Assignment-4 -Data Analytics 1 _ Linear Regression

Kaustubh Shrikant Kabra

ERP Number :- 38

TE Comp 1

1. Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset.
2. The objective is to predict the value of prices of the house using the given features.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [2]:
```
boston=pd.read_csv('boston.csv')
```

In [3]:
```
boston.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MED
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.

In [4]:
```
boston.shape
```

Out[4]:
```
(506, 14)
```

Input features in order: 1) CRIM: per capita crime rate by town

2) ZN: proportion of residential land zoned for lots over 25,000 sq.ft.

3) INDUS: proportion of non-retail business acres per town

4) CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise)

5) NOX: nitric oxides concentration (parts per 10 million) [parts/10M]

6) RM: average number of rooms per dwelling

7) AGE: proportion of owner-occupied units built prior to 1940

8) DIS: weighted distances to five Boston employment centres

9) RAD: index of accessibility to radial highways

10) TAX: full-value property-tax rate per $10,000[$/10k]

11) PTRATIO: pupil-teacher ratio by town

12) B: The result of the equation B=1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town

13) LSTAT: % lower status of the population

Output variable:

1) MEDV: Median value of owner-occupied homes in $1000's[k$]

In [5]:

```
boston.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    int64  
 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    int64  
 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  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

In [6]:

```
boston.describe()
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
count	506.000000	506.000000	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	68.574901	3.795043
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500

In [7]:
```
boston.isnull().sum()
```

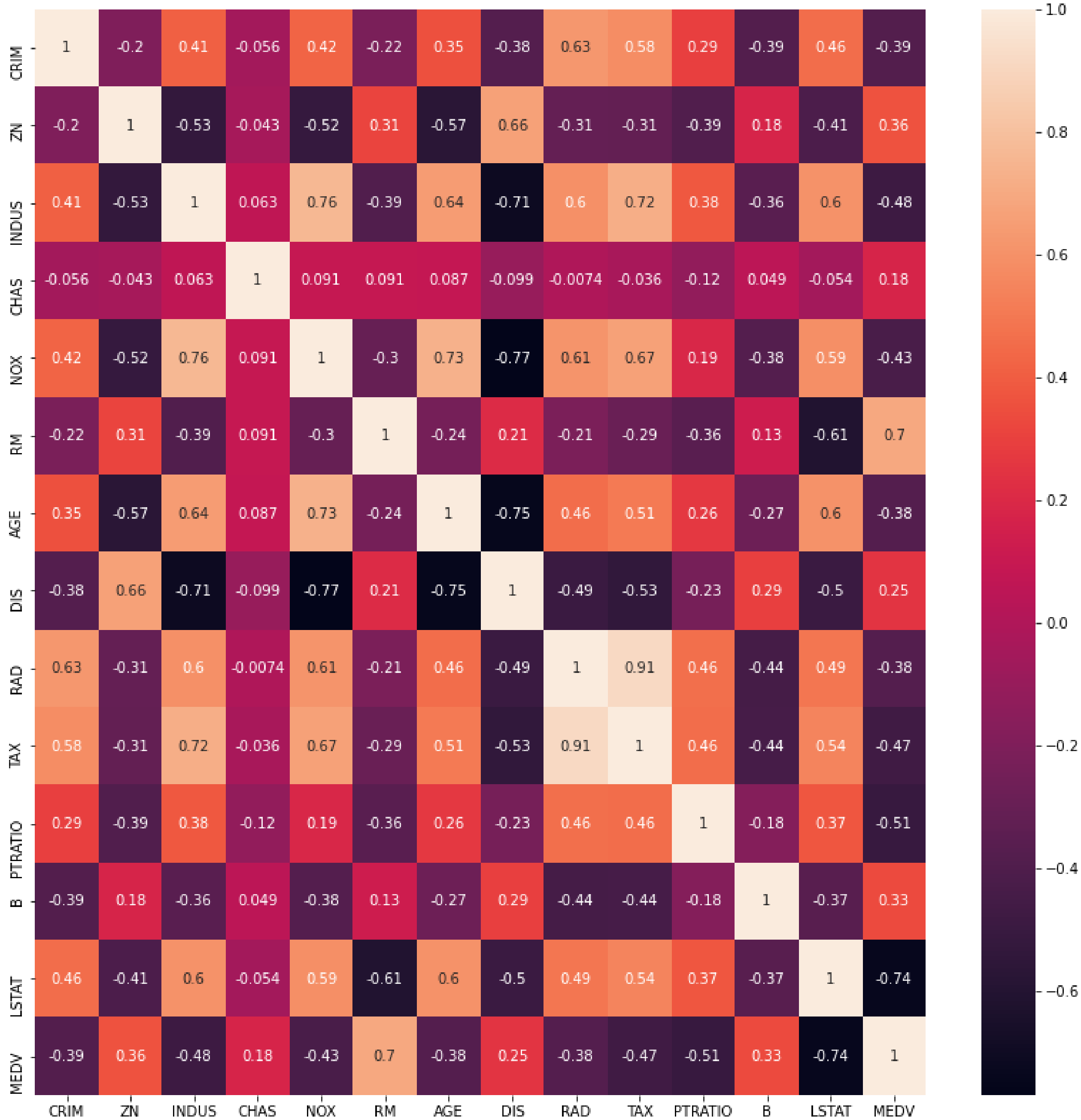
Out[7]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

In [8]:
```
boston.duplicated().sum()
```

Out[8]:
```
0
```

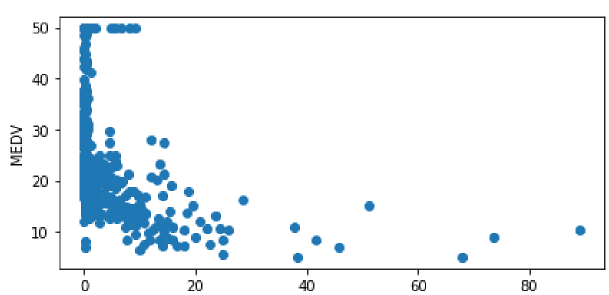
In [9]:
```
corr_m=boston.corr()
plt.figure(figsize=(14,14))
sns.heatmap(data=corr_m, annot=True)
```

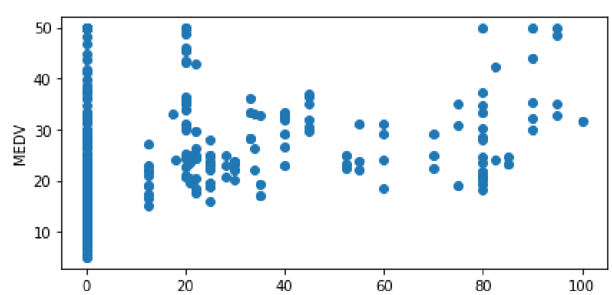
Out[9]:
```
<AxesSubplot:>
```

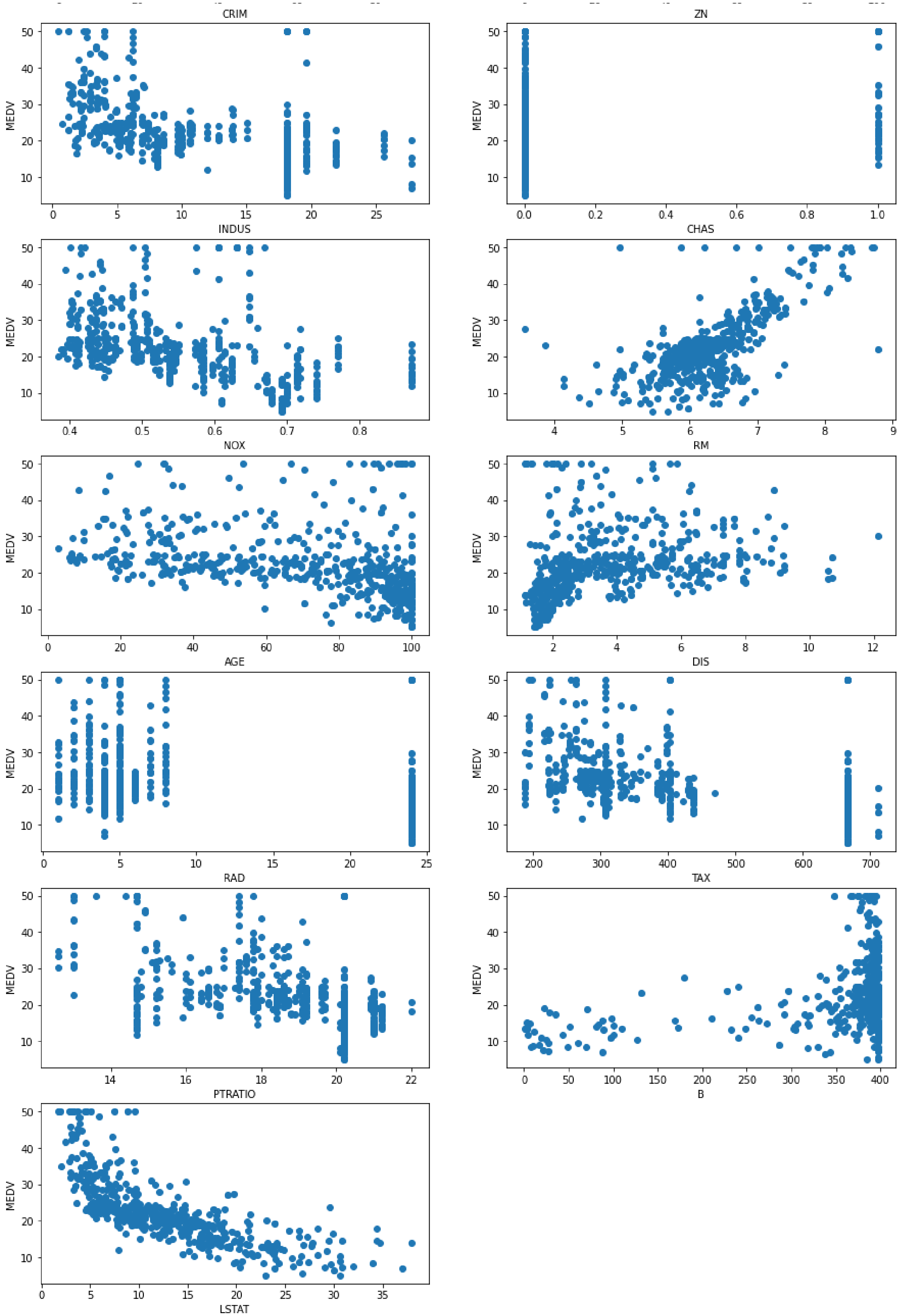


In [10]:

```
plt.figure(figsize=(15,50))
features=boston.columns[:-1]
target=boston.MEDV
for i, column in enumerate(features):
    plt.subplot(len(features),2,i+1)
    plt.scatter(x=boston[column], y=target, marker='o')
    plt.xlabel(column)
    plt.ylabel('MEDV')
```







In [11]:

```
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
# mean at 0 and std at 1
```

In [12]:

```
df=sc.fit_transform(boston)
```

In [13]:

```
df=pd.DataFrame(df)
df.head()
```

Out[13]:

	0	1	2	3	4	5	6	7	8	
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982843	-0.666
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867883	-0.987
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867883	-0.987
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752922	-1.106
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752922	-1.106

In [14]:

```
df.columns=boston.columns
```

In [15]:

```
df.head()
```

Out[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	T
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982843	-0.666
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867883	-0.987
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867883	-0.987
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752922	-1.106
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752922	-1.106

In [16]:

```
df.describe()
```

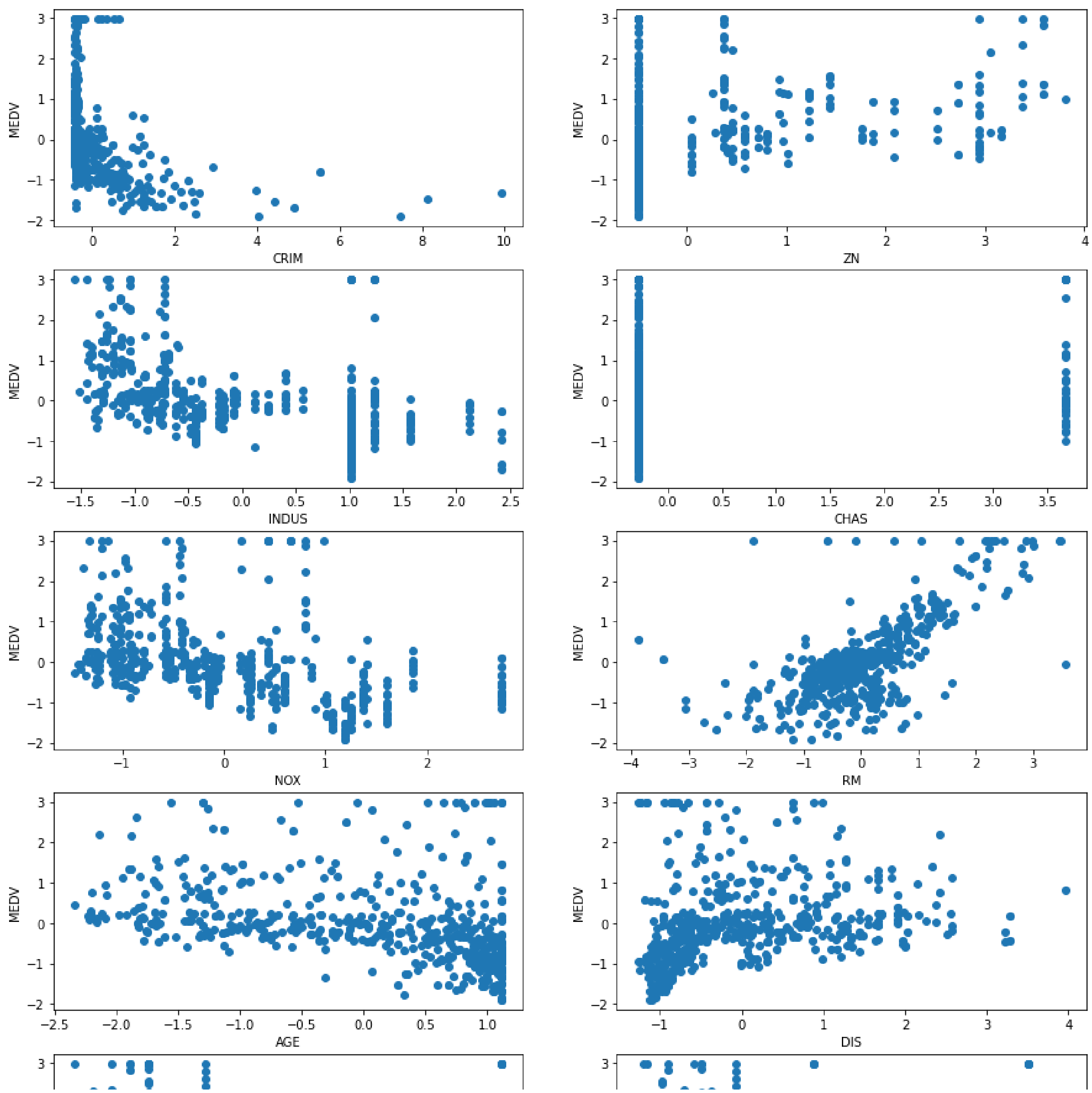
Out[16]:

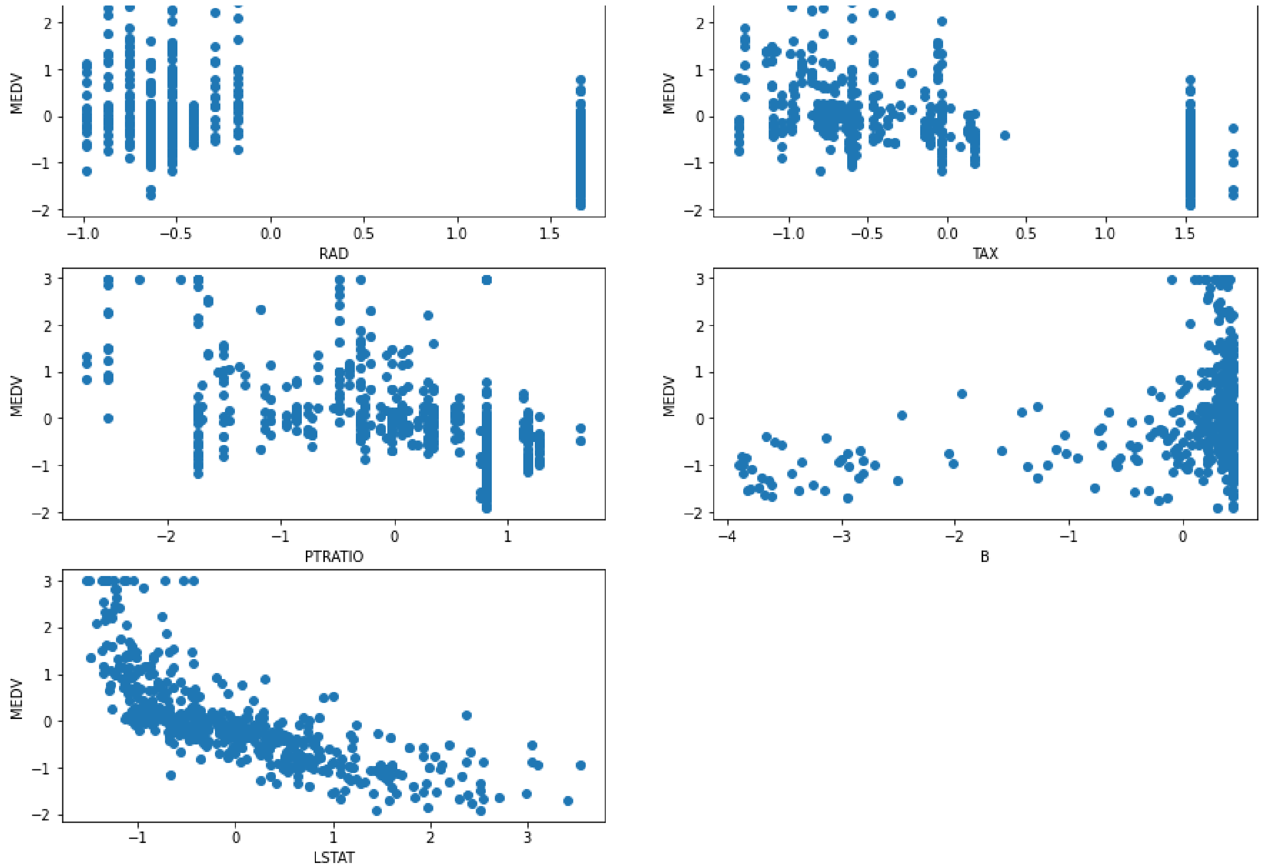
	CRIM	ZN	INDUS	CHAS	NOX	RM	
count	5.060000e+02	5.060000e+02	5.060000e+02	5.060000e+02	5.060000e+02	5.060000e+02	5.0600
mean	-8.513173e-17	3.306534e-16	2.804081e-16	-3.100287e-16	-8.071058e-16	-5.189086e-17	-2.650
std	1.000990e+00	1.000990e+00	1.000990e+00	1.000990e+00	1.000990e+00	1.000990e+00	1.0009
min	-4.197819e-01	-4.877224e-01	-1.557842e+00	-2.725986e-01	-1.465882e+00	-3.880249e+00	-2.3354
25%	-4.109696e-01	-4.877224e-01	-8.676906e-01	-2.725986e-01	-9.130288e-01	-5.686303e-01	-8.374

	CRIM	ZN	INDUS	CHAS	NOX	RM	
50%	-3.906665e-01	-4.877224e-01	-2.110985e-01	-2.725986e-01	-1.442174e-01	-1.084655e-01	3.1738
75%	7.396560e-03	4.877224e-02	1.015999e+00	-2.725986e-01	5.986790e-01	4.827678e-01	9.0679
max	9.933931e+00	3.804234e+00	2.422565e+00	3.668398e+00	2.732346e+00	3.555044e+00	1.1174

In [17]:

```
plt.figure(figsize=(15,50))
features=df.columns[:-1]
target=df.MEDV
for i, column in enumerate(features):
    plt.subplot(len(features),2,i+1)
    plt.scatter(x=df[column], y=target, marker='o')
    plt.xlabel(column)
    plt.ylabel('MEDV')
```





OBSERVATIONS

- **Variables LSTAT and RM have a hi correlation with the price of the house.**
- **INDUS-TAX, INDUS-DIS, INDUS-NOX, DIS-NOX , AGE-NOX, all these pairs have high correlation between them.**

In [18]:

```
X=boston[['LSTAT','RM']]
Y = boston[['MEDV']]
X.head()
```

Out[18]:

	LSTAT	RM
0	4.98	6.575
1	9.14	6.421
2	4.03	7.185
3	2.94	6.998
4	5.33	7.147

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 = 0.15, random_stat
# test_size=0.15 means 15% will be for test data
# random_state
print(X_train.shape)
print(X_test.shape)
```

```
print(Y_train.shape)
print(Y_test.shape)
```

```
(430, 2)
(76, 2)
(430, 1)
(76, 1)
```

Linear Regression

```
   Linear Regression is a machine learning algorithm based on supervised
learning. It performs a regression task. Regression models a target
prediction value based on independent variables. It is mostly used for
finding out the relationship between variables and forecasting. Different
regression models differ based on – the kind of relationship between
dependent and independent variables they are considering, and the number
of independent variables getting used.<br>
    Linear regression performs the task to predict a dependent variable
value (y) based on a given independent variable (x). So, this regression
technique finds out a linear relationship between x (input) and
y(output). Hence, the name is Linear Regression.
```

$$Y = \theta_1 + X\theta_0 + \epsilon$$

```
While training the model we are given :<br>
x: input training data (univariate – one input variable(parameter))<br>
y: labels to data (supervised learning)<br>

When training the model – it fits the best line to predict the value of y
for a given value of x. The model gets the best regression fit line by
finding the best θ1 and θ2 values.<br>
θ1: intercept<br>
θ2: coefficient of x
e:error term
```

Training Model

In [20]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lin_model = LinearRegression()  # Make an instance of the model
lin_model.fit(X_train, Y_train)
```

Out[20]:

```
LinearRegression()
```

In [21]:

```
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))

print("The model performance for training set")
print('RMSE is {}'.format(rmse))
print("\n")

# on testing set
y_test_predict = lin_model.predict(X_test)
```

```
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))

print("The model performance for testing set")
print('RMSE is {}'.format(rmse))

print(lin_model.coef_.ravel())
print(lin_model.intercept_)
```

```
The model performance for training set
RMSE is 5.596970449422867


The model performance for testing set
RMSE is 5.178451251951529
[-0.70376468  4.88802288]
[0.68155642]
```

In [22]:

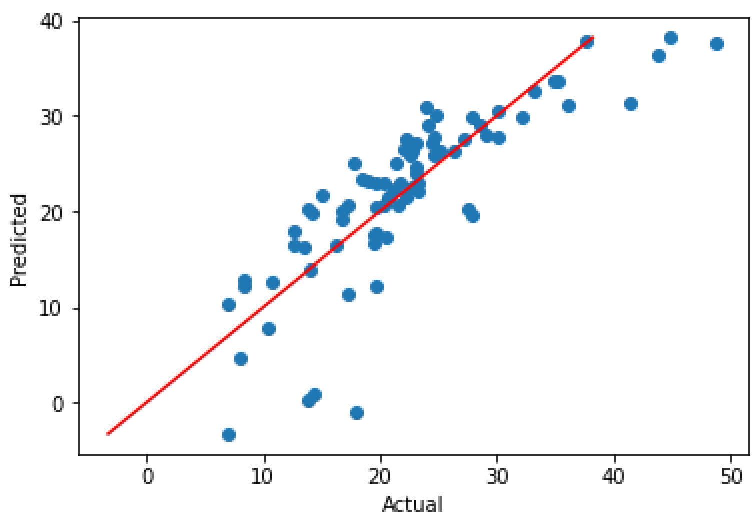
```
plt.scatter(Y_test, y_test_predict)

plt.plot([min(y_test_predict),max(y_test_predict)],[min(y_test_predict),max(y_test_pred
# Ploting a straight line y = x (red in color)
# For the 100% perfect fit, Predicted values will be same as Actual value
# That means for the curve below, y = x line represent 100% fit.

plt.xlabel('Actual')
plt.ylabel('Predicted')
```

Out[22]:

```
Text(0, 0.5, 'Predicted')
```



The red line shown represents y=x line (fit with 100% accuracy).

> **Now let's try to fit the linear regression model using all the variables**

In [23]:

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

In [24]:

```
X=df.drop(labels='MEDV', axis=1)
X.head()
```

Out[24]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982843
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867883
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867883
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752922
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752922

In [25]:

```
X.shape
```

Out[25]:

```
(506, 13)
```

In [26]:

```
Y=df.MEDV
```

In [27]:

```
x_train,x_test,y_train,y_test=train_test_split(X, Y, test_size=0.3, random_state=2)
```

In [28]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [29]:

```
lin_model = LinearRegression()
lin_model.fit(x_train, y_train)
```

Out[29]:

```
LinearRegression()
```

In [30]:

```
y_train_predict = lin_model.predict(x_train)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))

print("The model performance for training set")
print('RMSE is {}'.format(rmse))
print("\n")

# on testing set
y_test_predict = lin_model.predict(x_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))

print("The model performance for testing set")
print('RMSE is {}'.format(rmse))

print(lin_model.coef_.ravel())
print(lin_model.intercept_)
```

```
The model performance for training set
RMSE is 0.5112786395135811


The model performance for testing set
RMSE is 0.5224064330994048
```

```
[-0.08725644  0.07895821 -0.01355751  0.08825089 -0.1903045   0.2674622
  0.05808119 -0.28974664  0.30499803 -0.20057974 -0.25649371  0.12447486
 -0.47178377]
0.0076784214248864485
```

In [31]:

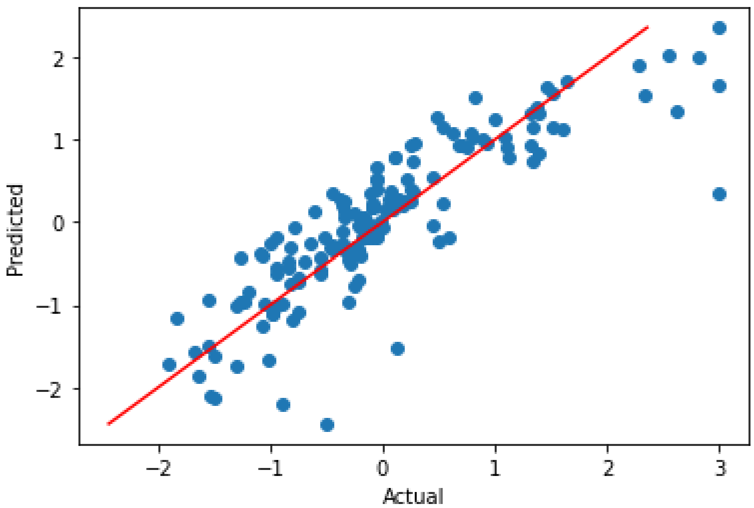
```python
plt.scatter(y_test, y_test_predict)

plt.plot([min(y_test_predict),max(y_test_predict)],[min(y_test_predict),max(y_test_pred
# Ploting a straight line y = x (red in color)
# For the 100% perfect fit, Predicted values will be same as Actual value
# That means for the curve below, y = x line represent 100% fit.

plt.xlabel('Actual')
plt.ylabel('Predicted')
```

Out[31]:

```
Text(0, 0.5, 'Predicted')
```



In [ ]: