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

In [1]:
```
column_names =['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PTRA
```

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_boston
import seaborn as sns
import pickle
%matplotlib inline


def get_data():
    with open('A:\MLDL\housing.csv', 'r') as open_:
        values = open_.readlines()
    data_lis =[]
    for i in values:
        data_lis.append(i.split())
    a = pd.DataFrame(data_lis,columns = column_names)
    return a

data = get_data()
```

In [3]:
```
data
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	BK
0	0.00632	18.00	2.310	0	0.5380	6.5750	65.20	4.0900	1	296.0	15.30	396.90

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	BK
1	0.02731	0.00	7.070	0	0.4690	6.4210	78.90	4.9671	2	242.0	17.80	396.90
2	0.02729	0.00	7.070	0	0.4690	7.1850	61.10	4.9671	2	242.0	17.80	392.83
3	0.03237	0.00	2.180	0	0.4580	6.9980	45.80	6.0622	3	222.0	18.70	394.63
4	0.06905	0.00	2.180	0	0.4580	7.1470	54.20	6.0622	3	222.0	18.70	396.90
...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.00	11.930	0	0.5730	6.5930	69.10	2.4786	1	273.0	21.00	391.99
502	0.04527	0.00	11.930	0	0.5730	6.1200	76.70	2.2875	1	273.0	21.00	396.90
503	0.06076	0.00	11.930	0	0.5730	6.9760	91.00	2.1675	1	273.0	21.00	396.90
504	0.10959	0.00	11.930	0	0.5730	6.7940	89.30	2.3889	1	273.0	21.00	393.45
505	0.04741	0.00	11.930	0	0.5730	6.0300	80.80	2.5050	1	273.0	21.00	396.90

In [4]:
```
data.dtypes
```

Out[4]:
```
CRIM       object
ZN         object
INDUS      object
CHAS       object
NOX        object
RM         object
AGE        object
DIS        object
RAD        object
TAX        object
PTRATIO    object
BK         object
LSTAT      object
PRICE      object
dtype: object
```

In [5]:
```
for i in data.columns:
    data[i] = [float(j) for j in data[i]]
```

In [6]:
```
data
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	BK	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	4.9
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	9.1
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	4.0
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	2.9
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	5.3
...	...	...	...	...	...	...	...	...	...	...	...	...	
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	9.6
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	9.0

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	BK	LSTA
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	5.6
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	6.4
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	7.8

In [7]:
```
# Checking for Null Values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
BK         506 non-null float64
LSTAT      506 non-null float64
PRICE      506 non-null float64
dtypes: float64(14)
memory usage: 55.5 KB
```

In [8]:
```
data.isna().sum()
```

Out[8]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
BK         0
LSTAT      0
PRICE      0
dtype: int64
```

In [9]:
```
descriptions = data.describe()
descriptions
```

Out[9]:

	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

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
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

In [10]:
```
descriptions
```

Out[10]:

	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 [11]:
```
data.median()
```

Out[11]:
```
CRIM         0.25651
ZN           0.00000
INDUS        9.69000
CHAS         0.00000
NOX          0.53800
RM           6.20850
AGE         77.50000
DIS          3.20745
RAD          5.00000
TAX        330.00000
PTRATIO     19.05000
BK         391.44000
LSTAT       11.36000
PRICE       21.20000
dtype: float64
```

From the above description we can see the results of Mean, Standard Deviation, Median and various others of the data. To detect outliers we have to see the balance between Mean of that feature and Median of that feature.

> Mean > Median -- Positive Skewness
>
> Mean < Median -- Negative Skewness

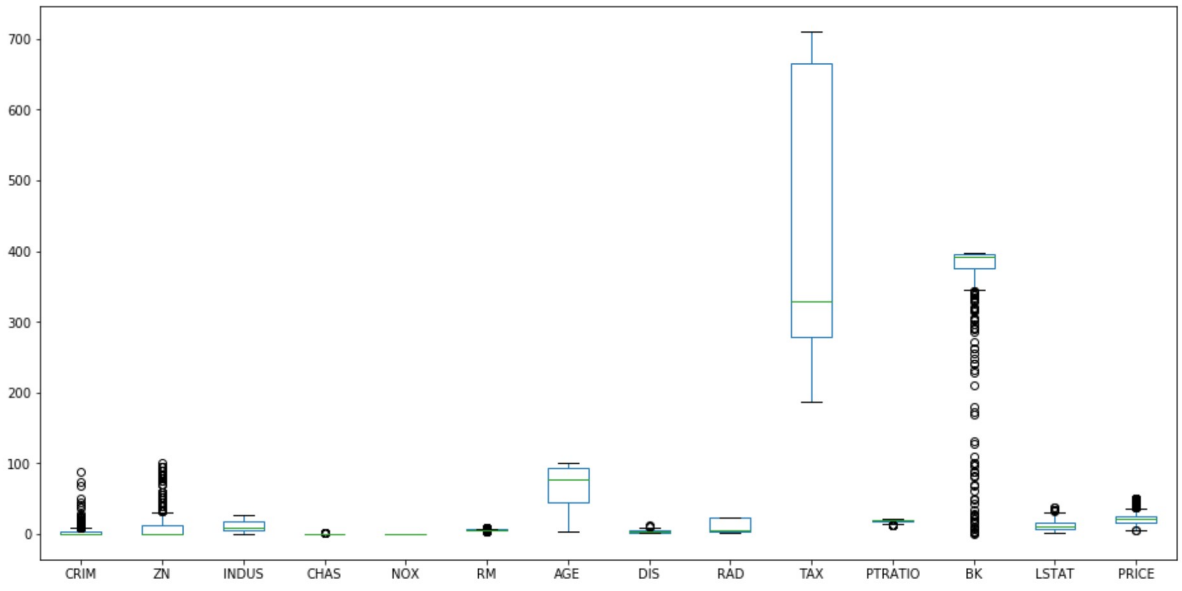
These indicates that our dataset contains outliers.

Feature	Mean	Median	Skewness
CRIM	3.613524	0.25651	Positive
INDUS	11.136779	9.69000	Positive
CHAS	0.069170	0.00000	Positive

Feature	Mean	Median	Skewness
NOX	0.554695	0.53800	Positive
RM	6.284634	6.20850	---
AGE	68.574901	77.50000	Negative
DIS	3.795043	3.20745	Positive
RAD	9.549407	5.00000	Positive
TAX	408.237154	330.00000	Positive
PTRATIO	18.455534	19.05000	Negative
BK	356.674032	391.44000	Negative
LSTAT	12.653063	11.36000	Positive

In [12]:
```
# Visualization of Outliers using Box plot
data_original = data.copy()
data_original.plot(kind='box',figsize=(16,8))
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x16ae3066b20>



As from the above plot we can clearly see there are outliers in our date set. Now Let's remove them.

Using Interquatile range to detect and remove outliers.

In [13]:
```
data_original.describe()
```

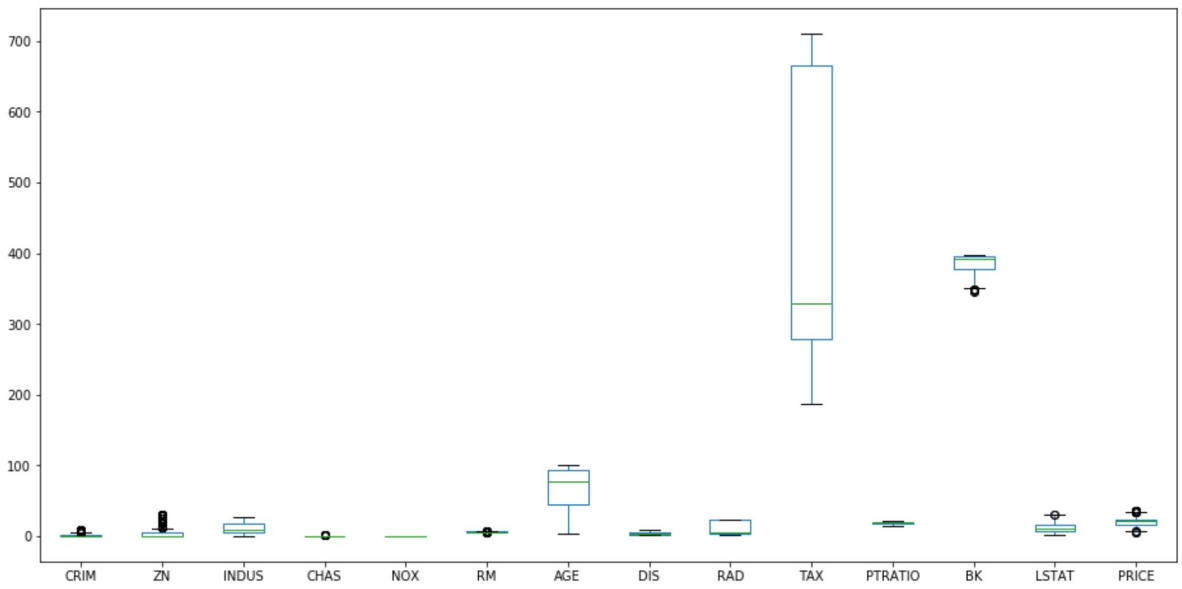
Out[13]:

	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

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
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

In [14]:
```
def rem_out(data_values): # Function to remove outliers from the dataset.
    #print(data_values.describe())
    length = len(data_values)
    col_name = data_values.columns
    des = data_values.describe()
    for col in col_name:
        #print(col)
        q1 = des[col]['25%']
        q3 = des[col]['75%']
        iqr = q3- q1
        min_ = (q1-(1.5*iqr))
        max_ = (q3+(1.5*iqr))
        if max_ == min_:
            continue
        #print(min_,max_)
        i=0
        while i<length:
            #print('i= ',i)
            #print('Length = ',length)
            value = data_values[col].iloc[i]
            if value >=max_ or value <=min_:
                #print(i,  value)
                data_values[col].iloc[i] = data_values[col].mean()
                length = len(data_values)
            i+=1
            data_values.reset_index(drop=True)
    data_values.plot(kind='box',figsize=(16,8))
    #print(data_values.describe())
    return data_values
```

In [15]:
```
data_df1 = rem_out(data.copy())
```



In [16]:
```
data_df1.isna().sum()
```

Out[16]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
BK         0
LSTAT      0
PRICE      0
dtype: int64
```

Checking Collinearity

Using Correlation to check for Collinearity

In [17]:
```
data_df1.corr()
```

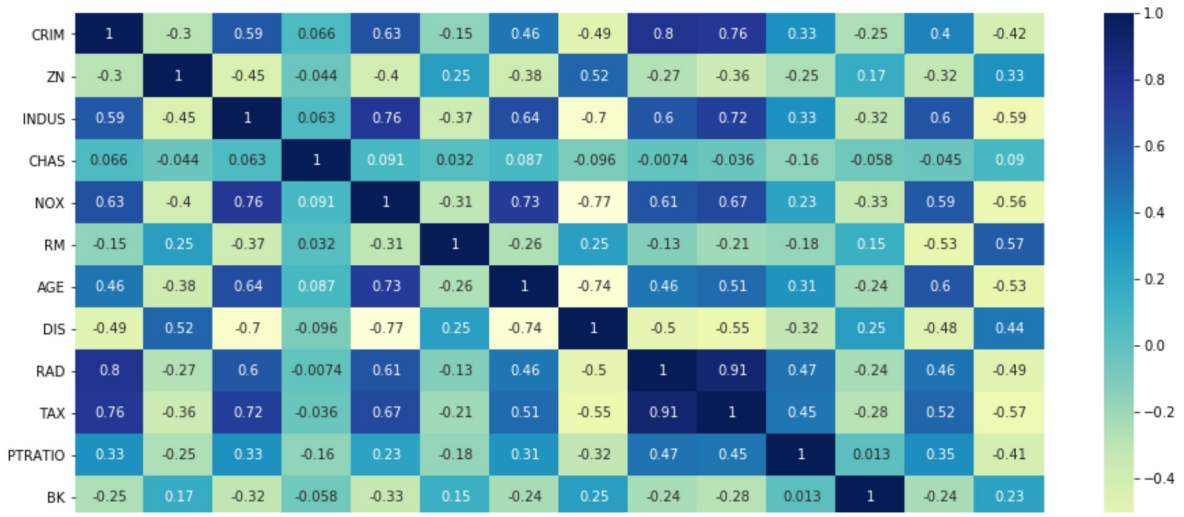
Out[17]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
CRIM	1.000000	-0.295162	0.588655	0.066006	0.625434	-0.151322	0.458455	-0.488797	(
ZN	-0.295162	1.000000	-0.446895	-0.043763	-0.396811	0.254585	-0.378445	0.520028	-(
INDUS	0.588655	-0.446895	1.000000	0.062938	0.763651	-0.371376	0.644779	-0.704480	(
CHAS	0.066006	-0.043763	0.062938	1.000000	0.091203	0.031663	0.086518	-0.095710	-(
NOX	0.625434	-0.396811	0.763651	0.091203	1.000000	-0.305767	0.731470	-0.773815	(
RM	-0.151322	0.254585	-0.371376	0.031663	-0.305767	1.000000	-0.258570	0.248824	-(
AGE	0.458455	-0.378445	0.644779	0.086518	0.731470	-0.258570	1.000000	-0.742173	(
DIS	-0.488797	0.520028	-0.704480	-0.095710	-0.773815	0.248824	-0.742173	1.000000	-(
RAD	0.797086	-0.270816	0.595129	-0.007368	0.611441	-0.125438	0.456022	-0.504133	1
TAX	0.761196	-0.359608	0.720760	-0.035587	0.668023	-0.208653	0.506456	-0.551650	(
PTRATIO	0.332184	-0.251374	0.332144	-0.159330	0.234990	-0.175433	0.307892	-0.316034	(
BK	-0.250042	0.169226	-0.316099	-0.057981	-0.332659	0.152428	-0.236587	0.253852	-(
LSTAT	0.395124	-0.320980	0.599086	-0.045319	0.589775	-0.530714	0.601298	-0.480916	(
PRICE	-0.423200	0.327165	-0.587373	0.089638	-0.557076	0.567371	-0.533419	0.440055	-(

Ploting Heat map of correlation Values

In [18]:
```
fig = plt.figure(figsize=(16,8))
sns.heatmap(data=data_df1.corr(),cmap="YlGnBu", annot=True)
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x16ae5329700>
```



In [19]:

```
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
scaler = StandardScaler()

dep_var = data_df1['PRICE']
ind_var = data_df1.drop('PRICE',axis=1)
x_data = scaler.fit_transform(ind_var)
vif_values = [vif(x_data,i) for i in range(x_data.shape[1])]
column_nam = ind_var.columns
for i in range(len(column_nam)):
    print(column_nam[i],"=",vif_values[i])
```

```
CRIM = 3.1098261776731344
ZN = 1.4796428497005059
INDUS = 3.7836182708129584
CHAS = 1.089344146650186
NOX = 4.414889956948909
RM = 1.523672035408481
AGE = 2.9820733326726456
DIS = 3.6975036128590055
RAD = 8.199167944510771
TAX = 8.634159253631685
PTRATIO = 1.5265664941555255
BK = 1.1716877112749342
LSTAT = 2.436080915385449
```

VIF Score of RAD and TAX Feature are greater than 5 , which indicates high collinearity.

Implementinh OLS Model using the features. including RAD and TAX.

Model 1

In [20]:

```
import statsmodels.formula.api as smf
ols_model_1 = smf.ols(formula= f'PRICE ~ CRIM+ZN+INDUS+CHAS+NOX+RM+AGE+DIS+RAD+TA
ols_model_1.summary()
```

Out[20]:

OLS Regression Results

Dep. Variable:	PRICE	**R-squared:**	0.655
Model:	OLS	**Adj. R-squared:**	0.645
Method:	Least Squares	**F-statistic:**	71.72
Date:	Sat, 18 Jun 2022	**Prob (F-statistic):**	1.41e-104

Time:	14:10:45	**Log-Likelihood:**	-1371.6
No. Observations:	506	**AIC:**	2771.
Df Residuals:	492	**BIC:**	2830.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	16.7903	6.459	2.600	0.010	4.100	29.481
CRIM	0.1232	0.142	0.869	0.385	-0.155	0.402
ZN	0.0140	0.027	0.524	0.601	-0.039	0.067
INDUS	-0.0275	0.047	-0.590	0.556	-0.119	0.064
CHAS	1.1742	0.675	1.740	0.082	-0.152	2.500
NOX	-5.3961	2.978	-1.812	0.071	-11.247	0.454
RM	3.3620	0.392	8.583	0.000	2.592	4.132
AGE	-0.0296	0.010	-2.935	0.003	-0.049	-0.010
DIS	-0.4287	0.159	-2.688	0.007	-0.742	-0.115
RAD	0.0122	0.054	0.227	0.821	-0.094	0.118
TAX	-0.0090	0.003	-3.145	0.002	-0.015	-0.003
PTRATIO	-0.3663	0.105	-3.495	0.001	-0.572	-0.160
BK	0.0098	0.014	0.721	0.471	-0.017	0.037
LSTAT	-0.3029	0.039	-7.837	0.000	-0.379	-0.227

Omnibus:	13.916	**Durbin-Watson:**	1.391
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	18.678
Skew:	0.261	**Prob(JB):**	8.79e-05
Kurtosis:	3.784	**Cond. No.**	2.32e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.32e+04. This might indicate that there are

strong multicollinearity or other numerical problems.
In this we can observe that Condition number is very high which indicates collinearity in data.
Model's R-squared: 0.655 and Adj. R-squared: 0.645 indicates that the model accuracy is 64%.

Using these data let's see OLS model without TAX and RAD Features.

Model 2

OLS model without TAX and RAD Features.

In [21]:
```
ols_model_2 = smf.ols(formula= f'PRICE ~ CRIM+ZN+INDUS+CHAS+NOX+RM+AGE+DIS+PTRATI(
ols_model_2.summary()
```

Out[21]:

OLS Regression Results

Dep. Variable:	PRICE	**R-squared:**	0.640
Model:	OLS	**Adj. R-squared:**	0.632
Method:	Least Squares	**F-statistic:**	79.97
Date:	Sat, 18 Jun 2022	**Prob (F-statistic):**	3.14e-102
Time:	14:10:45	**Log-Likelihood:**	-1381.8
No. Observations:	506	**AIC:**	2788.
Df Residuals:	494	**BIC:**	2838.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	17.7705	6.496	2.736	0.006	5.007	30.533
CRIM	-0.1733	0.112	-1.550	0.122	-0.393	0.046
ZN	0.0192	0.027	0.713	0.476	-0.034	0.072
INDUS	-0.1004	0.043	-2.335	0.020	-0.185	-0.016
CHAS	1.5379	0.681	2.259	0.024	0.200	2.875
NOX	-7.1438	3.003	-2.379	0.018	-13.044	-1.244
RM	3.1112	0.391	7.953	0.000	2.343	3.880
AGE	-0.0250	0.010	-2.453	0.015	-0.045	-0.005
DIS	-0.4603	0.161	-2.854	0.004	-0.777	-0.143
PTRATIO	-0.4795	0.101	-4.757	0.000	-0.678	-0.281
BK	0.0134	0.014	0.966	0.335	-0.014	0.041
LSTAT	-0.3269	0.039	-8.435	0.000	-0.403	-0.251

Omnibus:	8.190	**Durbin-Watson:**	1.365
Prob(Omnibus):	0.017	**Jarque-Bera (JB):**	11.277
Skew:	0.129	**Prob(JB):**	0.00356
Kurtosis:	3.684	**Cond. No.**	1.55e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.55e+04. This might indicate that there are

strong multicollinearity or other numerical problems.

In model 2 Condtion number is low as compared to model 1 and same goes for R square and

Adj R square.

As we keep on decreasing the feaures model accuracy keep decreasing .

Model 3

In [22]:
```
import statsmodels.formula.api as smf
ols_model_1 = smf.ols(formula= f'PRICE ~ CRIM+ZN+INDUS+CHAS+NOX+RM+AGE+PTRATIO+BK-
ols_model_1.summary()
```

Out[22]:

OLS Regression Results

Dep. Variable:	PRICE	**R-squared:**	0.634
Model:	OLS	**Adj. R-squared:**	0.627
Method:	Least Squares	**F-statistic:**	85.91
Date:	Sat, 18 Jun 2022	**Prob (F-statistic):**	1.80e-101
Time:	14:10:45	**Log-Likelihood:**	-1385.9
No. Observations:	506	**AIC:**	2794.
Df Residuals:	495	**BIC:**	2840.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	12.5939	6.282	2.005	0.046	0.250	24.937
CRIM	-0.1994	0.112	-1.777	0.076	-0.420	0.021
ZN	-0.0043	0.026	-0.166	0.868	-0.055	0.046
INDUS	-0.0772	0.043	-1.814	0.070	-0.161	0.006
CHAS	1.6087	0.685	2.348	0.019	0.262	2.955
NOX	-3.8668	2.795	-1.384	0.167	-9.358	1.624
RM	3.1388	0.394	7.969	0.000	2.365	3.913
AGE	-0.0142	0.010	-1.489	0.137	-0.033	0.005
PTRATIO	-0.4430	0.101	-4.399	0.000	-0.641	-0.245
BK	0.0137	0.014	0.982	0.327	-0.014	0.041
LSTAT	-0.3441	0.039	-8.925	0.000	-0.420	-0.268

Omnibus:	11.662	**Durbin-Watson:**	1.348
Prob(Omnibus):	0.003	**Jarque-Bera (JB):**	15.489
Skew:	0.223	**Prob(JB):**	0.000433
Kurtosis:	3.732	**Cond. No.**	1.48e+04

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 1.48e+04. This might indicate that there are strong multicollinearity or other numerical problems.

Model 4

In [23]:
```
import statsmodels.formula.api as smf
ols_model_1 = smf.ols(formula= f'PRICE ~ CRIM+ZN+INDUS+CHAS+RM+AGE+PTRATIO+BK+LST
ols_model_1.summary()
```

Out[23]:

OLS Regression Results

Dep. Variable:	PRICE	**R-squared:**	0.633
Model:	OLS	**Adj. R-squared:**	0.626
Method:	Least Squares	**F-statistic:**	95.07
Date:	Sat, 18 Jun 2022	**Prob (F-statistic):**	4.62e-102
Time:	14:10:45	**Log-Likelihood:**	-1386.9
No. Observations:	506	**AIC:**	2794.
Df Residuals:	496	**BIC:**	2836.
Df Model:	9		
Covariance Type:	nonrobust		

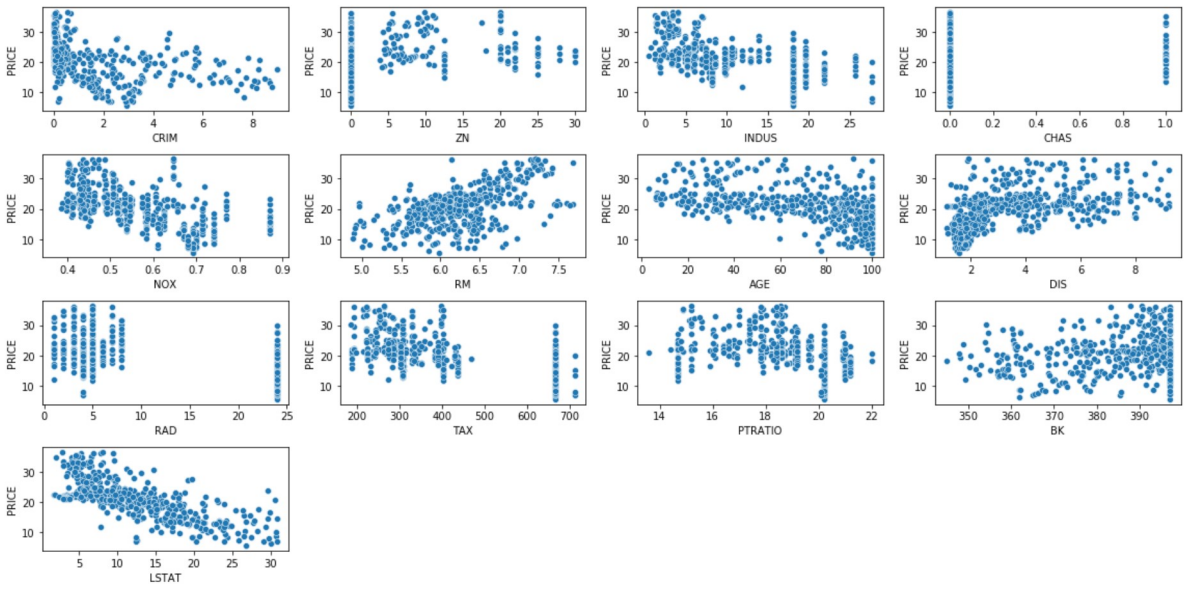
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	10.0521	6.013	1.672	0.095	-1.763	21.867
CRIM	-0.2513	0.106	-2.374	0.018	-0.459	-0.043
ZN	-0.0028	0.026	-0.108	0.914	-0.053	0.048
INDUS	-0.0999	0.039	-2.543	0.011	-0.177	-0.023
CHAS	1.5972	0.686	2.329	0.020	0.250	2.945
RM	3.1452	0.394	7.979	0.000	2.371	3.920
AGE	-0.0198	0.009	-2.302	0.022	-0.037	-0.003
PTRATIO	-0.4189	0.099	-4.219	0.000	-0.614	-0.224
BK	0.0155	0.014	1.114	0.266	-0.012	0.043
LSTAT	-0.3500	0.038	-9.124	0.000	-0.425	-0.275

Omnibus:	9.611	**Durbin-Watson:**	1.347
Prob(Omnibus):	0.008	**Jarque-Bera (JB):**	12.822
Skew:	0.179	**Prob(JB):**	0.00164
Kurtosis:	3.693	**Cond. No.**	1.40e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.4e+04. This might indicate that there are

strong multicollinearity or other numerical problems.

In [24]:
```
plt.figure(figsize=(16,8))
plotnumber = 1
for columns in data_df1.drop('PRICE', axis = 1):
    if plotnumber<=16:
        ax = plt.subplot(4,4,plotnumber)
        sns.scatterplot(x=columns, y = 'PRICE', data = data_df1)
        plt.xlabel(columns,fontsize=10)
    plotnumber+=1
plt.tight_layout()
```



Choosing Model 1 for Development Purpose.

In [25]:
```
from sklearn.model_selection import train_test_split
scaler = StandardScaler()
x_data = scaler.fit_transform(ind_var)
x_train,x_test,y_train,y_test = train_test_split(x_data,dep_var,test_size = 0.25,r
```

In [42]:
```
x_data.shape
```

Out[42]: (506, 13)

In [39]:
```
ind_var.shape
```

Out[39]: (506, 13)

In [26]:
```
from sklearn.linear_model import LinearRegression
linear_reg = LinearRegression()
regression_line = linear_reg.fit(x_train,y_train)
```

Traning Accuracy

In [27]:
```
linear_reg.score(x_train,y_train)
```

Out[27]: 0.649482634297512

Testing Accuracy

In [28]:
```
linear_reg.score(x_test,y_test)
```

Out[28]: 0.6590753281521365

Here Testing accuracy is more than Traning Acuracy. So, our model is not overfitting.

Moving Forward with saving the model and deployment.

Saving File

In [29]:
```
file_name = 'Boston_data.pickle'
pickle.dump(linear_reg,open(file_name,'wb'))
```

In [30]:
```
a = "0.03237    0.0     2.18    0.0     0.458   6.998   45.8    6.0622  3.0     2
a = a.split()
a = [float(i) for i in a]
```

In [31]:
```
model_file = pickle.load(open(file_name,'rb'))
model_file.predict(scaler.transform([a]))
```

Out[31]: array([27.67242328])