ASSIGNMENT 4

Data Analytics I

In [7]:

```
import numpy as np
import pandas as pd
from sklearn import linear_model
from sklearn.model_selection import train_test_split
```

In [8]:

```
#load the Boston Housing Dataset and print
from sklearn.datasets import load_boston
boston = load_boston()
print(boston)
boston.keys()
```

```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690
e+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]]), '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]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
 'DIS', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'), 'DESCR': ".. _boston_d
 ataset:\n\nBoston house prices dataset\n---------------------------\n\n**Dat
 a Set Characteristics:**  \n\n    :Number of Instances: 506 \n\n    :Number
 of Attributes: 13 numeric/categorical predictive. Median Value (attribute 1
 4) is usually the target.\n\n    :Attribute Information (in order):\n
 - CRIM     per capita crime rate by town\n        - ZN       proportion of r
 esidential land zoned for lots over 25,000 sq.ft.\n        - INDUS    propor
 tion 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       avera
 ge number of rooms per dwelling\n        - AGE      proportion of owner-occu
 pied units built prior to 1940\n        - DIS      weighted distances to fiv
 e Boston employment centres\n        - RAD      index of accessibility to ra
 dial 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 sta
 tus of the population\n        - MEDV     Median value of owner-occupied hom
 es in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harris
 on, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhtt
 ps://archive.ics.uci.edu/ml/machine-learning-databases/housing/\n\n\nThis da
 taset was taken from the StatLib library which is maintained at Carnegie Mel
 lon University.\n\nThe Boston house-price data of Harrison, D. and Rubinfel
 d, D.L. 'Hedonic\nprices and the demand for clean air', J. Environ. Economic
 s & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regr
 ession 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 d
 ata has been used in many machine learning papers that address regression\np
 roblems.   \n     \n.. topic:: References\n\n   - Belsley, Kuh & Welsch, 'Re
 gression diagnostics: Identifying Influential Data and Sources of Collineari
 ty', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining Instance-Based
 and Model-Based Learning. In Proceedings on the Tenth International Conferen
 ce of Machine Learning, 236-243, University of Massachusetts, Amherst. Morga
 n Kaufmann.\n", 'filename': 'C:\\Users\\Atharv Karanjkar\\anaconda3\\lib\\si
 te-packages\\sklearn\\datasets\\data\\boston_house_prices.csv'}
```

Out[8]:

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [9]:

```
# Transform dataset into dataframe
# Date: The data that we want or the independent variable also known as x vvalues
# feature_names: The columns names of the data
# Target: Target variable or prices of the houses or dependent variable or y values
df_x = pd.DataFrame(boston.data, columns = boston.feature_names)
print(df_x)
df_y = pd.DataFrame(boston.target)
print(df_y)
#get some statistics from data
df_x.describe()
```

```
        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
..       ...   ...    ...   ...    ...    ...   ...     ...  ...    ...
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786  1.0  273.0
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875  1.0  273.0
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675  1.0  273.0
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889  1.0  273.0
505  0.04741   0.0  11.93   0.0  0.573  6.030  80.8  2.5050  1.0  273.0

     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
..       ...     ...    ...
501     21.0  391.99   9.67
502     21.0  396.90   9.08
503     21.0  396.90   5.64
504     21.0  393.45   6.48
505     21.0  396.90   7.88

[506 rows x 13 columns]
        0
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
..    ...
501  22.4
502  20.6
503  23.9
504  22.0
505  11.9

[506 rows x 1 columns]
```

Out[9]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000

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

In [10]:

```
# Initialize the linear regression model
reg = linear_model.LinearRegression()
# Splitting of datasets into 70% training and 30% testing dta
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size = 0.30, random_st
# Train the model with training data
a = reg.fit(x_train, y_train)
print(a)
```

```
LinearRegression()
```

In [11]:

```
# Print coefficients/weights for each feature/column of our model
print(reg.coef_)
```

```
[[-1.02065294e-01  3.92035307e-02 -6.13494400e-02  3.48084703e+00
  -1.74598953e+01  3.66444175e+00 -5.31304197e-03 -1.37067900e+00
   2.51447673e-01 -9.43832755e-03 -8.58133141e-01  6.78308990e-03
  -4.96519703e-01]]
```

In [12]:

```
y_pred = reg.predict(x_test)
print(y_pred)
```

```
[[21.90897572]
 [32.36829283]
 [ 9.38919345]
 [16.40673353]
 [17.80964232]
 [31.83838312]
 [25.10363218]
 [15.4942598 ]
 [21.82825591]
 [-3.63190569]
 [26.12960431]
 [15.57300292]
 [ 5.61225053]
 [ 5.58756072]
 [25.41154332]
 [34.70503462]
 [26.17912943]
 [19.13532445]
 [23.91967422]
```

In [13]:

```
# Print the actual values
print(y_test)
```

```
        0
358  22.7
197  30.3
48   14.4
450  13.4
469  20.1
..    ...
212  22.4
133  18.4
279  35.1
274  32.4
23   14.5

[152 rows x 1 columns]
```

In [14]:

```
# Performance matrix and accuracy using mean square error (MSE)
print(np.mean((y_pred - y_test)**2))
```

```
0    31.829631
dtype: float64
```

Performance matrix and accuracy using mean square error (MSE) and sklearn.metrics

In [15]:

```
# Load the Boston Housing DataSet from scikit-learn
from sklearn.datasets import load_boston
boston_dataset = load_boston()
# boston_dataset is a dictionary
# let's check what it contains
boston_dataset.keys()
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
boston.head()
```

Out[15]:

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

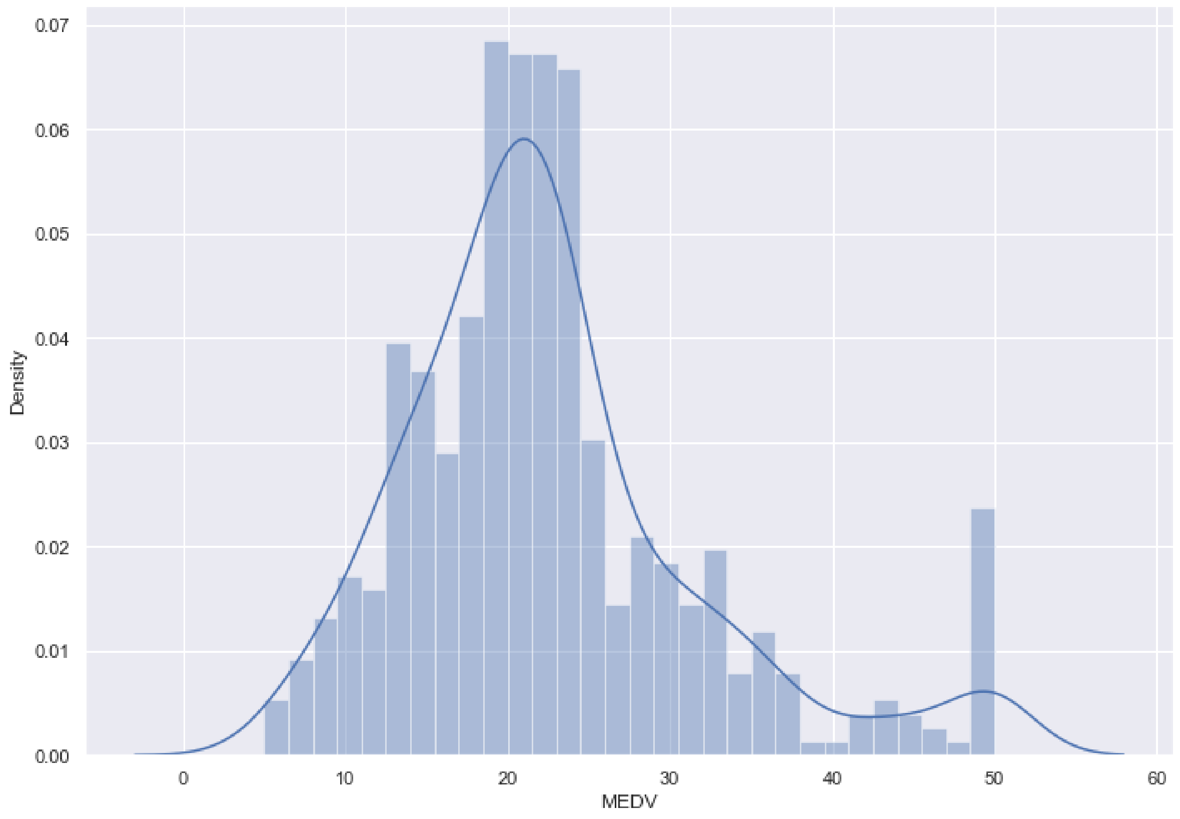
In [16]:

```
# The target values is missing from the data. Create a new column of target values and add
boston['MEDV'] = boston_dataset.target
```

In [17]:

```
# Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
# set the size of the figure
sns.set(rc={'figure.figsize':(11.7,8.27)})
# plot a histogram showing the distribution of the target values
sns.distplot(boston['MEDV'], bins=30)
plt.show()
```

```
C:\Users\Atharv Karanjkar\anaconda3\lib\site-packages\seaborn\distributions.
py:2557: FutureWarning: `distplot` is a deprecated function and will be remo
ved in a future version. Please adapt your code to use either `displot` (a f
igure-level function with similar flexibility) or `histplot` (an axes-level
function for histograms).
  warnings.warn(msg, FutureWarning)
```

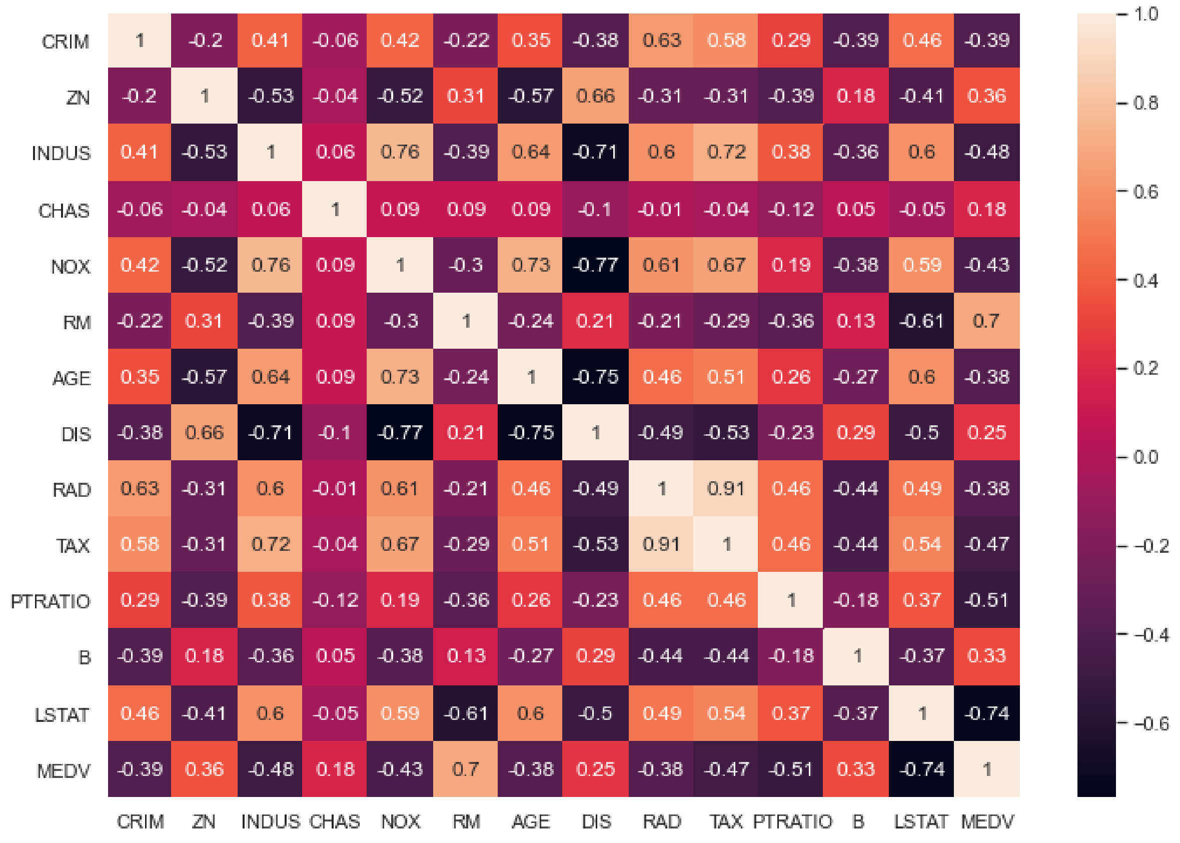


In [18]:

```
# Correlation matrix
# compute the pair wise correlation for all columns
correlation_matrix = boston.corr().round(2)
# use the heatmap function from seaborn to plot the correlation matrix
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

Out[18]:

```
<AxesSubplot:>
```

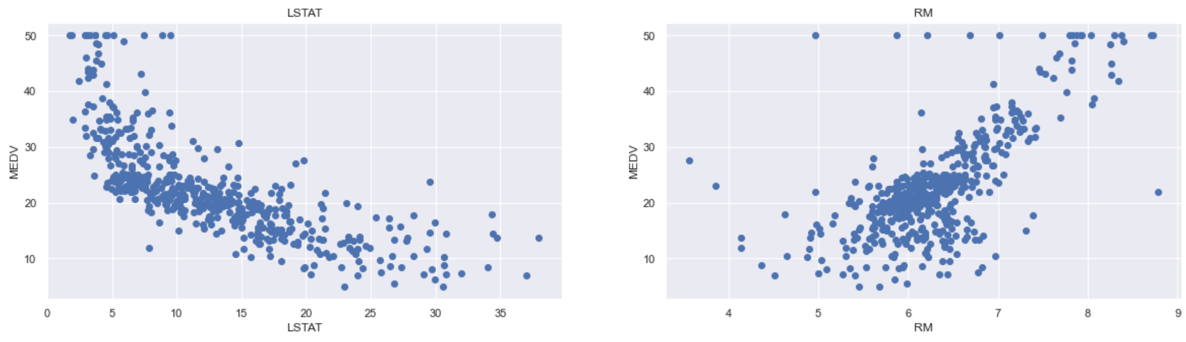


	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
CRIM	1	-0.2	0.41	-0.06	0.42	-0.22	0.35	-0.38	0.63	0.58	0.29	-0.39	0.46	-0.39
ZN	-0.2	1	-0.53	-0.04	-0.52	0.31	-0.57	0.66	-0.31	-0.31	-0.39	0.18	-0.41	0.36
INDUS	0.41	-0.53	1	0.06	0.76	-0.39	0.64	-0.71	0.6	0.72	0.38	-0.36	0.6	-0.48
CHAS	-0.06	-0.04	0.06	1	0.09	0.09	0.09	-0.1	-0.01	-0.04	-0.12	0.05	-0.05	0.18
NOX	0.42	-0.52	0.76	0.09	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	0.09	-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	0.09	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	-0.1	-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	-0.01	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	-0.04	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	0.05	-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	-0.05	0.59	-0.61	0.6	-0.5	0.49	0.54	0.37	-0.37	1	-0.74
MEDV	-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 [19]:

```
plt.figure(figsize=(20, 5))
features = ['LSTAT', 'RM']
target = boston['MEDV']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = boston[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```



In [20]:

```
# Prepare the data for training

X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT','RM'])
Y = boston['MEDV']
```

In [21]:

```
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
# splits the training and test data set in 80% : 20%
# assign random_state to any value.This ensures consistency.
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(404, 2)
(102, 2)
(404,)
(102,)
```

In [22]:

```
# Train the model using sklearn LinearRegression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

Out[22]:

```
LinearRegression()
```

In [23]:

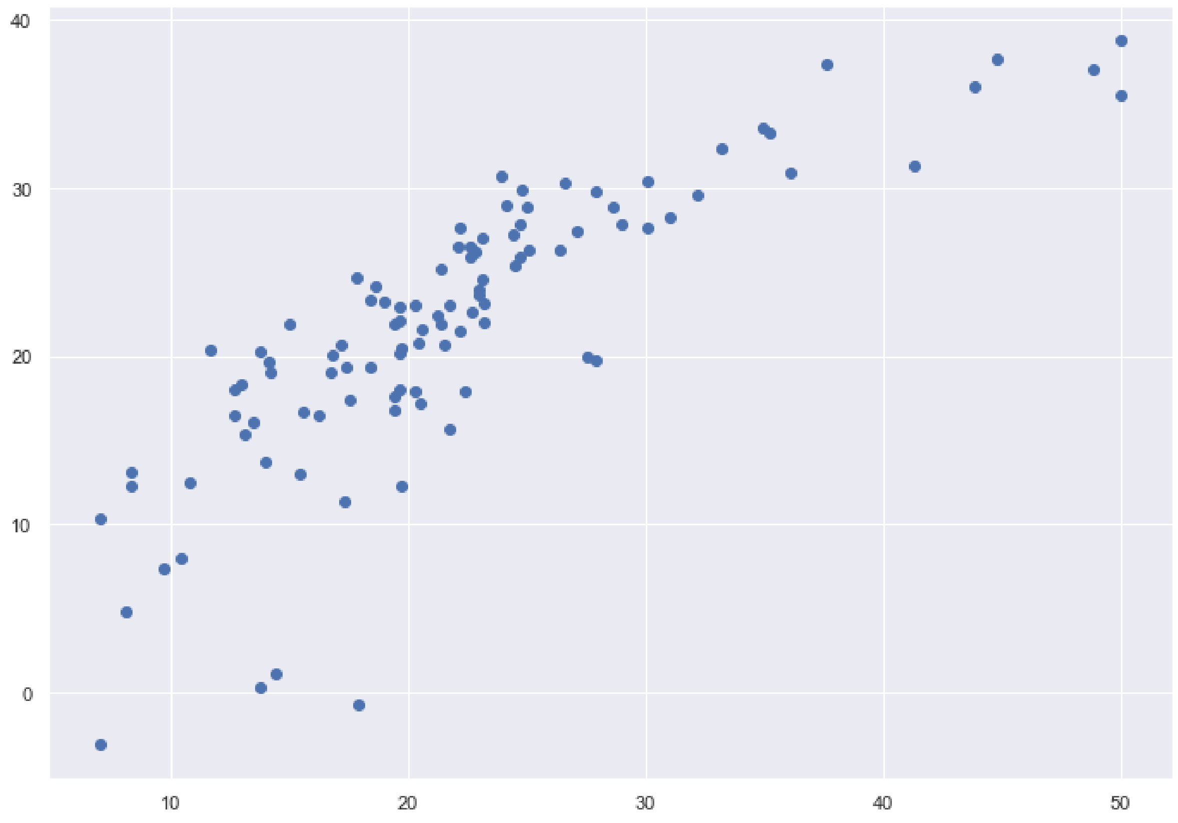
```
# model evaluation for training set
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)
print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
# Model evaluation for testing set
y_test_predict = lin_model.predict(X_test)
# root mean square error of the model
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
# r-squared score of the model
r2 = r2_score(Y_test, y_test_predict)
print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 5.6371293350711955
R2 score is 0.6300745149331701


The model performance for testing set
--------------------------------------
RMSE is 5.137400784702911
R2 score is 0.6628996975186953
```

In [24]:

```
# plotting the y_test vs y_pred
# ideally should have been a straight line
plt.scatter(Y_test, y_test_predict)
plt.show()
```


40
30
20
10
0
10
20
30
40
50