In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
df= pd.read_csv("D:/machine_learning/csv_files/student_scores.csv")
df
```

Out[2]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85
11	5.9	62
12	4.5	41
13	3.3	42
14	1.1	17
15	8.9	95
16	2.5	30
17	1.9	24
18	6.1	67
19	7.4	69
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

In [3]:

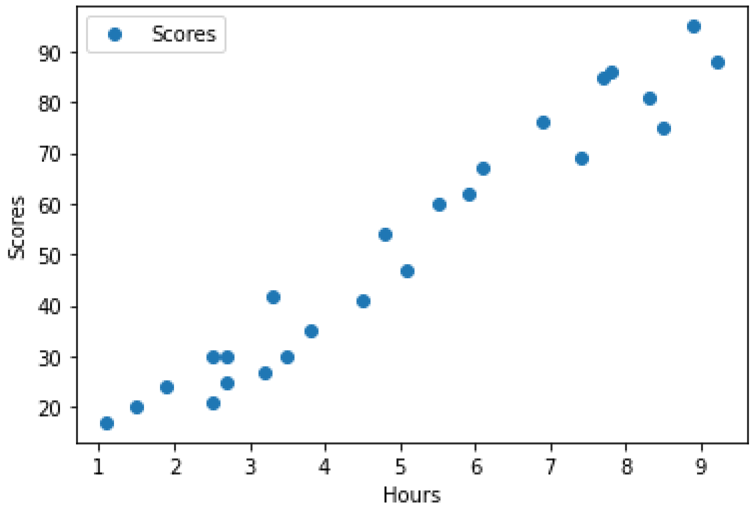
```
df.describe()
```

Out[3]:

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

In [4]:

```
%matplotlib inline
df.plot(x = 'Hours', y = 'Scores', style='o')
plt.xlabel('Hours')
plt.ylabel('Scores')
plt.show()
```



In [5]:

```
# Prepare the Data now
```

In [6]:

```
# we have to divide our dataset into label and feature.
# features are the independent variables whereas labels are dependent variables.
```

In [7]:

```
# We want to predict the percentage score depending upon the hours studied. Therefore
# our features will consist of the "Hours" column, and the label will be the "Score" column
```

In [8]:

```
X = df.iloc[:,:-1].values # features
y = df.iloc[:,1].values       # label
```

In [9]:

```
X
```

Out[9]:

```
array([[2.5],
       [5.1],
       [3.2],
       [8.5],
       [3.5],
       [1.5],
       [9.2],
       [5.5],
       [8.3],
       [2.7],
       [7.7],
       [5.9],
       [4.5],
       [3.3],
       [1.1],
       [8.9],
       [2.5],
       [1.9],
       [6.1],
       [7.4],
       [2.7],
       [4.8],
       [3.8],
       [6.9],
       [7.8]])
```

In [10]:

```
y
```

Out[10]:

```
array([21, 47, 27, 75, 30, 20, 88, 60, 81, 25, 85, 62, 41, 42, 17, 95, 30,
       24, 67, 69, 30, 54, 35, 76, 86], dtype=int64)
```

In [11]:

```
# Now that we have our features and labels, the next step is to split this data into
# training and test sets.We'll do this by using Scikit-Learn's built-in train_test_split()
```

In [12]:

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

In [13]:

```
# Now We have to split our dataset into train and test data
```

In [14]:

```
# convention is to distribute the 80% data for the training purpose and 20% data to testing
# but it's not compulsary,
```

In [15]:

```
X_train,X_test, y_train,y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

In [16]:

```
# Training the Algorithm
```

In [17]:

```
X_train
```

Out[17]:

```
array([[3.8],
       [1.9],
       [7.8],
       [6.9],
       [1.1],
       [5.1],
       [7.7],
       [3.3],
       [8.3],
       [9.2],
       [6.1],
       [3.5],
       [2.7],
       [5.5],
       [2.7],
       [8.5],
       [2.5],
       [4.8],
       [8.9],
       [4.5]])
```

In [18]:

```
from sklearn.linear_model import LinearRegression
```

In [19]:

```
# now we have to create an object of LinearRegression class
# and fit the Train data to the model
```

In [20]:

```
reg = LinearRegression()
reg.fit(X_train, y_train)
```

Out[20]:

```
LinearRegression()
```

In [21]:

```
# now we have successfully trained our data
```

In [22]:

```
# With Scikit-Learn it is extremely straight forward to implement linear regression models,
# as all you really need to do is import the LinearRegression class,
# instantiate it, and call the fit() method along with our training data.
```

In [23]:

```
# In the theory section, linear regression model basically finds the best value for the
# intercept and slope, which results in a line that best fits the data.
```

In [24]:

```
# To retrieve the intercept
```

In [25]:

```
print(reg.intercept_)
```

```
2.018160041434683
```

In [26]:

```
# For retrieving the slope (coefficient of x)
```

In [27]:

```
print(reg.coef_)
```

```
[9.91065648]
```

In [28]:

```
# This means that for every one unit of change in hours studied, the change in the score is
# Or in simpler words, if a student studies one hour more than they previously studied for
# they can expect to achieve an increase of 9.91% in the score achieved by the student prev
```

In [29]:

```
# lets see the predictions now
```

In [30]:

```
# Now that we have trained our algorithm, it's time to make some predictions.
# To do so, we will use our test data and see how accurately our algorithm predicts the per
```

In [31]:

```
y_pred = reg.predict(X_test)
y_pred
```

Out[31]:

```
array([16.88414476, 33.73226078, 75.357018  , 26.79480124, 60.49103328])
```

In [32]:

```
# now lets compare the actual output values for X_test with the predicted values
```

In [33]:

```
ndf = pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
ndf
```

Out[33]:

	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

In [34]:

```
df
```

Out[34]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85
11	5.9	62
12	4.5	41
13	3.3	42
14	1.1	17
15	8.9	95
16	2.5	30
17	1.9	24
18	6.1	67
19	7.4	69
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

In [35]:

```
# Now Evaluate the Performance of Your Algorithm
```

In [36]:

```
# This step is particularly important to compare how well different algorithms perform on
# a particular dataset.For regression algorithms, three evaluation metrics are commonly use
```

In [37]:

```
# 1.Mean Absolute Error (MAE) is the mean of the absolute value of the errors.

# 2.Mean Squared Error (MSE) is the mean of the squared errors.
# 3.Root Mean Squared Error (RMSE)
```

In [38]:

```
# The Scikit-Learn library comes with pre-built functions that can be used to find out thes
```

In [39]:

```
from sklearn import metrics
```

In [40]:

```
print("Mean absolute error is:",metrics.mean_absolute_error(y_test, y_pred))
print("Mean sqared error:",metrics.mean_squared_error(y_test, y_pred))
print("Root mean squared Error:", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean absolute error is: 4.183859899002975
Mean sqared error: 21.5987693072174
Root mean squared Error: 4.6474476121003665
```

In [41]:

```
# You can see that the value of root mean squared error is 4.64, which is less
# than 10% of the mean value of the percentages of all the students
# i.e. 51.48. This means that our algorithm did a decent job.
```

In [42]:

```
df.describe()
```

Out[42]:

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

In [44]:

```
X_new = pd.DataFrame({'Hours': [df.Hours.min(), df.Hours.max()]})
X_new
```

Out[44]:

	Hours
0	1.1
1	9.2

In [45]:

```
y_new = reg.predict(X_new)
y_new
```

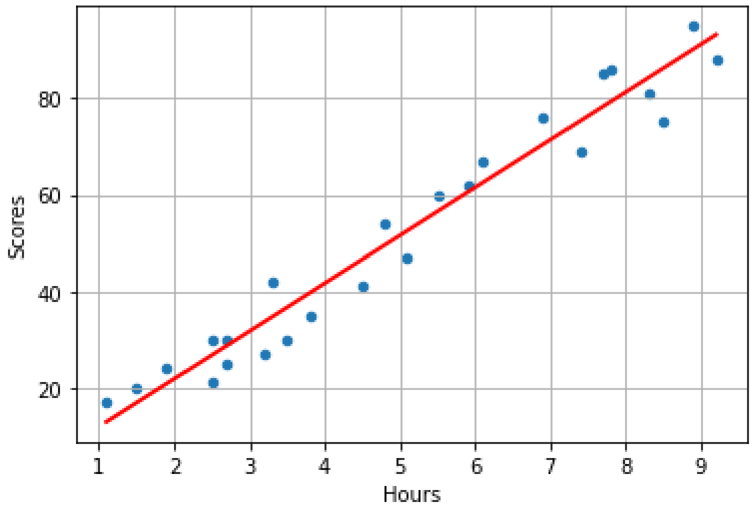
Out[45]:

```
array([12.91988217, 93.19619966])
```

In [48]:

```
# first, plot the observed data
df.plot(kind='scatter', x='Hours', y='Scores')

# then, plot the least squares line
plt.plot(X_new, y_new, c='red', linewidth=2)
plt.grid()
```



In [50]:

```
reg.predict([[7]])
```

Out[50]:

```
array([71.39275541])
```

In [ ]: