In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

```
In [5]: from sklearn.datasets import load_boston
        boston = load_boston()
```

```
C:\Users\Joshua Deshmukh\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

In [6]:
```
boston.DESCR
```

Out[6]:
```
".. _boston_dataset:\n\nBoston house prices dataset\n---------------------------\n\n**Data Set Characteristics:**  \n\n    :Numbe
r 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       proportio
n of residential land zoned for lots over 25,000 sq.ft.\n        - INDUS    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 a
ccessibility to radial highways\n        - TAX      full-value property-tax rate per $10,000\n        - PTRATIO  pupil-teacher ra
tio by town\n        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town\n        - LSTAT    % lower
status of the population\n        - MEDV     Median value of owner-occupied homes in $1000's\n\n    :Missing Attribute Values: No
ne\n\n    :Creator: Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttps://archive.ics.uci.edu/m
l/machine-learning-databases/housing/\n\n\nThis dataset was taken from the StatLib library which is maintained at Carnegie Mellon
University.\n\nThe Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air', J.
Environ. Economics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley, 198
0.   N.B. Various transformations are used in the table 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 & W
elsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R.
(1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learni
ng, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.\n"
```

In [7]:
```
df = pd.DataFrame(boston.data,columns=boston.feature_names)
df['Price'] = boston.target
```

```
In [8]: df
```

```
Out[8]:
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  PTRATIO       B  LSTAT  Price
  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   24.0
  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   21.6
  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   34.7
  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   33.4
  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   36.2
...      ...   ...    ...   ...    ...    ...   ...     ...  ...    ...      ...     ...    ...    ...
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   9.67   22.4
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   9.08   20.6
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   5.64   23.9
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   6.48   22.0
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   7.88   11.9

506 rows × 14 columns
```

```
In [9]: df.describe()
```

```
Out[9]:
             CRIM          ZN       INDUS        CHAS         NOX          RM         AGE         DIS         RAD         TAX     PTRATIO           B      LS
count  506.000000  506.000000  506.000000  506.000000  506.000000  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    3.795043    9.549407  408.237154   18.455534  356.674032   12.65
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617   28.148861    2.105710    8.707259  168.537116    2.164946   91.294864    7.14
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000    2.900000    1.129600    1.000000  187.000000   12.600000    0.320000    1.73
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500   45.025000    2.100175    4.000000  279.000000   17.400000  375.377500    6.95
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500   77.500000    3.207450    5.000000  330.000000   19.050000  391.440000   11.36
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500   94.075000    5.188425   24.000000  666.000000   20.200000  396.225000   16.95
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000  100.000000   12.126500   24.000000  711.000000   22.000000  396.900000   37.97
```

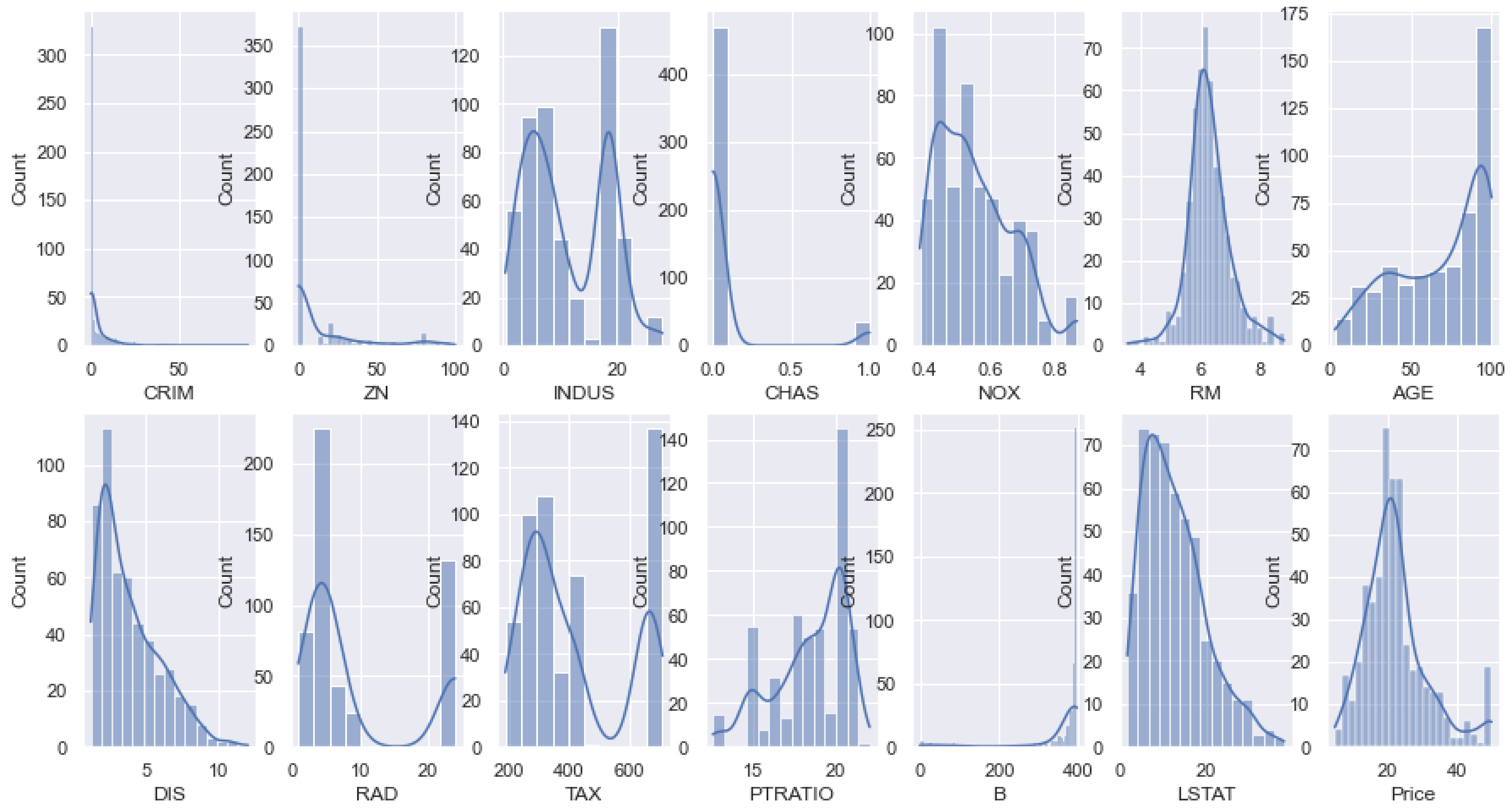
```
In [10]: df.info()
```

```
<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 [11]: df.isnull().sum()
```

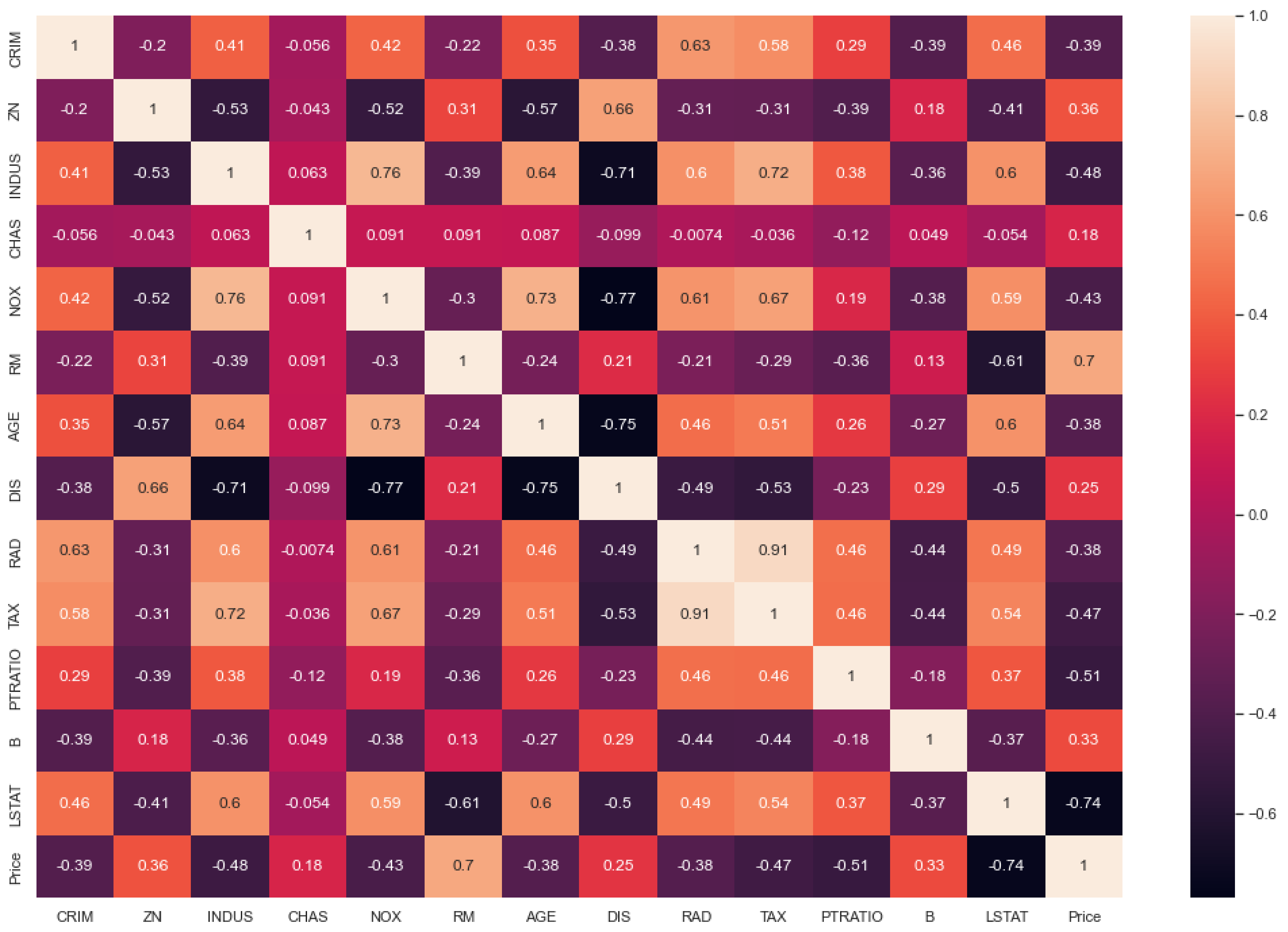
```
Out[11]: CRIM       0
         ZN         0
         INDUS      0
         CHAS       0
         NOX        0
         RM         0
         AGE        0
         DIS        0
         RAD        0
         TAX        0
         PTRATIO    0
         B          0
         LSTAT      0
         Price      0
         dtype: int64
```

```
In [13]: row = 2
         col = 7
         s = 0
         fig, ax=plt.subplots(nrows=row, ncols=col, figsize=(15,8))
         for i in range(row):
             for j in range(col):
                 sns.histplot(x=df.columns[s], data=df, kde=True, ax=ax[i][j])
                 s+=1
```



In [14]:
```
fig, ax = plt.subplots(figsize = (18,12))
sns.heatmap(data=df.corr(), annot=True, ax=ax)
```

Out[14]:
```
<AxesSubplot:>
```



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
CRIM	1	-0.2	0.41	-0.056	0.42	-0.22	0.35	-0.38	0.63	0.58	0.29	-0.39	0.46	-0.39
ZN	-0.2	1	-0.53	-0.043	-0.52	0.31	-0.57	0.66	-0.31	-0.31	-0.39	0.18	-0.41	0.36
INDUS	0.41	-0.53	1	0.063	0.76	-0.39	0.64	-0.71	0.6	0.72	0.38	-0.36	0.6	-0.48
CHAS	-0.056	-0.043	0.063	1	0.091	0.091	0.087	-0.099	-0.0074	-0.036	-0.12	0.049	-0.054	0.18
NOX	0.42	-0.52	0.76	0.091	1	-0.3	0.73	-0.77	0.61	0.67	0.19	-0.38	0.59	-0.43
RM	-0.22	0.31	-0.39	0.091	-0.3	1	-0.24	0.21	-0.21	-0.29	-0.36	0.13	-0.61	0.7
AGE	0.35	-0.57	0.64	0.087	0.73	-0.24	1	-0.75	0.46	0.51	0.26	-0.27	0.6	-0.38
DIS	-0.38	0.66	-0.71	-0.099	-0.77	0.21	-0.75	1	-0.49	-0.53	-0.23	0.29	-0.5	0.25
RAD	0.63	-0.31	0.6	-0.0074	0.61	-0.21	0.46	-0.49	1	0.91	0.46	-0.44	0.49	-0.38
TAX	0.58	-0.31	0.72	-0.036	0.67	-0.29	0.51	-0.53	0.91	1	0.46	-0.44	0.54	-0.47
PTRATIO	0.29	-0.39	0.38	-0.12	0.19	-0.36	0.26	-0.23	0.46	0.46	1	-0.18	0.37	-0.51
B	-0.39	0.18	-0.36	0.049	-0.38	0.13	-0.27	0.29	-0.44	-0.44	-0.18	1	-0.37	0.33
LSTAT	0.46	-0.41	0.6	-0.054	0.59	-0.61	0.6	-0.5	0.49	0.54	0.37	-0.37	1	-0.74
Price	-0.39	0.36	-0.48	0.18	-0.43	0.7	-0.38	0.25	-0.38	-0.47	-0.51	0.33	-0.74	1

```
In [15]: cor=df.corr()
         features=[]
         for i in cor['Price'].index:
             if abs(cor['Price'][i]) < 0.45:
                 features.append(i)
         df = df.drop(columns=features)
```

```
In [16]: df
```

```
Out[16]:
         INDUS     RM    TAX  PTRATIO  LSTAT  Price
    0     2.31  6.575  296.0     15.3   4.98   24.0
    1     7.07  6.421  242.0     17.8   9.14   21.6
    2     7.07  7.185  242.0     17.8   4.03   34.7
    3     2.18  6.998  222.0     18.7   2.94   33.4
    4     2.18  7.147  222.0     18.7   5.33   36.2
  ...      ...    ...    ...      ...    ...    ...
  501    11.93  6.593  273.0     21.0   9.67   22.4
  502    11.93  6.120  273.0     21.0   9.08   20.6
  503    11.93  6.976  273.0     21.0   5.64   23.9
  504    11.93  6.794  273.0     21.0   6.48   22.0
  505    11.93  6.030  273.0     21.0   7.88   11.9

506 rows × 6 columns
```

```
In [17]: x = df.drop(labels=['Price'],axis=1)
         y = df['Price']
```

```
In [18]: x.head()
```

Out[18]:

	INDUS	RM	TAX	PTRATIO	LSTAT
0	2.31	6.575	296.0	15.3	4.98
1	7.07	6.421	242.0	17.8	9.14
2	7.07	7.185	242.0	17.8	4.03
3	2.18	6.998	222.0	18.7	2.94
4	2.18	7.147	222.0	18.7	5.33

```
In [19]: from sklearn.model_selection import train_test_split
         x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3, random_state=42)
```

```
In [20]: x_train.shape , x_test.shape

Out[20]: ((354, 5), (152, 5))
```

```
In [21]: from sklearn.linear_model import LinearRegression
         lr = LinearRegression()
         lr.fit(x_train,y_train)

Out[21]: LinearRegression()
```

```
In [22]: lr.coef_

Out[22]: array([ 8.89192908e-02,  4.58472965e+00, -3.21801353e-03, -8.30469649e-01,
                -6.14838082e-01])
```

```
In [23]: lr.intercept_

Out[23]: 17.150526471659887
```

```
In [24]: y_pred = lr.predict(x_test)
         y_pred
```

```
Out[24]: array([26.62981059, 31.10008241, 16.95701338, 25.59771173, 18.09307064,
                22.90871478, 17.61601889, 13.53637406, 20.48659658, 19.63042973,
                20.14356915, 21.23936603, -2.17703728, 22.46574239, 19.55741359,
                24.67368263, 19.10912156,  3.38842032, 38.69074984, 17.05082016,
                26.1619823 , 27.58438964, 11.74375654, 24.17459161, 17.7242101 ,
                13.56944671, 22.81648456, 19.18372228, 18.46079157, 18.70360048,
                19.2864636 , 25.64061437, 25.10598582, 18.15940204, 14.25506589,
                21.85273114, 32.68155189, 20.99806076, 20.54233085, 25.18099014,
                12.5229096 , 28.12041259, 39.52307563, 18.48292269, 25.8060045 ,
                15.51396261, 14.30180164, 26.47373087, 18.0995971 , 30.77991619,
                23.61711953, 33.50182842, 16.09617732, 25.6923864 , 38.16291719,
                22.27510835, 18.15355162, 30.2656657 , 24.76833162, 15.09517744,
                25.49349461, 31.94366429, 29.9910815 , 17.2733645 , 27.56807723,
                12.05824047, 18.98303931, 25.66074343, 28.98449592, 15.55155416,
                19.97704179, 26.02936009, 12.05329111, 21.7655206 , 23.31666209,
                 6.03391556, 20.01221693, 38.12144014, 16.8875505 , 10.72075752,
                22.46088278,  9.36198278, 24.09343534,  7.14093828, 22.17983415,
                28.15116909, 20.46791378, 26.25420747, 27.1620727 , 20.70409881,
                24.1563809 ,  7.77940126, 21.54982615, 20.04604896, 11.78985685,
                22.46163913, 22.48055429, -0.07257436, 18.36539259, 17.31496289,
                21.06156734, 24.80376727,  8.84307115, 20.98617093, 25.51248652,
                13.50230783, 18.76790985, 26.96699475, 22.8688555 , 27.5586772 ,
                11.78407124, 19.09435201, 24.89518268, 24.32419065, 31.1937723 ,
                18.97618967, 32.91808139, 14.96546128, 19.58974059, 28.640961  ,
                18.70459566, 26.86127531, 14.47455498, 23.13989195, 26.86074285,
                23.56807441, 27.48834026, 31.80605265, 24.3571917 , 37.91833369,
                11.93795084, 26.54212343, 19.26752655, 19.29065717, 11.75099637,
                21.90166758, 22.66230052, 32.72490364, 31.18752369, 16.58461442,
                17.88939265, 29.94107039, 23.23104109, 11.78611835,  8.36515471,
                24.64330163, 24.19025911, 17.20942048, 13.87458909, 39.4187899 ,
                19.37592804, 18.39248868])
```

```
In [25]: pred_df=pd.DataFrame(np.c_[ y_test, y_pred], columns = ["Price_original","Price_predicted"])
         pred_df
```

Out[25]:

	Price_original	Price_predicted
0	23.6	26.629811
1	32.4	31.100082
2	13.6	16.957013
3	22.8	25.597712
4	16.1	18.093071
...	...	...
147	17.1	17.209420
148	14.5	13.874589
149	50.0	39.418790
150	14.3	19.375928
151	12.6	18.392489

152 rows × 2 columns

```
In [26]: from sklearn.metrics import r2_score
         lr.score(x_test,y_test)

Out[26]: 0.6499135956539925
```

```
In [27]: from sklearn.ensemble import RandomForestRegressor
```

```
In [28]: rf = RandomForestRegressor()
         rf.fit(x_train,y_train)

Out[28]: RandomForestRegressor()
```

```
In [29]: y_pred = rf.predict(x_test)
         y_pred
```

```
Out[29]: array([22.79 , 31.42 , 17.212, 23.11 , 15.309, 21.744, 20.294, 14.599,
                21.656, 21.527, 21.135, 20.206, 12.55 , 21.966, 18.41 , 25.179,
                19.719,  8.576, 46.466, 15.646, 24.447, 24.189, 14.054, 23.025,
                14.996, 14.878, 24.305, 16.193, 20.784, 21.237, 18.798, 23.29 ,
                26.444, 21.41 , 10.588, 16.656, 35.483, 19.321, 20.404, 23.269,
                15.53 , 29.268, 45.572, 20.356, 23.262, 14.475, 17.28 , 23.639,
                15.239, 29.454, 22.301, 34.702, 17.868, 25.872, 43.895, 21.051,
                15.809, 33.086, 22.272, 19.617, 25.793, 34.747, 29.585, 19.324,
                26.71 , 20.405, 15.626, 22.965, 28.081, 20.448, 20.465, 31.97 ,
                10.85 , 22.128, 22.047,  7.744, 20.521, 47.725, 12.196, 11.4  ,
                22.783,  8.313, 23.154,  9.293, 21.6  , 27.652, 15.   , 23.166,
                23.439, 17.367, 21.153,  8.465, 19.754, 19.614, 32.306, 19.631,
                26.371, 10.978, 14.92 , 12.454, 19.932, 27.169, 11.098, 21.447,
                21.883, 12.086, 18.351, 24.556, 20.315, 23.644,  7.698, 13.563,
                23.387, 25.076, 32.805, 15.947, 43.107, 16.964, 17.706, 23.838,
                20.121, 24.059,  9.675, 20.379, 23.965, 21.377, 24.495, 35.854,
                18.997, 47.512, 17.886, 22.144, 19.358, 18.827, 14.21 , 21.555,
                21.367, 32.007, 29.392, 16.872, 17.313, 25.865, 20.816, 20.508,
                 8.312, 22.072, 18.461, 12.582, 13.678, 42.133, 15.365, 15.653])
```

```
In [30]: rf.score(x_test,y_test)
```

```
Out[30]: 0.8013549394401752
```

```
In [ ]:
```