In [69]:
```
from sklearn.datasets import load_boston
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,roc_curve


from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [11]:
```
data= load_boston()
```

In [12]:
```
data
```

Out[12]:

```
{'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n--------------
-------------\n\n**Data Set Characteristics:**  \n\n    :Number of Instances:
506 \n\n    :Number of Attributes: 13 numeric/categorical predictive. Median
Value (attribute 14) is usually the target.\n\n    :Attribute Information (in
order):\n        - CRIM     per capita crime rate by town\n        - ZN
proportion of residential land zoned for lots over 25,000 sq.ft.\n        - I
NDUS    proportion of non-retail business acres per town\n        - CHAS
Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n
- NOX      nitric oxides concentration (parts per 10 million)\n        - RM
average number of rooms per dwelling\n        - AGE      proportion of owner-
occupied units built prior to 1940\n        - DIS      weighted distances to
five Boston employment centres\n        - RAD      index of accessibility to
radial highways\n        - TAX      full-value property-tax rate per $10,000
\n        - PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk
- 0.63)^2 where Bk is the proportion of blacks by town\n        - LSTAT    %
lower status of the population\n        - MEDV     Median value of owner-occu
pied homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator:
Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing datase
t.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/housing/\n\n\nT
his dataset was taken from the StatLib library which is maintained at Carnegi
e Mellon University.\n\nThe Boston house-price data of Harrison, D. and Rubin
feld, D.L. 'Hedonic\nprices and the demand for clean air', J. Environ. Econom
ics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Reg
ression diagnostics\n...', Wiley, 1980.   N.B. Various transformations are us
ed in the table on\npages 244-261 of the latter.\n\nThe Boston house-price da
ta has been used in many machine learning papers that address regression\npro
blems.   \n     \n.. topic:: References\n\n   - Belsley, Kuh & Welsch, 'Regre
ssion diagnostics: Identifying Influential Data and Sources of Collinearity',
Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining Instance-Based and M
odel-Based Learning. In Proceedings on the Tenth International Conference of
Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufm
ann.\n",
 'data': array([[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]]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'filename': '/usr/local/lib/python3.6/dist-packages/sklearn/datasets/data/bo
ston_house_prices.csv',
 'target': array([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 [13]:
```
data.DESCR
```

Out[13]:
```
".. _boston_dataset:\n\nBoston house prices dataset\n-----------------------
---\n\n**Data Set Characteristics:**  \n\n    :Number of Instances: 506 \n\n
:Number of Attributes: 13 numeric/categorical predictive. Median Value (attri
bute 14) is usually the target.\n\n    :Attribute Information (in order):\n
- CRIM     per capita crime rate by town\n        - ZN       proportion of re
sidential land zoned for lots over 25,000 sq.ft.\n        - INDUS    proporti
on of non-retail business acres per town\n        - CHAS     Charles River du
mmy variable (= 1 if tract bounds river; 0 otherwise)\n        - NOX      nit
ric oxides concentration (parts per 10 million)\n        - RM       average n
umber of rooms per dwelling\n        - AGE      proportion of owner-occupied 
units built prior to 1940\n        - DIS      weighted distances to five Bost
on employment centres\n        - RAD      index of accessibility to radial hi
ghways\n        - TAX      full-value property-tax rate per $10,000\n
- PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2 
where Bk is the proportion of blacks by town\n        - LSTAT    % lower stat
us of the population\n        - MEDV     Median value of owner-occupied homes
in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, 
D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttps://
archive.ics.uci.edu/ml/machine-learning-databases/housing/\n\n\nThis dataset 
was taken from the StatLib library which is maintained at Carnegie Mellon Uni
versity.\n\nThe Boston house-price data of Harrison, D. and Rubinfeld, D.L.
'Hedonic\nprices and the demand for clean air', J. Environ. Economics & Manag
ement,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression dia
gnostics\n...', Wiley, 1980.   N.B. Various transformations are used in the t
able on\npages 244-261 of the latter.\n\nThe Boston house-price data has been
used in many machine learning papers that address regression\nproblems.   \n
\n.. topic:: References\n\n   - Belsley, Kuh & Welsch, 'Regression diagnostic
s: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 24
4-261.\n   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Lear
ning. In Proceedings on the Tenth International Conference of Machine Learnin
g, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.\n"
```

In [14]:
```
df= pd.DataFrame(data.data, columns=data.feature_names)
```

In [15]:
```
df.columns
```

Out[15]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TA
X',
       'PTRATIO', 'B', 'LSTAT'],
      dtype='object')
```

In [16]:
```
df.head()
```

Out[16]:

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

In [17]:
```
df['Price']=data.target
```

In [18]:
```
df
```

Out[18]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
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 × 14 columns

In [19]:
```
df.head()
```

Out[19]:

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

In [20]:
```
df.columns
```

Out[20]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TA
X',
       'PTRATIO', 'B', 'LSTAT', 'Price'],
      dtype='object')
```

In [21]:
```
df.shape
```

Out[21]:
```
(506, 14)
```

In [22]:
```
df.isnull().sum()
```

Out[22]:
```
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 [23]:
```
df.describe()
```

Out[23]:

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

In [24]:
```
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 [25]:
```
#df.to_csv("boston_data")
```

In [26]:
```
df.head()
```

Out[26]:

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

In [28]:
```
df
```

Out[28]:

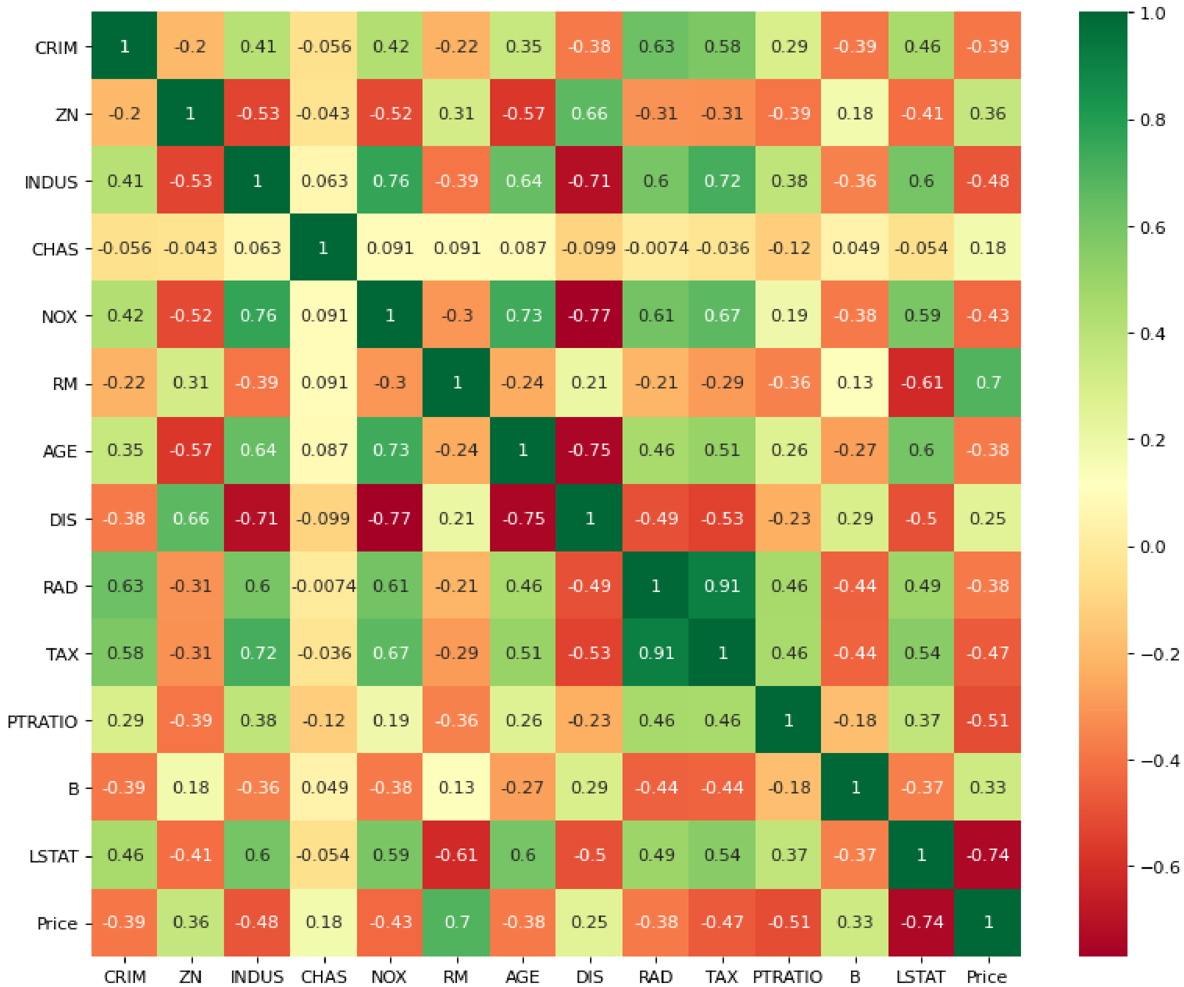
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	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 × 14 columns

In [32]:
```
#df.to_csv("boston_data")
```

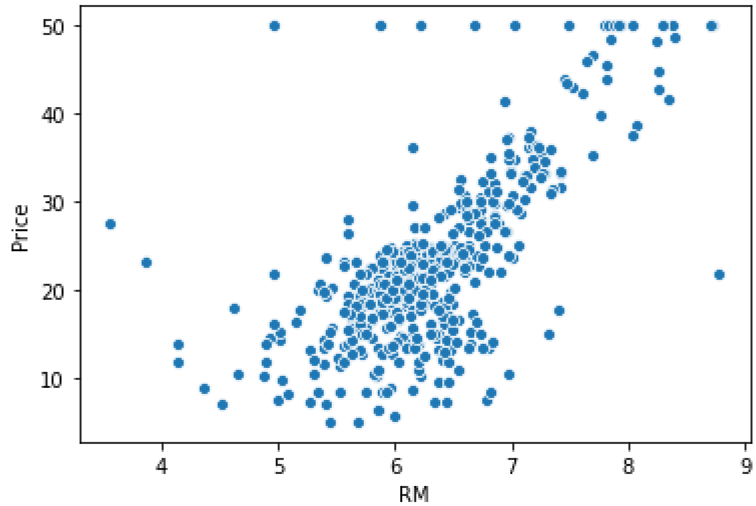
In [46]:
```
## data visualizations

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12,10),dpi = 80)
sns.heatmap(data=df.corr(),annot=True,cmap='RdYlGn')

## getting the correlation plot
```

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x7fee0d8ab8d0>



In [47]:
```
sns.scatterplot(data = df,x=df['RM'],y=df['Price'])  # scatter plot
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x7fee0d2e86a0>

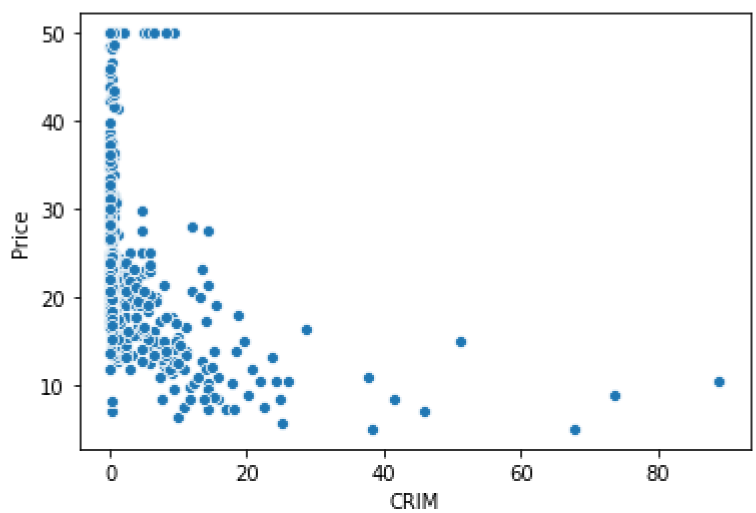


In [48]:
```
df.columns
```

Out[48]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TA
X',
       'PTRATIO', 'B', 'LSTAT', 'Price'],
      dtype='object')
```

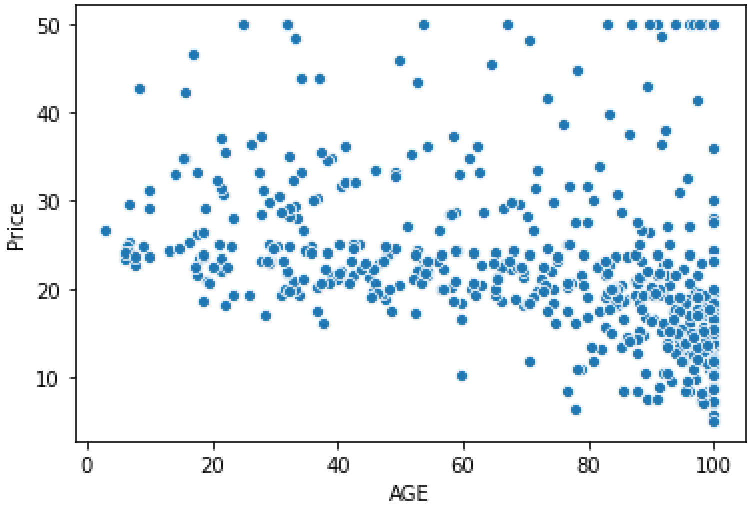
In [49]:
```
sns.scatterplot(data = df,x=df['CRIM'],y=df['Price'])  # scatter plot

## as we can see that the area where per capita crime rate is low , house pric
es are higher
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x7fee0d2c7080>



In [50]:
```
sns.scatterplot(data = df,x=df['AGE'],y=df['Price'])  # scatter plot

## we can see  a tren where the age of the house is old then the prices ar low too
```

Out[50]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fee0d226630>
```



In [77]:
```
X = df.RM
y = df.Price
```

In [78]:
```
X=X[:,np.newaxis]  ## np.newaxis to change the dimension of array , so the new array will be in 2d
```

In [79]:
```
y =y[:,np.newaxis]
```

In [80]:
```
theta = np.zeros([2,1])
iterations = 2000
lr = 0.01
ones = np.ones((len(y),1))
X = np.hstack((ones,X)) # 1X+c where slope =1 for now
```

In [81]:
```
X
```

Out[81]:
```
array([[1.   , 6.575],
       [1.   , 6.421],
       [1.   , 7.185],
       ...,
       [1.   , 6.976],
       [1.   , 6.794],
       [1.   , 6.03 ]])
```

In [82]:
```
def costFunctionCompute(X,y,theta):
  error = np.dot(X,theta)-y    ### predicted - actual
  return np.sum(np.power(error,2))/(2*m)

J = costFunctionCompute(X,y,theta)

print(J)
```

```
296.0734584980237
```

In [87]:
```
def gradientDescent(X,y,theta,lr,m,iterations):

  for i in range(iterations):
    temp=np.dot(X,theta)-y
    temp = np.dot(X.T,temp)

    theta = theta - (lr/m)* temp
  return theta
theta=gradientDescent(X,y,theta,lr,m,iterations)
theta
```

Out[87]:
```
array([[-6.97054334],
       [ 4.74751638]])
```

In [89]:
```
J = costFunctionCompute(X, y, theta)
J
```

Out[89]:
```
26.52709949493679
```

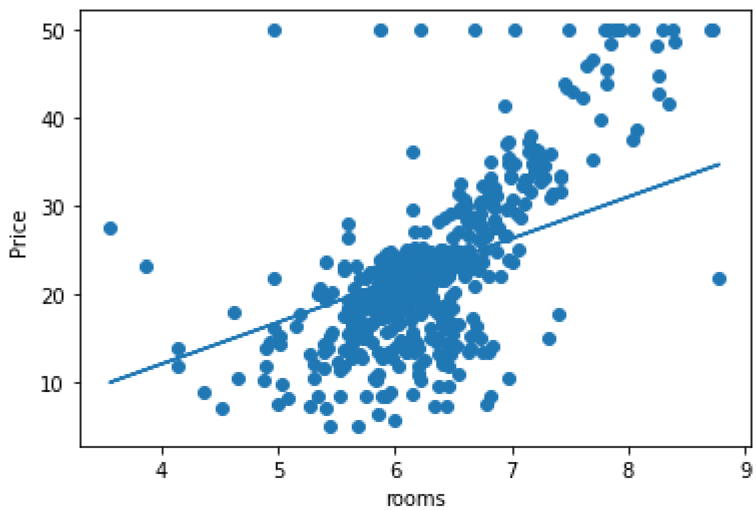
In [95]:
```
plt.scatter(X[:,1], y)
plt.xlabel('rooms')
plt.ylabel('Price ')
plt.plot(X[:,1], np.dot(X, theta))
plt.show()
```

Out[95]: <matplotlib.collections.PathCollection at 0x7fee0cdc17b8>

Out[95]: Text(0.5, 0, 'rooms')

Out[95]: Text(0, 0.5, 'Price ')

Out[95]: [<matplotlib.lines.Line2D at 0x7fee0cdc1a90>]



In [98]:
```
#plt.plot(X[:,1], np.dot(X, theta))
```

In [105]:
```
## another way to do this is using sklearn

reg =LinearRegression()
reg.fit(X,y)
```

Out[105]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)

In [106]:
```
reg.score
```

Out[106]: <bound method RegressorMixin.score of LinearRegression(copy_X=True, fit_inter
cept=True, n_jobs=None, normalize=False)>

In [107]:
```
reg.coef_
```

Out[107]: array([[0.   , 9.10210898]])

In [118]:
```
mean_absolute_error(y,reg.predict(X))
```

Out[118]: 4.447772901532234

In [ ]:
```
## thus this way we can use MLR to perform operations
```