  9.76791574, 20.28915417, 17.71113013,
       27.45637077, 26.66319632, 17.10112721, 22.57985801,  1.7296673 ,
       20.58479405, 17.87865192,  4.34340966,  6.70983829, 22.71114893,
       34.28455434, 23.055947  , 19.39217948, 24.49889928, 15.66732815,
       35.42870375, 13.60752019, 32.90886105, 24.98379635, 36.14242032,
       29.03925819, 21.93532581, 21.71817967, 20.24747986, 20.50831643,
       24.86664646, 24.07003804, 28.8757406 , 25.6785744 , -0.23400937,
       24.88197726, 25.55953235, 11.9363952 , 36.60086295, 26.74081754,
       25.1929427 , 10.52873993, 19.15795595, 21.47613761, 30.72956898,
        7.79566921, 13.75848843, 29.15598704, 20.78588524, 32.55445146,
       14.25879118, 20.78376397, 17.16959343, 13.37098929, 15.63037829,
       37.22159371, 25.0004319 , 18.0836259 , 22.62264165, 10.20743311,
       23.25511343, 13.11813923, 29.77533454, 13.91016337, 25.49115206,
       15.98976512,  4.57956761, 29.78260633, 18.52792707, 19.12915633,
       21.2179911 , 25.61629936, 21.83311757, 29.12291699, 32.11381314,
       19.40982872, 16.12273523, 25.9968222 , 18.11940996, 26.45392971,
       18.66219764, 13.49719305, 20.24261533, 13.4594494 , 18.5513846 ,
       10.93173044, 16.79905557, 22.29027312, 20.12925555, 18.65810201,
       23.43081403, 10.24298825, 20.55058669,  8.65794007, 19.43321998,
        6.13141912, 18.65069388, 26.13931311, 22.35657609, 19.83791901,
       24.14245176, 18.01044292, 24.11018372,  6.37749415, 29.36638489,
       24.39914647, 27.89469674, 31.00315572, 22.67316443, 17.63904305,
       34.00548455, 35.37768265, 26.62176649, 22.7781386 , 24.97210281,
       19.50897172, 21.62116718, 22.70604851, 35.89213029, 22.14704538,
       22.93953403, 26.57059325, 28.8755948 , 17.78095571, 38.17535806,
       17.83236195, 21.2435862 , 21.80465299, 17.91024311, 26.2268157 ,
       15.20023204, 25.82043688, 27.49505856, 22.09921939, 30.15073758,
       32.54023852, 19.47952826, 20.87293377, 13.42168405, 21.76504344,
       11.8022726 , 34.9518832 , 20.38607318, 25.20425326, 26.84212068,
       19.36409275, 13.79327497, 18.89432554, 17.60815573, 30.11257116,
       29.4974732 , 15.27235538])
```

In [245]:
```
lasso_model.coef_
```

Out[245]:
```
array([-0.        ,  0.        , -0.        ,  0.09154745, -0.        ,
        2.43878411, -0.        , -0.        , -0.        , -0.11045338,
       -1.07955341,  0.        , -3.33388011])
```

In [246]:
```
lasso_model.intercept_

```

Out[246]:
```
22.331355932203394
```

In [256]:
```
Lasso_model_coefficent=pd.DataFrame(data=df.feature_names,columns=["Independ
```

In [257]:
```
Lasso_model_coefficent[" lasso coefficent"]=lasso_model.coef_
```

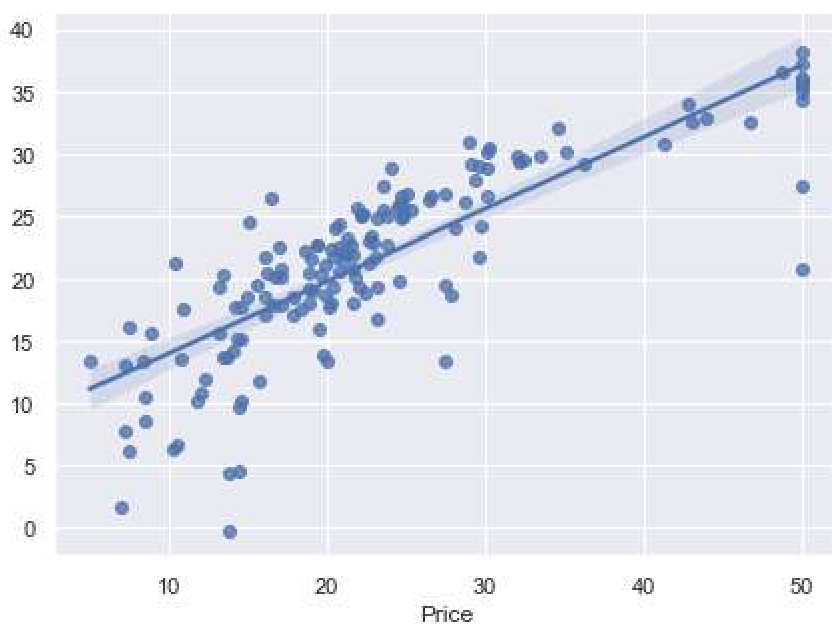
In [258]:
```
Lasso_model_coefficent
```

Out[258]:

	Independent factors	lasso coefficent
0	CRIM	-0.000000
1	ZN	0.000000
2	INDUS	-0.000000
3	CHAS	0.091547
4	NOX	-0.000000
5	RM	2.438784
6	AGE	-0.000000
7	DIS	-0.000000
8	RAD	-0.000000
9	TAX	-0.110453
10	PTRATIO	-1.079553
11	B	0.000000
12	LSTAT	-3.333880

In [259]:
```
sns.regplot(x=y_test,y=lasso_predict)
```

Out[259]:
```
<AxesSubplot:xlabel='Price'>
```



Performance Metrics

In [260]:

```
print("Mean squared error:",mean_squared_error(y_test,lasso_predict))
print("Mean absolute error:",mean_absolute_error(y_test,lasso_predict))
print("Root mean squared error:",np.sqrt(mean_squared_error(y_test,lasso_pre
```

```
Mean squared error: 40.3546502317852
Mean absolute error: 4.28976184314201
Root mean squared error: 6.35253100990347
```

Elastic Net regression model

In [267]:

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

In [269]:

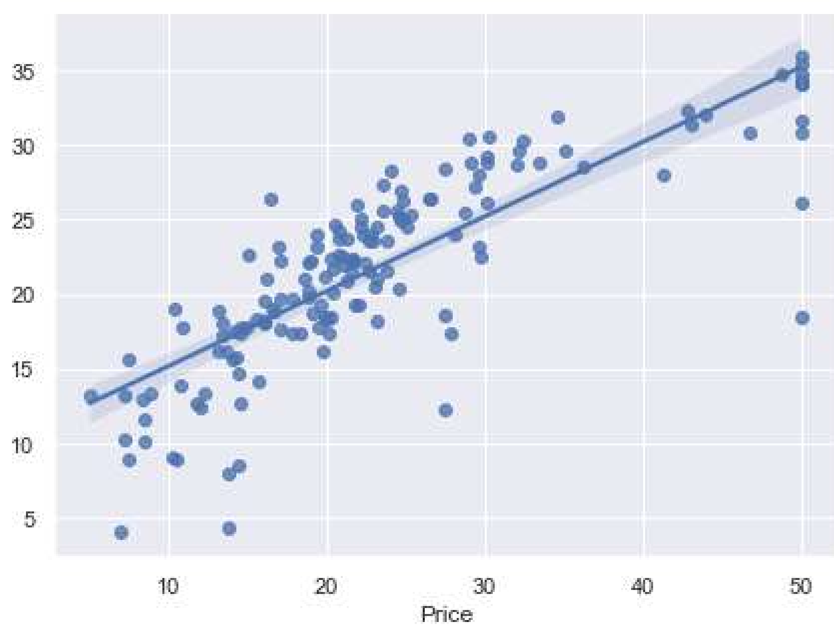
```
en=ElasticNet()
Elasticnet_model=en.fit(x_train,y_train)
Elasticnet_predict=en.predict(x_test)
```

In [272]:

```
sns.regplot(x=y_test,y=Elasticnet_predict)
```

Out[272]:

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



Performance Metrics

In [273]:

```
print("Mean squared error:",mean_squared_error(y_test,Elasticnet_predict))
print("Mean absolute error:",mean_absolute_error(y_test,Elasticnet_predict))
print("Root mean squared error:",np.sqrt(mean_squared_error(y_test,Elasticne
```

```
Mean squared error: 43.90854807474513
Mean absolute error: 4.295235872781503
Root mean squared error: 6.626352546819791
```

Comparsion of coeffiecent between linear,Ridge,Lasso resgression

In [264]:
```
Comparsion_of_coeffiecent=pd.DataFrame(data=df.feature_names,columns=["Indep
```

In [270]:
```
Comparsion_of_coeffiecent["Linear regreesion coefficient"]=Linear_regression
Comparsion_of_coeffiecent["Ridge regression coefficient"]=ridge_model.coef_
Comparsion_of_coeffiecent["Lasso regression coefficient"]=lasso_model.coef_
Comparsion_of_coeffiecent["ElasticNet regression coefficient"]=Elasticnet_mo
```

In [271]:
```
Comparsion_of_coeffiecent
```

Out[271]:

	Independent factors	Linear regreesion coefficient	Ridge regression coefficient	Lasso regression coefficient	ElasticNet regression coefficient
0	CRIM	-0.620852	-0.089677	-0.000000	-0.117033
1	ZN	0.896045	0.038729	0.000000	0.062132
2	INDUS	-0.418102	-0.094633	-0.000000	-0.467268
3	CHAS	0.857945	3.278535	0.091547	0.522033
4	NOX	-1.983452	-8.833106	-0.000000	-0.320978
5	RM	2.340541	3.704982	2.438784	2.105497
6	AGE	-0.147083	-0.013216	-0.000000	-0.017915
7	DIS	-2.864497	-1.237408	-0.000000	-0.000000
8	RAD	2.154137	0.225516	-0.000000	-0.000000
9	TAX	-1.584108	-0.010085	-0.110453	-0.369758
10	PTRATIO	-1.744400	-0.784861	-1.079553	-1.059870
11	B	0.630548	0.007350	0.000000	0.365535
12	LSTAT	-3.220109	-0.507459	-3.333880	-2.083438

From this table we can conclude that how overfitting and underfitting can be reduce with ridge and lasso regreesion. In ridge regression in cost function we take lamba and square of slope in plus of linear regreesion but In Lasso regreesion in cost function we take lamba and absolute of slope in plus of linear regreesion In Elastic net regression in cost function we take addition of linear,ridge and lasso cost function

In [ ]:
```

```