Linear Regression on Boston Housing Dataset

This data was originally a part of UCI Machine Learning Repository and has been removed now. This data also ships with the scikit-learn library. There are 506 samples and 13 feature variables in this data-set. The objective is to predict the value of prices of the house using the given features.

The description of all the features is given below:

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 oxide 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 centers

RAD: Index of accessibility to radial highways

TAX: Full-value property tax rate per $10,000

B: $1000(Bk - 0.63)^2$, where Bk is the proportion of [people of African American descent] by town

LSTAT: Percentage of lower status of the population

MEDV: Median value of owner-occupied homes in $1000s

Import the required Libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

import pandas as pd
import seaborn as sns

%matplotlib inline
```

Load the Boston Housing DataSet from scikit-learn

In [2]:
```
from sklearn.datasets import load_boston

boston_dataset = load_boston()
```

```
# boston_dataset is a dictionary
# let's check what it contains
boston_dataset.keys()
```

```
C:\Users\gptkgf\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: Futur
eWarning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 a
nd 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)
```

Out[2]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename', 'data_module'])
```

Load the data into pandas dataframe

In [3]:
```
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
boston.head()
```

Out[3]:

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

In [4]:
```
boston.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 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
dtypes: float64(13)
memory usage: 51.5 KB
```

The target values is missing from the data. Create a new column of target values and add it to dataframe

In [5]:
```
boston['PRICE'] = boston_dataset.target
```

In [6]:
```
boston
```

Out[6]:

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

506 rows × 14 columns

Data preprocessing

In [5]:
```
# check for missing values in all the columns
boston.isnull().sum()
```

Out[5]:
```
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
```

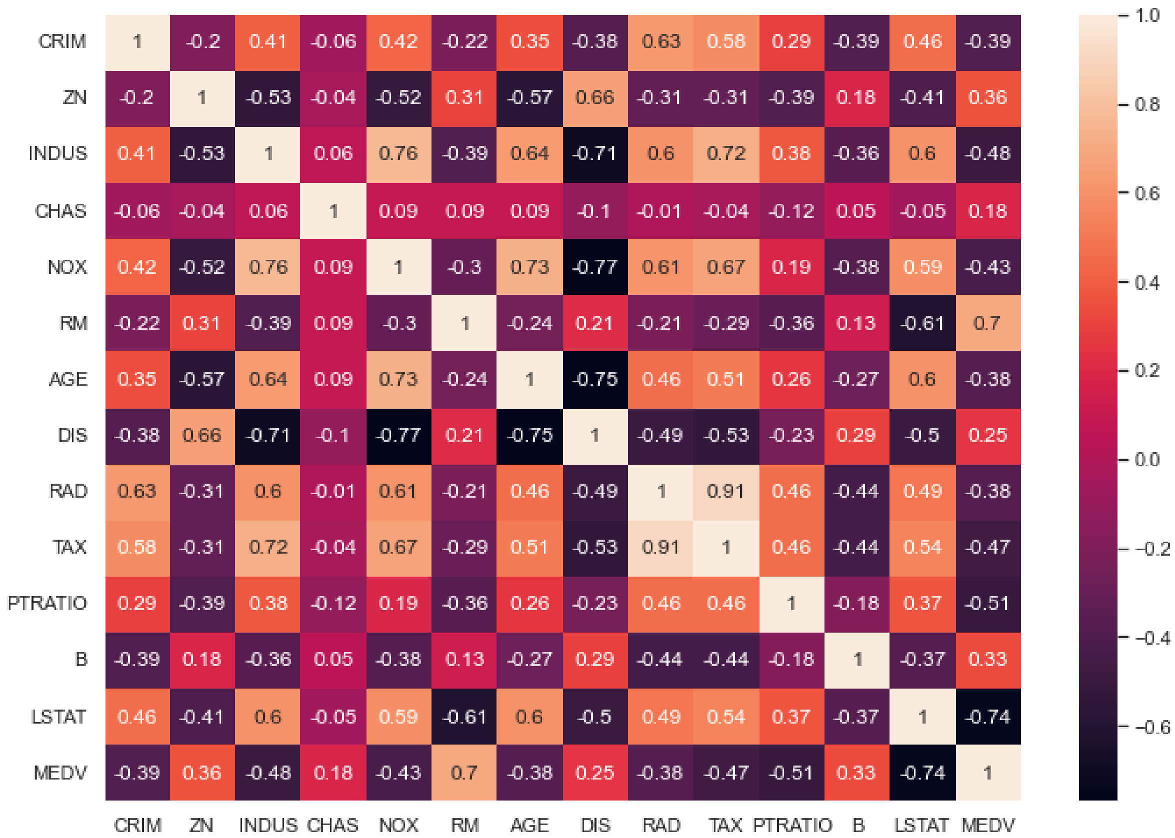
Data Visualization

Correlation matrix

In [7]:
```python
# compute the pair wise correlation for all columns
correlation_matrix = boston.corr().round(2)
```

In [8]:
```python
# 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[8]:
```
<AxesSubplot:>
```

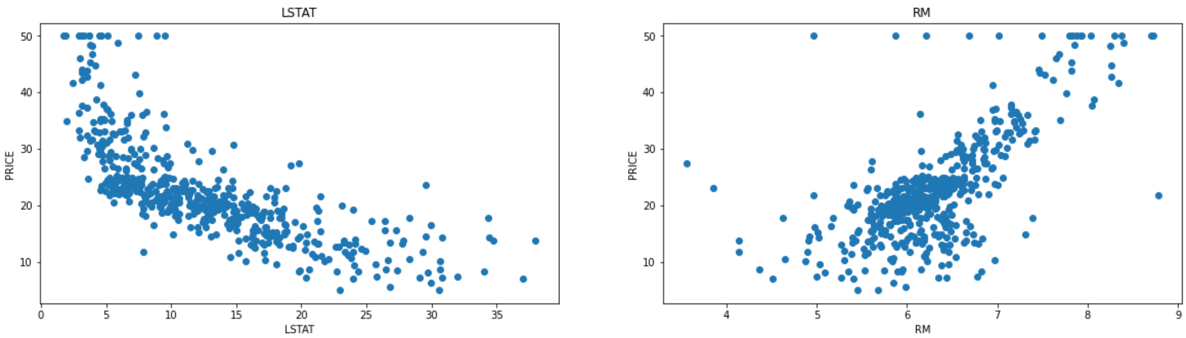


Observations

- From the above coorelation plot we can see that **PRICE** is strongly correlated to **LSTAT**, **RM**

- **RAD** and **TAX** are stronly correlated, so we don't include this in our features together to avoid multi-colinearity

In [7]:
```
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = boston['PRICE']

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('PRICE')
```



Prepare the data for training

In [10]:
```
X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT','RM'])
Y = boston['PRICE']
```

Split the data into training and testing sets

In [11]:
```
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_
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

