LinearReg_Boston

December 18, 2020

[3]:
```python
'''
BOSTON DATASET - Linear Regression

@Author - Rahul Garg (rahu.garg3@hpe.com)
'''
```

[3]:
```
'\nBOSTON DATASET - Linear Regression\n\n@Author - Rahul Garg
(rahu.garg3@hpe.com)\n'
```

[66]:
```python
#Importing Required Modules
import os,sys
import scipy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import scipy.stats as stats
import sklearn
from sklearn.datasets import load_boston
import seaborn as sns
%matplotlib inline
```

[67]:
```python
boston=load_boston()
df_bos=pd.DataFrame(boston.data,columns=list(boston.feature_names))
```

[68]:
```python
df_bos.head()
```

[68]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
```

```
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

[69]:
```python
df_bos['Price']=list(boston.target)
```

[70]:
```python
df_bos.head()
```

[70]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  Price
0     15.3  396.90   4.98   24.0
1     17.8  396.90   9.14   21.6
2     17.8  392.83   4.03   34.7
3     18.7  394.63   2.94   33.4
4     18.7  396.90   5.33   36.2
```

[71]:
```python
#Boston Dataset 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 25,000
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 blacks 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/


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

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. '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 that
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, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

[72]:
```python
print(df_bos.shape)
```

```
(506, 14)
```

[73]:
```python
print(df_bos.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: 55.4 KB
None
```

No NULL Values are present in boston dataset.

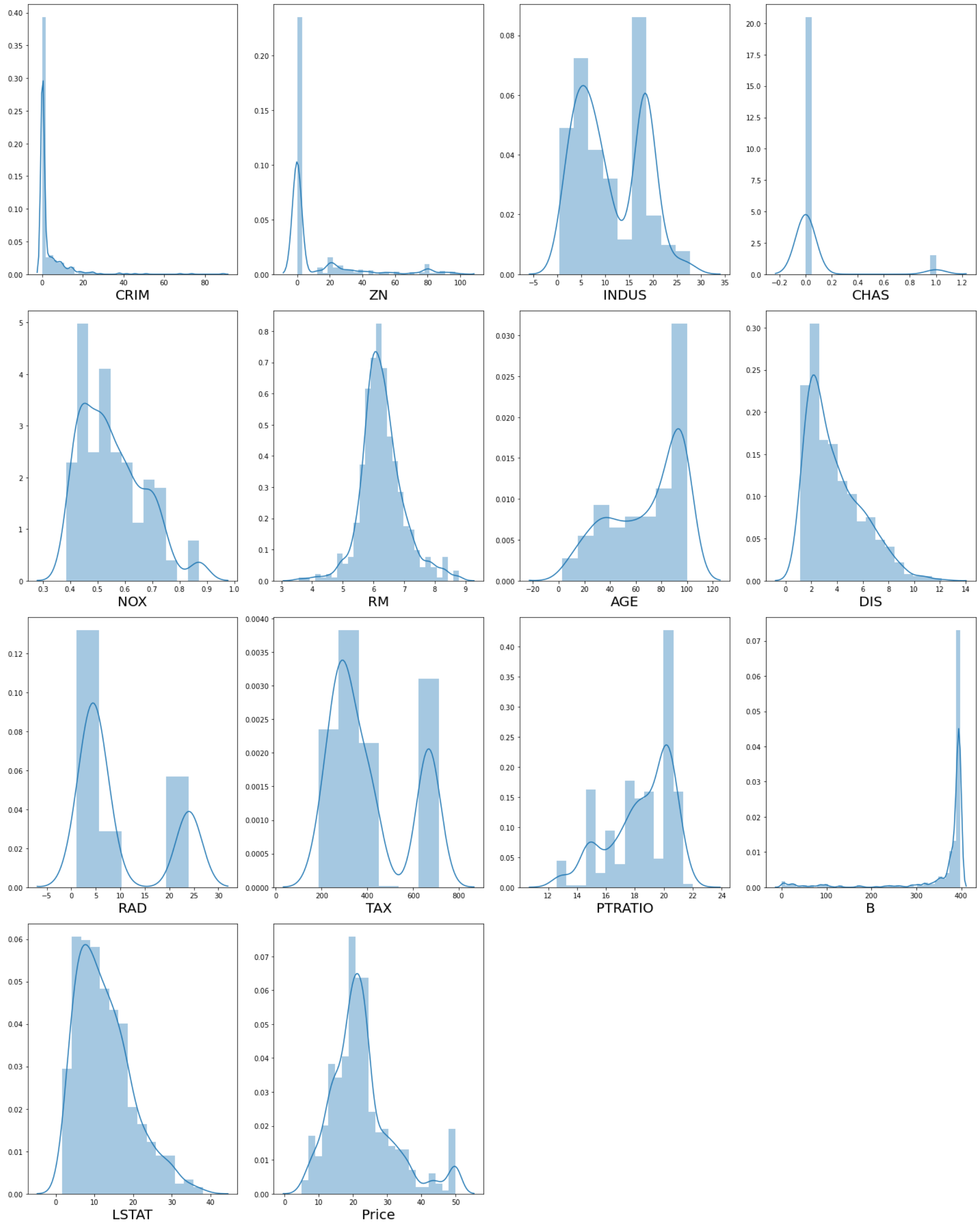
[74]:
```
df_bos.describe()
```

[74]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT       Price
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
```

```
min       1.730000    5.000000
25%       6.950000   17.025000
50%      11.360000   21.200000
75%      16.955000   25.000000
max      37.970000   50.000000
```

[75]:

```
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in df_bos:
    if plotnumber<=16 :
        ax = plt.subplot(4,4,plotnumber)
        sns.distplot(df_bos[column])
        plt.xlabel(column,fontsize=20)
        #plt.ylabel('Salary',fontsize=20)
    plotnumber+=1
plt.tight_layout()
```

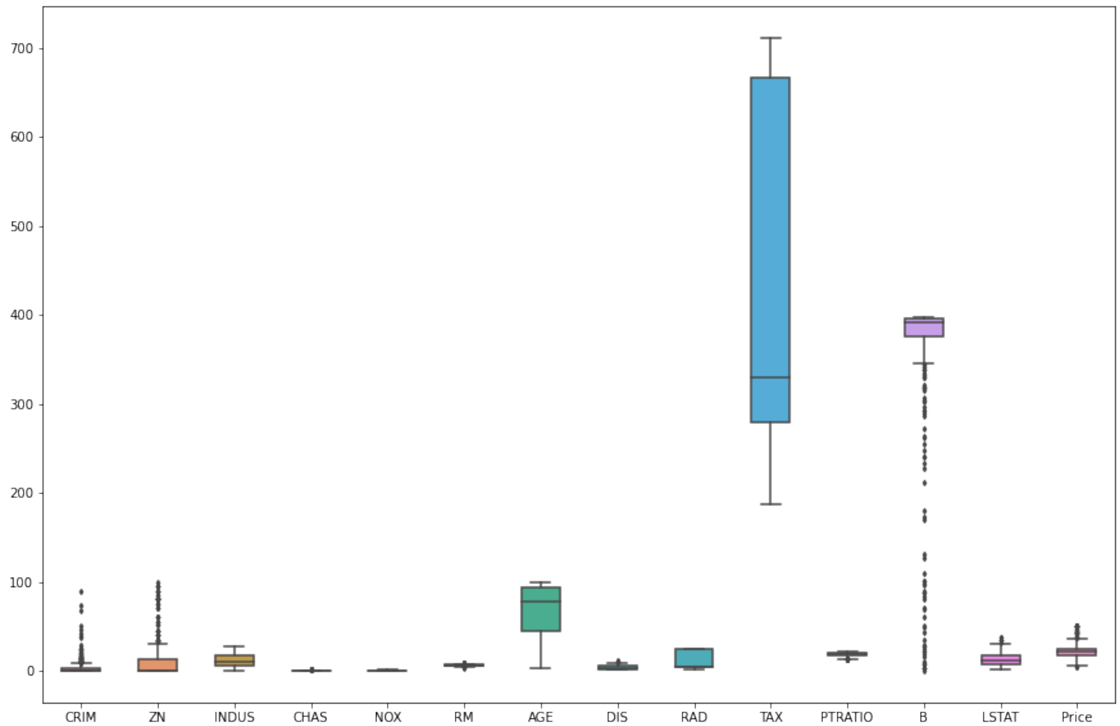


Skewness is observed in many features let us visualize if any outliers are present

[76]:
```
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=df_bos, width= 0.5,ax=ax,  fliersize=3)
'''plt.figure(figsize=(15,10), facecolor='white')
```

```
df_bos.boxplot()'''
```

[76]: "plt.figure(figsize=(15,10), facecolor='white')\ndf_bos.boxplot()"



Outliers are observed in the dataset as can be seen in boxplot

[77]:
```
df_bos.columns
```

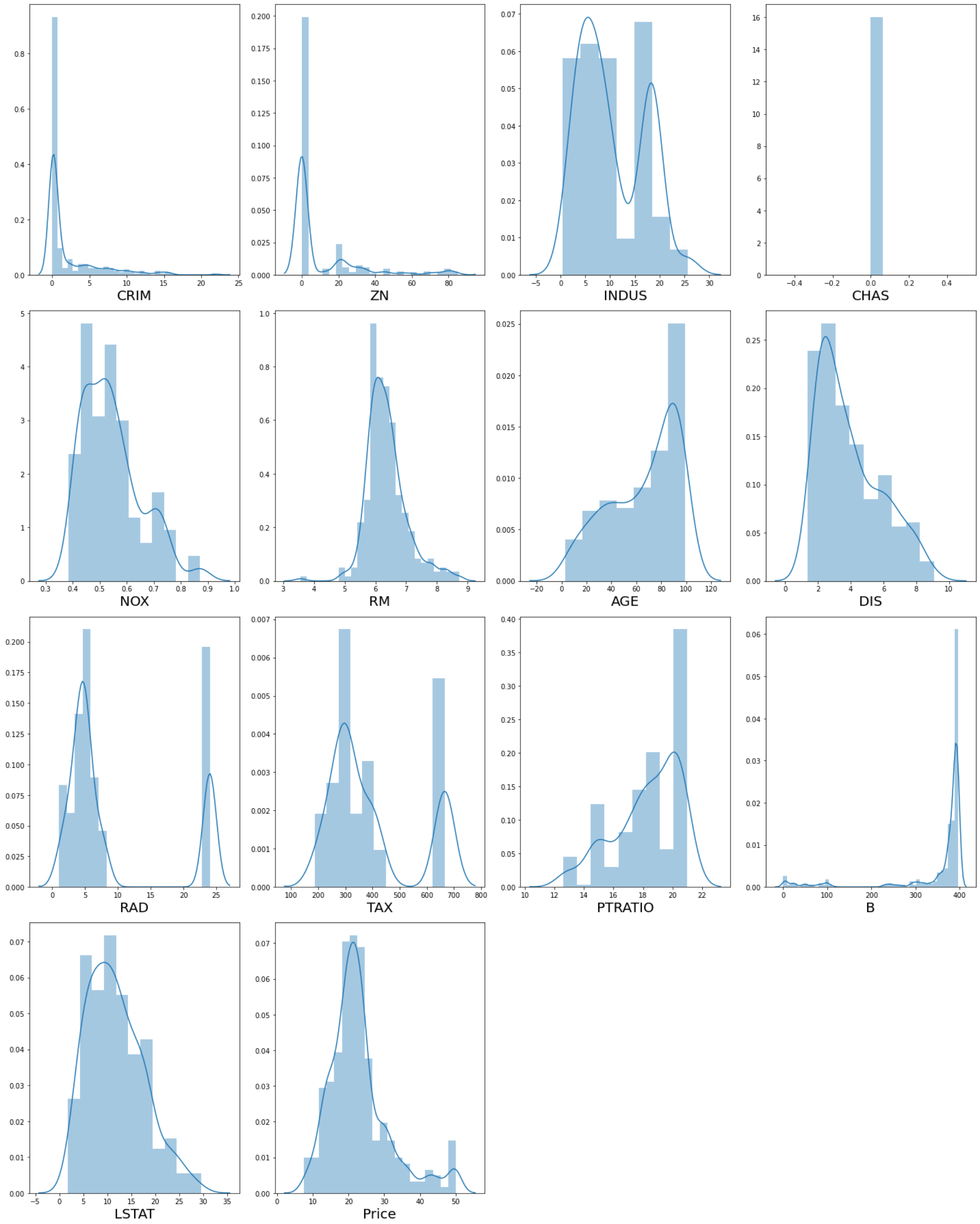
[77]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'Price'],
      dtype='object')
```

[78]:
```
#Handling Outliers
q = df_bos['CRIM'].quantile(0.98)
data_cleaned = df_bos[df_bos['CRIM']<q]

q = data_cleaned['ZN'].quantile(0.98)
data_cleaned  = data_cleaned[data_cleaned['ZN']<q]

q = data_cleaned['INDUS'].quantile(0.99)
data_cleaned  = data_cleaned[data_cleaned['INDUS']<q]

q = data_cleaned['CHAS'].quantile(0.99)
data_cleaned  = data_cleaned[data_cleaned['CHAS']<q]
```

```
q = data_cleaned['B'].quantile(0.95)
data_cleaned  = data_cleaned[data_cleaned['B']<q]

q = data_cleaned['LSTAT'].quantile(0.98)
data_cleaned  = data_cleaned[data_cleaned['LSTAT']<q]

q = data_cleaned['PTRATIO'].quantile(0.99)
data_cleaned  = data_cleaned[data_cleaned['PTRATIO']<q]

q = data_cleaned['DIS'].quantile(0.99)
data_cleaned  = data_cleaned[data_cleaned['DIS']<q]

q = data_cleaned['AGE'].quantile(0.99)
data_cleaned  = data_cleaned[data_cleaned['AGE']<q]
```

[79]:
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data_cleaned:
    if plotnumber<=16 :
        ax = plt.subplot(4,4,plotnumber)
        sns.distplot(data_cleaned[column])
        plt.xlabel(column,fontsize=20)
        #plt.ylabel('Salary',fontsize=20)
    plotnumber+=1
plt.tight_layout()
```

```
c:\users\garahul\appdata\local\programs\python\python38-32\lib\site-
packages\seaborn\distributions.py:283: UserWarning: Data must have variance to
compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
```



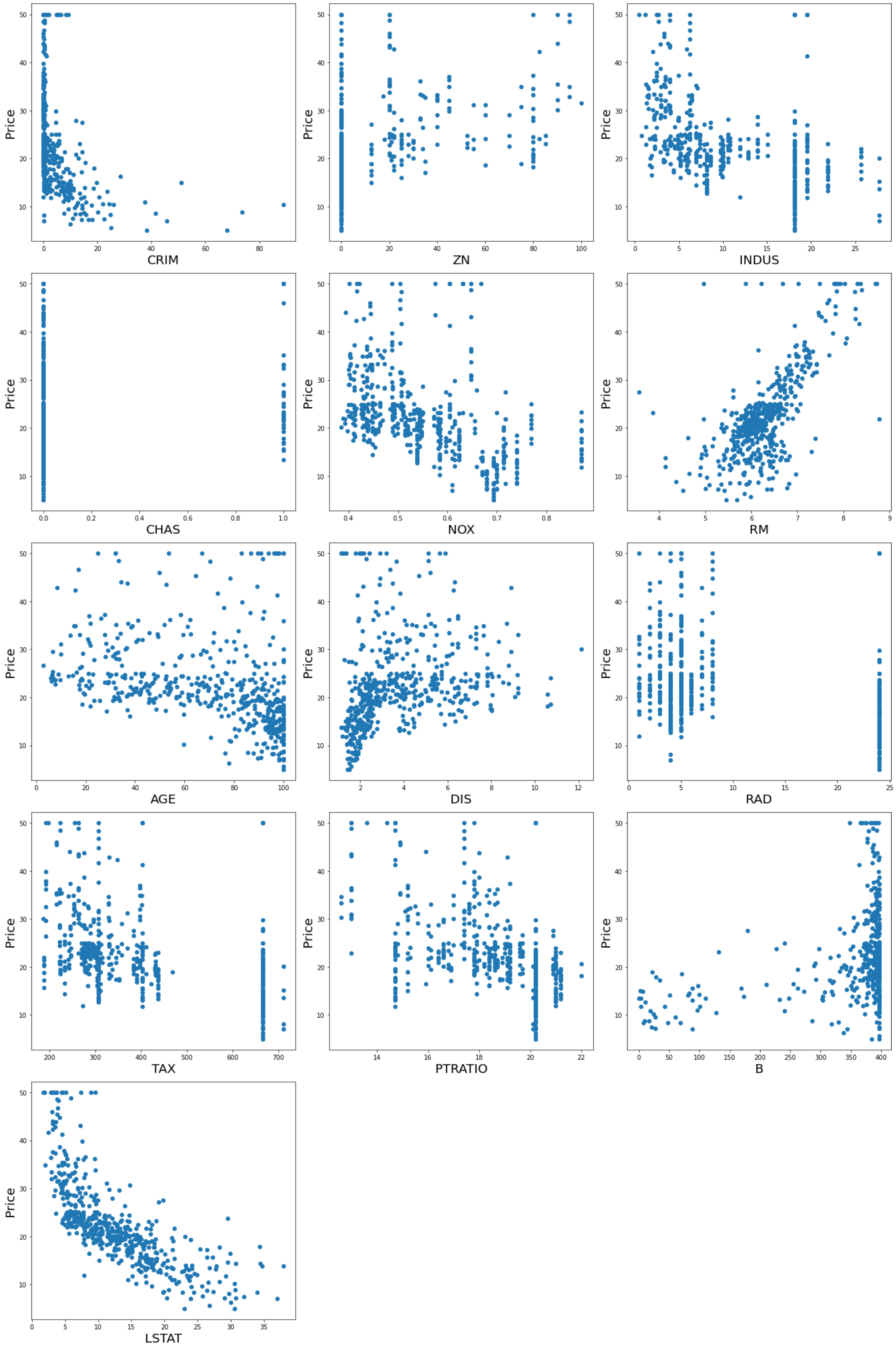
[80]:
```
data_cleaned.shape
```

[80]: (287, 14)

[85]:
```
y = df_bos['Price']
X =df_bos.drop(columns = ['Price'])

'''y=data_cleaned['Price']
X=data_cleaned.drop(columns=['Price'])'''
```

[85]: "y=data_cleaned['Price']\nX=data_cleaned.drop(columns=['Price'])"

[86]:
```
plt.figure(figsize=(20,30), facecolor='white')
plotnumber = 1

for column in X:
    if plotnumber<=15 :
        ax = plt.subplot(5,3,plotnumber)
        plt.scatter(X[column],y)
        plt.xlabel(column,fontsize=20)
        plt.ylabel('Price',fontsize=20)
    plotnumber+=1
plt.tight_layout()
```


Price
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT

[88]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import Ridge,Lasso,RidgeCV, LassoCV, ElasticNet,␣
↪ElasticNetCV, LinearRegression
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
```

[89]:
```
#Scaling the features
scaler =StandardScaler()

X_scaled = scaler.fit_transform(X)
```

[90]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
variables = X_scaled
vif = pd.DataFrame()

# here we make use of the variance_inflation_factor, which will basically␣
↪output the respective VIFs
vif["VIF"] = [variance_inflation_factor(variables, i) for i in range(variables.
↪shape[1])]
# Finally, I like to include names so it is easier to explore the result
vif["Features"] = X.columns
```

[92]:
```
vif.sort_values(by='VIF')
```

[92]:

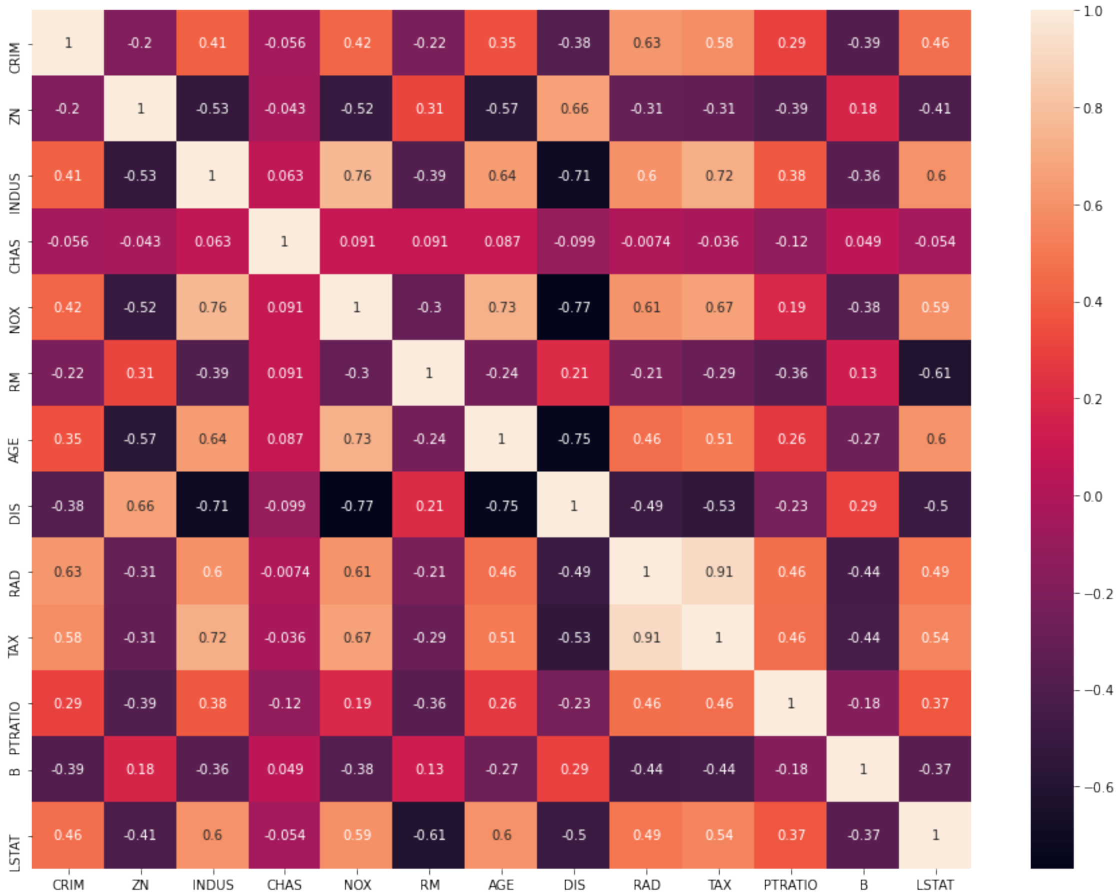
	VIF	Features
3	1.073995	CHAS
11	1.348521	B
0	1.792192	CRIM
10	1.799084	PTRATIO
5	1.933744	RM
1	2.298758	ZN
12	2.941491	LSTAT
6	3.100826	AGE
7	3.955945	DIS
2	3.991596	INDUS
4	4.393720	NOX
8	7.484496	RAD
9	9.008554	TAX

As 5 is the VIF threshold value for features hence we see RAD and TAX columns have high multicollinearity.

[95]:
```
# Lets look at the correlation matrix of our data.
fig = plt.figure(figsize=(16,12))
ax = fig.add_subplot(111)
```

```python
sns.heatmap(X.corr(),annot=True)
```

[95]: <AxesSubplot:>



Our target variable, seems to be highly correlated, with LSTAT and RM, which makes sense, as these two are very important factors for house pricing, but there seems to be a lot of multicollinearity as well.

The issue here is, that there is a lot of collinearity between our predictor variables, for example DIS is highly correlated to INUDS, INOX and AGE.

This is not good, as multicollinearity can make our model unstable, we need to look at it a little more, before modeling our data, I have explained, the probem of multicollinearity below.

[96]:
```python
#Applying PCA to remove the collinearily between the data
```

[97]:
```python
from sklearn.decomposition import PCA
```

[99]:
```python
pca=PCA(n_components=13)
```

[100]:
```python
X_pca=pca.fit_transform(X)
```

[101]:
```python
X_pca
```

[101]:
```
array([[-1.19818843e+02, -5.56005586e+00, -3.17269264e+00, …,
         3.78374287e-01, -7.13108946e-02,  3.35451015e-02],
       [-1.68890155e+02,  1.01162086e+01, -3.07818868e+01, …,
         4.86740794e-02, -9.47343278e-02, -3.31502751e-02],
       [-1.69311707e+02,  1.40805323e+01, -1.67536282e+01, …,
        -4.67469154e-01, -1.07257460e-01, -4.50902543e-03],
       …,
       [-1.38387163e+02,  9.38092201e-01, -3.72851813e+01, …,
        -2.97952531e-01, -1.04654969e-01,  4.30883930e-02],
       [-1.37505173e+02,  4.25182510e+00, -3.59883419e+01, …,
        -1.94149871e-01, -9.54593524e-02,  4.51072934e-02],
       [-1.39190333e+02,  1.00906423e+00, -2.97724323e+01, …,
         4.16189839e-01, -7.03283698e-02,  4.55682936e-02]])
```

Checking Multicollinearity after PCA using VIF and HeatMap

[103]:
```python
variables = X_pca
vif = pd.DataFrame()

# here we make use of the variance_inflation_factor, which will basically␣
↪output the respective VIFs
vif["VIF"] = [variance_inflation_factor(variables, i) for i in range(variables.
↪shape[1])]
# Finally, I like to include names so it is easier to explore the result
vif["Features"] = X.columns
```

[104]:
```python
vif.sort_values(by='VIF')
```

[104]:

	VIF	Features
10	1.0	PTRATIO
11	1.0	B
0	1.0	CRIM
1	1.0	ZN
2	1.0	INDUS
5	1.0	RM
6	1.0	AGE
7	1.0	DIS
8	1.0	RAD
9	1.0	TAX
12	1.0	LSTAT
3	1.0	CHAS
4	1.0	NOX

```
[106]: df_pca=pd.DataFrame(X_pca,columns=list(X.columns))
```

```
[112]: fig = plt.figure(figsize=(16,12))
       ax = fig.add_subplot(111)
       sns.heatmap(df_pca.corr(),annot=True)
```

[112]: <AxesSubplot:>


```
[113]: df_pca
```

[113]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	\
0	-119.818843	-5.560056	-3.172693	5.291593	-1.818728	-6.312070	
1	-168.890155	10.116209	-30.781887	1.296776	0.369680	-3.241821	
2	-169.311707	14.080532	-16.753628	-10.278399	-0.093409	-5.910068	
3	-190.230642	18.302463	-6.534195	-19.644921	1.513442	-6.959925	
4	-190.133451	16.097947	-13.158520	-14.178141	1.761005	-5.760987	
..	…	…	…	…	…	…	
501	-138.697933	5.781485	-20.978012	-5.706647	-0.480929	0.145623	
502	-139.504439	1.039389	-26.794150	-0.878985	-0.897763	-1.145200	

```
503 -138.387163   0.938092 -37.285181   8.073690 -2.368902 -5.829921
504 -137.505173   4.251825 -35.988342   7.016434 -2.102859 -4.911739
505 -139.190333   1.009064 -29.772432   1.676860 -1.369458 -2.651498

          AGE       DIS       RAD       TAX   PTRATIO         B     LSTAT
0   -1.032609  5.477971 -1.935498 -0.329154  0.378374 -0.071311  0.033545
1   -0.628651  0.915626  0.467686  1.299810  0.048674 -0.094734 -0.033150
2    1.718753  0.510026  0.414966  0.910646 -0.467469 -0.107257 -0.004509
3   -1.971382  0.845947  1.063487  0.964424 -0.276214 -0.052466  0.043716
4   -3.059650  1.032843  1.062885  1.123178 -0.538077 -0.055400  0.035538
..        …        …        …        …        …        …        …
501  3.776814  0.333697  2.909919 -1.690514 -0.322279 -0.063431  0.059858
502  3.795357  0.434186  3.104086 -1.639706  0.238205 -0.065791  0.045959
503  4.696785  0.604877  3.628018 -1.349502 -0.297953 -0.104655  0.043088
504  4.428370  0.624099  3.590587 -1.188895 -0.194150 -0.095459  0.045107
505  4.106156  0.500430  3.345011 -1.291621  0.416190 -0.070328  0.045568

[506 rows x 13 columns]
```

Linear Regression Application

[114]:
```
x_train,x_test,y_train,y_test=train_test_split(X_pca,y,test_size=0.
↪25,random_state=355)
```

[116]:
```
x_train.shape
```

[116]:
```
(379, 13)
```

[117]:
```
x_test.shape
```

[117]:
```
(127, 13)
```

[118]:
```
regression=LinearRegression()
regression.fit(x_train,y_train)
```

[118]:
```
LinearRegression()
```

[121]:
```
print('Coeff Values: ',regression.coef_)
print('Intercept Value: ',regression.intercept_)
```

```
Coeff Values:  [-2.52051940e-02 -7.68651599e-03  7.19116339e-02  4.41728011e-02
 -2.20021733e-01 -8.13427545e-01  1.86247037e-01 -3.20614939e-01
 -1.21547311e+00 -1.30135645e+00 -4.49148013e+00  3.52503055e+00
 -1.71830561e+01]
Intercept Value:  22.50722857528611
```

[128]:
```
from sklearn.metrics import mean_squared_error,r2_score
```

[126]:
```
#Training model on training data and predicting on training data
y_pred=regression.predict(x_train)
print(y_train.to_list())
print(list(y_pred))
```

```
[31.6, 11.3, 14.5, 24.5, 41.3, 16.3, 7.2, 50.0, 21.2, 22.9, 22.0, 19.5, 13.0,
48.8, 27.5, 21.7, 17.8, 19.1, 18.5, 8.5, 19.1, 13.8, 15.7, 9.5, 19.5, 27.5,
17.5, 14.0, 34.9, 26.4, 33.0, 14.8, 16.6, 21.7, 6.3, 22.6, 24.8, 19.4, 14.2,
13.8, 35.1, 11.8, 30.1, 42.8, 17.5, 22.7, 10.5, 17.8, 21.9, 25.0, 19.3, 17.8,
23.1, 23.0, 50.0, 21.4, 5.0, 29.0, 19.0, 23.1, 19.6, 50.0, 21.2, 43.8, 17.9,
22.4, 21.0, 18.4, 36.4, 25.0, 30.1, 25.0, 12.7, 13.6, 10.5, 41.7, 20.2, 20.8,
14.1, 12.5, 29.4, 50.0, 23.2, 32.5, 21.6, 11.7, 10.8, 50.0, 21.8, 19.7, 16.5,
25.3, 18.3, 17.2, 20.7, 28.6, 22.0, 16.5, 16.1, 24.8, 8.5, 32.4, 19.0, 23.7,
8.1, 15.6, 37.6, 43.5, 20.4, 21.5, 18.8, 16.8, 18.7, 23.4, 23.9, 20.1, 32.7,
15.1, 18.2, 21.7, 24.7, 24.3, 22.6, 50.0, 14.6, 21.0, 21.1, 22.5, 35.4, 21.5,
23.9, 17.6, 11.0, 8.7, 22.5, 20.9, 22.1, 16.2, 27.5, 22.0, 22.0, 13.2, 18.9,
17.8, 14.6, 7.5, 25.0, 18.6, 17.4, 13.4, 24.4, 8.8, 19.6, 20.0, 34.9, 33.4,
29.8, 33.3, 13.3, 28.1, 46.7, 24.4, 23.2, 20.4, 31.2, 24.3, 23.1, 22.0, 13.1,
24.8, 23.3, 14.4, 20.3, 32.0, 36.5, 23.9, 21.6, 19.7, 19.1, 36.2, 30.8, 19.9,
20.0, 23.3, 48.3, 29.9, 50.0, 37.0, 10.9, 20.6, 27.9, 18.7, 50.0, 17.4, 13.5,
17.7, 13.4, 17.1, 31.5, 15.6, 22.3, 32.9, 22.9, 22.2, 13.1, 32.2, 21.7, 45.4,
10.4, 12.0, 17.5, 27.9, 28.4, 15.6, 22.2, 19.8, 18.9, 29.1, 19.3, 12.7, 43.1,
20.3, 24.5, 19.3, 31.6, 23.1, 19.9, 10.9, 13.3, 31.0, 14.3, 16.1, 23.8, 12.3,
11.7, 28.5, 50.0, 18.3, 33.2, 19.4, 28.7, 29.0, 20.6, 19.8, 20.6, 10.2, 24.4,
19.2, 8.4, 22.3, 20.0, 14.9, 12.8, 21.4, 23.9, 22.8, 22.7, 11.9, 22.8, 26.2,
18.5, 39.8, 19.4, 21.2, 26.6, 29.8, 31.7, 17.1, 23.1, 17.2, 20.4, 21.7, 29.6,
7.0, 22.6, 23.1, 22.8, 18.9, 13.8, 15.6, 15.4, 16.1, 14.3, 48.5, 23.0, 20.8,
13.9, 23.0, 20.2, 25.1, 44.0, 18.4, 23.7, 11.5, 23.8, 24.1, 13.4, 22.8, 20.3,
24.0, 28.2, 18.5, 24.2, 10.2, 23.2, 19.4, 12.7, 50.0, 21.4, 16.2, 30.7, 17.0,
23.6, 28.0, 32.0, 21.1, 26.6, 7.0, 18.1, 23.7, 25.0, 30.1, 33.8, 31.5, 23.3,
15.0, 23.3, 18.2, 11.8, 18.2, 20.1, 23.7, 20.1, 20.6, 25.2, 26.5, 19.6, 33.2,
42.3, 21.2, 19.4, 13.8, 19.5, 21.9, 23.1, 24.8, 23.0, 37.9, 20.8, 21.7, 20.0,
24.1, 11.9, 21.2, 8.3, 50.0, 24.6, 20.1, 15.3, 10.4, 19.5, 22.6, 12.6, 8.4,
22.2, 12.1, 19.3, 22.0, 14.5, 14.5, 28.4, 7.2, 20.9, 20.5, 37.3, 22.4, 17.2,
19.2, 21.9]
[31.851986906332414, 13.52218438682431, 14.171577508537801, 27.67874242506792,
31.732624762009237, 11.338394755990214, 11.01308414014056, 43.627903049793424,
23.208976860566548, 24.821987150963945, 27.03806415547124, 17.647608222650526,
17.38653476420344, 40.8885141486155, 20.34243986286366, 21.099616279594315,
20.577489750837312, 17.03121357912451, 18.974079168284007, 16.128843927231074,
17.010953210186493, 7.161716238700679, 14.986919027343522, 13.856685915372266,
19.41049602317216, 24.64845034220419, 17.031827757793863, 14.60695584292602,
34.50685291019357, 22.183852068740155, 23.437563900579296, 15.087412312805762,
18.170119659159926, 21.187764269742434, 12.279901982807871, 22.771419093222033,
27.211990315589297, 25.617337023351567, 17.654534077410972, 19.815516049943618,
34.91791287601735, 9.039592893957451, 28.98881614719383, 30.856618113443908,
18.385653344333708, 22.523054063787853, 12.984796986311554, 22.901225453488344,
```

14.276474226576818, 28.602390320876886, 20.385455929071526, 16.728351980529744,
24.477623823981588, 19.58589790363279, 34.24521647881184, 21.2806859569455,
7.5614461064205045, 32.35846628644586, 21.136602035842817, 8.744887040301194,
17.575004566370296, 41.169471912628524, 21.260483170916604, 35.6118609135218,
2.6303626691805704, 24.008512473470656, 21.30739486955133, 18.416180384654172,
32.90627628461334, 27.370572931393397, 25.021003698989336, 27.787549342404244,
12.173327337922958, 13.93937959826285, 6.309538390143395, 37.83340410666332,
16.326304964675984, 17.722724492310768, 16.136062447438324, 19.459391788923472,
30.470634604755613, 25.798959751456238, 22.227934485135822, 29.98491638881071,
24.43217438093682, 14.440018388881901, 11.955184694525897, 42.16225189880892,
20.40203415600369, 14.541952308034112, 22.047139423171206, 25.041764696234242,
20.345764140256925, 17.541897689847072, 21.326034765871164, 27.873220320231077,
27.539227583911345, 12.324862226943551, 18.47659864827901, 24.895257195851435,
8.30145178050271, 36.92735321296846, 14.199744165324098, 27.262510936572905,
4.707333543284278, 15.761305395212446, 37.09477041050837, 38.8224055302274,
20.95263903533032, 20.100934218603957, 19.927872845521403, 20.68613018814371,
21.06071245603746, 24.184733402038383, 26.65360366484627, 21.369467423045933,
30.369248094041378, 16.667569538528824, 18.158520676222516, 23.776194340378353,
24.90680918282034, 23.988399311298295, 26.93876294835877, 36.115509734647105,
8.894809842916143, 22.834273481714572, 20.44573605779178, 17.968799079075684,
31.643389855743845, 25.588750243672745, 27.745960716643296, 15.705121766535495,
14.890412423682866, 9.792308644973835, 22.40152029509352, 20.66116620871812,
26.21056775805305, 15.04890392314515, 32.94447238215447, 25.688396921063703,
21.07348893257945, 10.508698969777434, 23.330571464480972, 10.151005922358767,
19.02381231766024, 14.726836724003508, 24.311976040211142, 19.82742144955305,
15.901815832101068, 16.834602626938537, 24.661644340683637, 3.7079486029083313,
16.99437560411844, 22.111202593257048, 30.793738394309937, 35.57864217913767,
25.519350308977803, 36.136240542569396, 20.83514165241623, 25.279306550811498,
35.85518808823609, 28.25212468070752, 26.709089185290583, 19.1586392435874,
28.78093781128333, 27.150731028058637, 25.534120012077672, 28.93808951445237,
12.738167788407136, 30.609591674177018, 28.205993663905318, 5.443269363507937,
22.09032100594666, 33.50827433658869, 35.270673872014704, 24.807523048563255,
24.727971451787962, 21.266344921242204, 22.980812481544685, 27.858255124457017,
31.574222334825528, 18.48520013137778, 20.642319769097874, 21.517217061489742,
37.11932335364917, 30.96961519949251, 41.21566056653424, 30.946751469826374,
18.924922317529127, 20.74365981119373, 18.784408811799718, 20.76460294291812,
40.29673692906268, 21.576609215299673, 12.796163939168583, 19.838167019974176,
14.271239521072557, 19.355017792519313, 32.18052482685483, 12.439896713179882,
26.79102618541826, 31.4541927986086, 28.474500046831622, 19.590376774250156,
16.573712117409944, 32.35403281161078, 21.820921865396084, 38.40311732347468,
15.078933925404982, 11.659828991901536, 16.49830617594752, 32.075254852780205,
30.772977257445238, 15.111607138959597, 25.52365952851958, 21.72326468012537,
16.481696673955266, 31.56614937938868, 18.512314862150525, 17.479382599473627,
36.72067103875451, 21.687177949357295, 21.037800149277274, 20.641661436736392,
32.78216794397545, 22.441713226132137, 17.6603223817236, 14.04784765147586,
14.247639123865504, 34.281810835178966, 14.040430598248461, 18.20298366853561,
25.298148644088545, 13.256284796703268, 15.642306650365855, 33.591667210209174,
23.77578433620688, 19.08236546055909, 32.43355871333175, 23.315193078439453,

```
24.64976561197613, 32.24974637074071, 22.0159492045891, 22.6596658786956,
16.885116359461424, 16.325449496262056, 23.586214328650296, 23.071673514135682,
15.861448479350997, 25.62349544959617, 22.666928095240678, 17.110764883405515,
13.235930754813115, 20.01762889707972, 25.37147464872067, 26.335017477169988,
23.480579840070597, 21.718171689420856, 24.32084302999008, 24.308809054980287,
13.575641767799087, 34.86488898551629, 17.095036111319633, 22.669448281218905,
27.49245067071524, 32.253676470380356, 32.78757405006918, 17.494531539497512,
24.053508752760727, 14.175369467412803, 20.39299248398923, 23.33232864618937,
25.052625460653655, -2.964665723658726, 18.617571179006372, 20.415230179569832,
27.365698982934763, 20.88904689102858, 15.745275312040508, 13.166968953603199,
18.19064773899586, 21.1557075317814, 17.0382930416128, 42.44926764484017,
23.502104414181176, 23.206126249503644, 13.44412513035238, 22.80302008769379,
21.641676064977872, 30.687889200555126, 37.65722228365474, 15.43497612007271,
28.355610767405505, 13.891809942698762, 24.594192357226756, 25.596822971506036,
13.84573800876732, 27.656262231977838, 19.13271258847255, 24.84323653387534,
32.519099433106625, 24.712542361009493, 25.34370909442649, 17.170322669348103,
25.38313064882725, 22.670945928453104, 12.754392277669513, 32.1736648102739,
21.694049909273915, 19.84843265610875, 31.434172268036093, 22.85909692695456,
28.450721176270214, 28.442601503149113, 33.38258258026427, 22.472899823964436,
22.716107386399205, 8.77891103049231, 17.45683336455088, 13.330403519092606,
21.584494821880725, 33.93316816900304, 34.55813221868196, 30.850286057068697,
24.230463593868052, 25.367183810330893, 27.26158060041313, 18.899257182693713,
12.884335998985206, 14.702985301608262, 23.455406890207005, 26.245780266996533,
18.27699030962122, 19.212866178796997, 27.22554651227808, 25.610832213174163,
19.04532561341212, 36.78244478323715, 37.71023336970519, 21.367523906599324,
20.10474097531688, 1.7556725110505838, 18.684040660486676, 38.73058897547675,
17.81406434344165, 25.999080671267564, 29.743999280211064, 33.199811782404495,
16.682952566578003, 25.310523054125294, 18.13981229684893, 29.213346107103007,
7.323575900042821, 21.768275008605176, 12.629295869409253, 39.9894993439187,
28.258092562942586, 15.764900901460923, 19.68671856325919, 7.516390535638763,
19.953586404910148, 24.536334826095413, 17.8692994162451, 6.757880929666975,
21.794351807577915, 18.502994324153214, 21.777738037261894, 27.506696228553846,
17.778269378444424, 18.259585569983283, 28.455534520144134, 18.03527140588134,
21.904708801574337, 23.321510111719245, 34.76675381859925, 23.640244600363772,
14.869934767709736, 19.777072820513016, 23.818583242719004]
```

[131]:
```python
r2 = r2_score(y_train,y_pred)
rmse = np.sqrt(mean_squared_error(y_train,y_pred))
print('R2 Score: ', r2)
print('Root Mean Sq Score: ',rmse)
```

```
R2 Score:  0.730647531347494
Root Mean Sq Score:  4.720391098867944
```

[134]:
```python
#Training model on training data and predicting on test data
y_pred=regression.predict(x_test)
print(y_train.to_list())
```

```python
print(list(y_pred))
r2 = r2_score(y_test,y_pred)
rmse = np.sqrt(mean_squared_error(y_test,y_pred))
print('R2 Score: ', r2)
print('Root Mean Sq Score: ',rmse)
```

```
[31.6, 11.3, 14.5, 24.5, 41.3, 16.3, 7.2, 50.0, 21.2, 22.9, 22.0, 19.5, 13.0,
48.8, 27.5, 21.7, 17.8, 19.1, 18.5, 8.5, 19.1, 13.8, 15.7, 9.5, 19.5, 27.5,
17.5, 14.0, 34.9, 26.4, 33.0, 14.8, 16.6, 21.7, 6.3, 22.6, 24.8, 19.4, 14.2,
13.8, 35.1, 11.8, 30.1, 42.8, 17.5, 22.7, 10.5, 17.8, 21.9, 25.0, 19.3, 17.8,
23.1, 23.0, 50.0, 21.4, 5.0, 29.0, 19.0, 23.1, 19.6, 50.0, 21.2, 43.8, 17.9,
22.4, 21.0, 18.4, 36.4, 25.0, 30.1, 25.0, 12.7, 13.6, 10.5, 41.7, 20.2, 20.8,
14.1, 12.5, 29.4, 50.0, 23.2, 32.5, 21.6, 11.7, 10.8, 50.0, 21.8, 19.7, 16.5,
25.3, 18.3, 17.2, 20.7, 28.6, 22.0, 16.5, 16.1, 24.8, 8.5, 32.4, 19.0, 23.7,
8.1, 15.6, 37.6, 43.5, 20.4, 21.5, 18.8, 16.8, 18.7, 23.4, 23.9, 20.1, 32.7,
15.1, 18.2, 21.7, 24.7, 24.3, 22.6, 50.0, 14.6, 21.0, 21.1, 22.5, 35.4, 21.5,
23.9, 17.6, 11.0, 8.7, 22.5, 20.9, 22.1, 16.2, 27.5, 22.0, 22.0, 13.2, 18.9,
17.8, 14.6, 7.5, 25.0, 18.6, 17.4, 13.4, 24.4, 8.8, 19.6, 20.0, 34.9, 33.4,
29.8, 33.3, 13.3, 28.1, 46.7, 24.4, 23.2, 20.4, 31.2, 24.3, 23.1, 22.0, 13.1,
24.8, 23.3, 14.4, 20.3, 32.0, 36.5, 23.9, 21.6, 19.7, 19.1, 36.2, 30.8, 19.9,
20.0, 23.3, 48.3, 29.9, 50.0, 37.0, 10.9, 20.6, 27.9, 18.7, 50.0, 17.4, 13.5,
17.7, 13.4, 17.1, 31.5, 15.6, 22.3, 32.9, 22.9, 22.2, 13.1, 32.2, 21.7, 45.4,
10.4, 12.0, 17.5, 27.9, 28.4, 15.6, 22.2, 19.8, 18.9, 29.1, 19.3, 12.7, 43.1,
20.3, 24.5, 19.3, 31.6, 23.1, 19.9, 10.9, 13.3, 31.0, 14.3, 16.1, 23.8, 12.3,
11.7, 28.5, 50.0, 18.3, 33.2, 19.4, 28.7, 29.0, 20.6, 19.8, 20.6, 10.2, 24.4,
19.2, 8.4, 22.3, 20.0, 14.9, 12.8, 21.4, 23.9, 22.8, 22.7, 11.9, 22.8, 26.2,
18.5, 39.8, 19.4, 21.2, 26.6, 29.8, 31.7, 17.1, 23.1, 17.2, 20.4, 21.7, 29.6,
7.0, 22.6, 23.1, 22.8, 18.9, 13.8, 15.6, 15.4, 16.1, 14.3, 48.5, 23.0, 20.8,
13.9, 23.0, 20.2, 25.1, 44.0, 18.4, 23.7, 11.5, 23.8, 24.1, 13.4, 22.8, 20.3,
24.0, 28.2, 18.5, 24.2, 10.2, 23.2, 19.4, 12.7, 50.0, 21.4, 16.2, 30.7, 17.0,
23.6, 28.0, 32.0, 21.1, 26.6, 7.0, 18.1, 23.7, 25.0, 30.1, 33.8, 31.5, 23.3,
15.0, 23.3, 18.2, 11.8, 18.2, 20.1, 23.7, 20.1, 20.6, 25.2, 26.5, 19.6, 33.2,
42.3, 21.2, 19.4, 13.8, 19.5, 21.9, 23.1, 24.8, 23.0, 37.9, 20.8, 21.7, 20.0,
24.1, 11.9, 21.2, 8.3, 50.0, 24.6, 20.1, 15.3, 10.4, 19.5, 22.6, 12.6, 8.4,
22.2, 12.1, 19.3, 22.0, 14.5, 14.5, 28.4, 7.2, 20.9, 20.5, 37.3, 22.4, 17.2,
19.2, 21.9]
[11.79886390884474, 19.007241580797846, 30.155699381483593, 23.816257251664148,
7.0053493903987665, 24.012905378064428, 21.021583528595926, 19.218417109982393,
43.38294937454914, 20.624138679044677, 26.396168035498345, 24.046187887471103,
25.852603466969878, 17.78208867484305, 10.529632719152168, 33.080094660063054,
19.305395878784335, 18.926745788335264, 24.769236760925054, 17.357732320997727,
19.56323069644976, 23.361381127941673, 32.82075533753671, 24.932591249386608,
35.84221428591097, 24.64559059461133, 28.167411774230956, 34.69448847542159,
20.652778537577646, 26.80558669699304, 21.235128341564238, 24.862869879200876,
15.711855766756486, 20.939783191180055, 19.148069084515576, 24.296738757206903,
36.9532098781208, 18.617961499210114, 30.293359522540264, 32.01991929515857,
22.931025104122572, 14.664341791862263, 19.208915629538335, 19.500242460449783,
```

```
19.91516961439937, 20.220648863201955, 30.36888079984444, 17.653872711870445,
22.7724668708535, 18.068422408586756, 17.079553172395386, 19.452781137740207,
15.351627878330548, 30.155490263726655, 46.2313754214958, 6.596741804413282,
41.02477652035137, 26.540808410166704, 16.390239905949308, 22.427711400700026,
10.121647920440182, 23.04344969149863, 12.70047754008469, 22.179696114779127,
24.6443072250818, 9.971705474583544, 27.51050763731752, 27.18312044702352,
18.143590017610585, 22.087736230862273, 16.51343670694519, 20.74119919655274,
29.415197030894277, 19.866840568719546, 17.57215192024931, 35.93451934051168,
20.214639585907992, 14.408620475083875, 18.943568160830733, 16.611306866568302,
8.919341418737035, 20.05588671600288, 32.44295929837773, 22.4719840021239,
22.072387971533757, 28.27050389357794, 18.608333225313846, 16.634567736759763,
12.920990952800945, 33.862920569729276, 29.046758985142063, 21.266604155924565,
30.72039818900508, 14.12580167695295, 19.2508862735914, 20.614086550600877,
17.59042857724502, 20.652352462726498, 33.88653533040376, 24.664666022311906,
14.495373611667546, 35.82489035581906, 17.365319415932987, 23.745043659674288,
19.223829332900387, 15.969942641169508, 30.101847573539224, 20.879002866112476,
23.30821292665901, 18.34264668967441, 11.385610165730183, 38.271922562623445,
27.445344695743895, 28.90381446928859, 35.33585382258353, 36.19121599027475,
14.774283649161891, 33.58186961771334, 20.562375069719398, 21.354699721114294,
20.49553842201783, 6.554267272024408, 21.09276571216047, 12.960203770335838,
34.94596148445363, 8.608967582928157, 32.64143591536392]
R2 Score:  0.7519030329262333
Root Mean Sq Score:  4.709397726218281
```

[136]:

```
st_file='stand_scaler.pickle'
pickle.dump(scaler, open(st_file, 'wb'))
```

[137]:

```
# saving the model to the local file system
filename = 'linear_model.pickle'
pickle.dump(regression, open(filename, 'wb'))
```

[138]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,␣
 ↪roc_auc_score
from sklearn.linear_model  import Ridge,Lasso,RidgeCV, LassoCV, ElasticNet,␣
 ↪ElasticNetCV
```

[144]:

```
# Lasso Regularization
# LassoCV will return best alpha and coefficients after performing 10 cross␣
 ↪validations
lasscv = LassoCV(alphas = None,cv =10, max_iter = 100000, normalize = True)
lasscv.fit(x_train, y_train)
```

[144]:

```
LassoCV(cv=10, max_iter=100000, normalize=True)
```

[145]:

```
# best alpha parameter
alpha = lasscv.alpha_
print(alpha)
```

```
#now that we have best parameter, let's use Lasso regression and see how well␣
↪our data has fitted before

lasso_reg = Lasso(alpha)
lasso_reg.fit(x_train, y_train)

print(lasso_reg.score(x_test, y_test))
```

```
0.00023566421563150474
0.7519176536247909
```

After L1 regularization RMS value is same as that of normal Linear Regression. This means our model does not overfit.

[146]:
```
# Using Ridge regression model
# RidgeCV will return best alpha and coefficients after performing 10 cross␣
↪validations.
# We will pass an array of random numbers for ridgeCV to select best alpha from␣
↪them

alphas = np.random.uniform(low=0, high=10, size=(50,))
ridgecv = RidgeCV(alphas = alphas,cv=10,normalize = True)
ridgecv.fit(x_train, y_train)

ridge_model = Ridge(alpha=ridgecv.alpha_)
ridge_model.fit(x_train, y_train)

ridge_model.score(x_test, y_test)
```

[146]: 0.7510566174329587

After L2 regularization also RMS value is same as that of normal Linear Regression. This means our model does not overfit.

[148]:
```
# Elastic net
elasticCV = ElasticNetCV(alphas = None, cv =10)
elasticCV.fit(x_train, y_train)

print(elasticCV.alpha_)
# l1_ration gives how close the model is to L1 regularization, below value␣
↪indicates we are giving equal
#preference to L1 and L2
print(elasticCV.l1_ratio)

elasticnet_reg = ElasticNet(alpha = elasticCV.alpha_,l1_ratio=0.5)
elasticnet_reg.fit(x_train, y_train)
elasticnet_reg.score(x_test, y_test)
```

```
1.5740233055986073
0.5
```

[148]:
```
0.7115488475279687
```

So, we can see by using different type of regularization, we still are getting the same r2 score. That means our OLS model has been well trained over the training data and there is no overfitting.