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

Train the model using sklearn LinearRegression

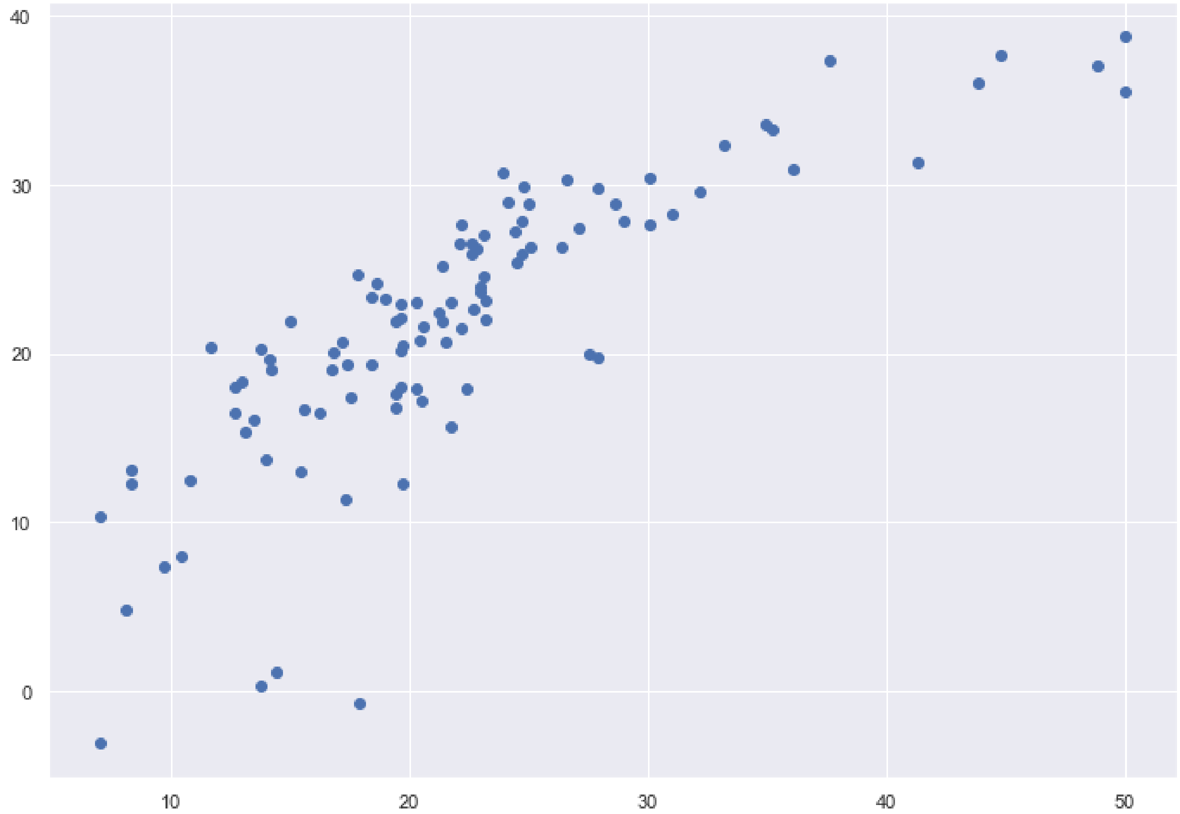
In [12]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

Out[12]:
```
LinearRegression()
```

In [13]:
```
# 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 5.13740078470291
R2 score is 0.6628996975186954
```

In [14]:
```
# plotting the y_test vs y_pred
# ideally should have been a straight line
plt.scatter(Y_test, y_test_predict)
plt.show()
```




```
In [ ]:
```

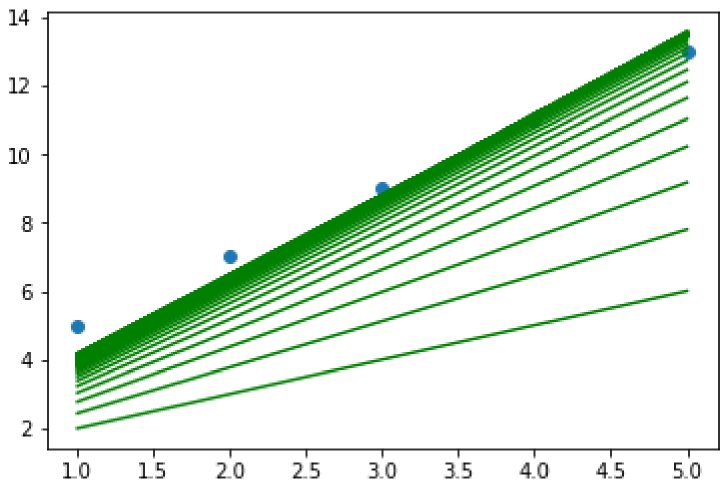
```
In [ ]:
```

In [80]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [81]:
```
%matplotlib inline
def gradient_descent(x,y):
    m = b = 1
    rate = 0.01
    n = len(x)
    plt.scatter(x,y)
    for i in range(100):
        y_predicted = m * x + b
        plt.plot(x,y_predicted,color='green')
        md = -(2/n)*sum(x*(y-y_predicted))
        yd = -(2/n)*sum(y-y_predicted)
        m = m - rate * md
        b = b - rate * yd
```

In [82]:
```
x = np.array([1,2,3,4,5])
y = np.array([5,7,9,11,13])
```

In [83]:
```
gradient_descent(x,y)
```



In [ ]:

In [1]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
datas = pd.read_csv('data.csv')
datas
```

Out[1]:

	sno	Temperature	Pressure
0	1	0	0.0002
1	2	20	0.0012
2	3	40	0.0060
3	4	60	0.0300
4	5	80	0.0900
5	6	100	0.2700

In [2]:
```
X = datas.iloc[:, 1:2].values
y = datas.iloc[:, 2].values
```

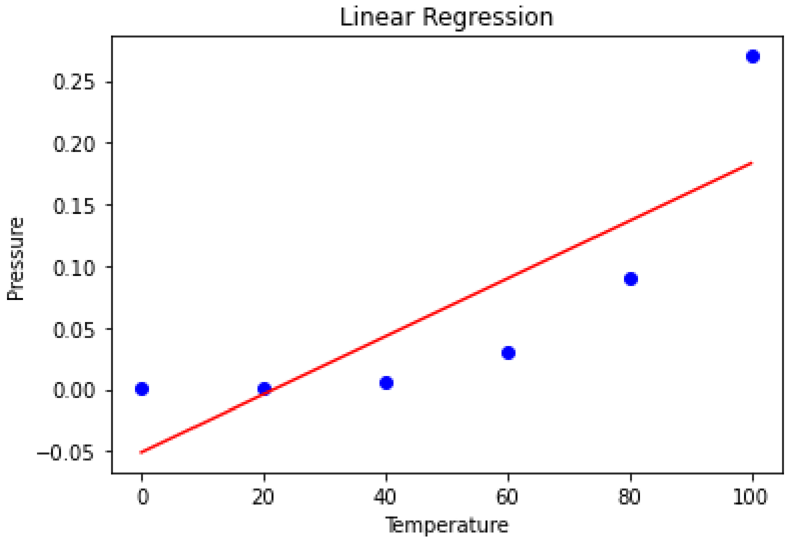
In [3]:
```
# Fitting Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin = LinearRegression()

lin.fit(X, y)
```

Out[3]: `LinearRegression()`

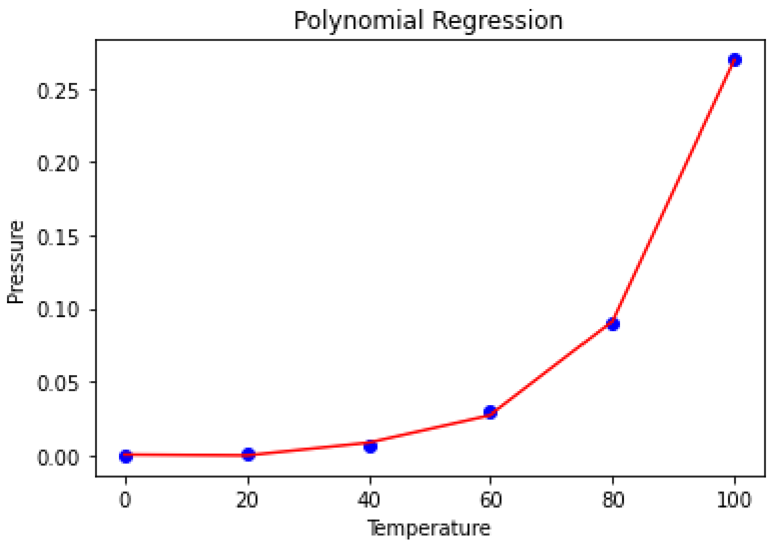
In [13]:
```
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 4)
X_poly = poly.fit_transform(X)

poly.fit(X_poly, y)
lin2 = LinearRegression()
lin2.fit(X_poly, y)
```

Out[13]: `LinearRegression()`

In [14]:
```
# Visualising the Linear Regression results
plt.scatter(X, y, color = 'blue')

plt.plot(X, lin.predict(X), color = 'red')
plt.title('Linear Regression')
plt.xlabel('Temperature')
plt.ylabel('Pressure')

plt.show()
```



In [15]:
```
# Visualising the Polynomial Regression results
plt.scatter(X, y, color = 'blue')

plt.plot(X, lin2.predict(poly.fit_transform(X)), color = 'red')
plt.title('Polynomial Regression')
plt.xlabel('Temperature')
plt.ylabel('Pressure')

plt.show()
```



In [ ]:

In [ ]:

In [1]:
```
# import packages
import pandas as pd
import numpy as np

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
```

In [2]:
```
from sklearn.model_selection import train_test_split
```

In [3]:
```
df = pd.read_csv('Advertising.csv')
df
```

Out[3]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

200 rows × 4 columns

In [4]:
```
# dropping rows which have null values
df.dropna(inplace=True,axis=0)
df
```

Out[4]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

189 rows × 4 columns

In [5]:
```
y = df['sales']
X = df.drop('sales',axis=1)
```

In [6]:
```
# splitting the dataframe into train and test sets
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=10
```

In [7]:
```
#normalizing the values in each column
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [8]:
```
lr = LinearRegression()
model = lr.fit(X_train,y_train)
```

In [9]:
```
y_pred = model.predict(X_test)
ydf = pd.DataFrame({'y_test':y_test,'y_pred':y_pred})
rslt_df = ydf.sort_values(by = 'y_test')
```

In [10]:
```
print(mean_squared_error(y_test,y_pred)) #lesser the better
```

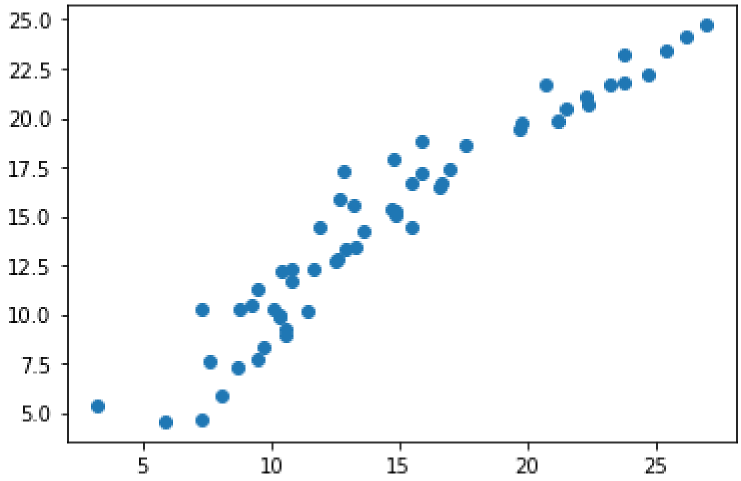
```
2.7506859249500466
```

In [11]:
```
print(r2_score(y_test, y_pred)) #high value- better model
```

```
0.9148625826187149
```

In [12]:
```
import matplotlib.pyplot as plt
plt.scatter(ydf['y_test'],ydf['y_pred'])
#it should be a straight line to satisfy normality assumption
```

Out[12]:
```
<matplotlib.collections.PathCollection at 0x2364e818b20>
```



In [15]:
```
model.coef_
```

Out[15]:
```
array([ 3.78101153,  2.58704901, -0.03067692])
```

In [16]:
```
model.intercept_
```

Out[16]:
```
13.945454545454544
```

In [ ]: