Course name: **Python programming and analytics by Rahul Sir**

Topic name: **simple Linear Regression (Manav Batch)**

Video name**: simple linear regression practical Manav batch**

Video length: **1 hour 05 minutes 51 seconds**

***Linear regression*:**

The X variable is called the independent variable and the Y variable is called the dependent variable. Simple linear regression plots one independent variable X against one dependent variable Y. Technically, in regression analysis, the independent variable is usually called the predictor variable and the dependent variable is called the criterion variable. Regression analysis can result in *linear* or *nonlinear* graphs. A linear regression is where the relationships between your variables can be described with a straight line. Non-linear regressions produce curved lines. Regression analysis is almost always performed by a computer program, as the equations are extremely time-consuming to perform by hand.

Regression analysis is used to find equations that fit data. Once we have the regression equation, we can use the model to make predictions.

If you recall from elementary algebra, the equation for a line is **$y = mx + b$**.

In statistics Linear regression is denote by $y = a + bx$ (or) $y = b0 + b1x$ where Y is the dependent variable (that's the variable that goes on the

Y axis), X is the independent variable (i.e. it is plotted on the X axis), b is the slope of the line and a is the y-intercept.

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line.

$$A = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$B = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$

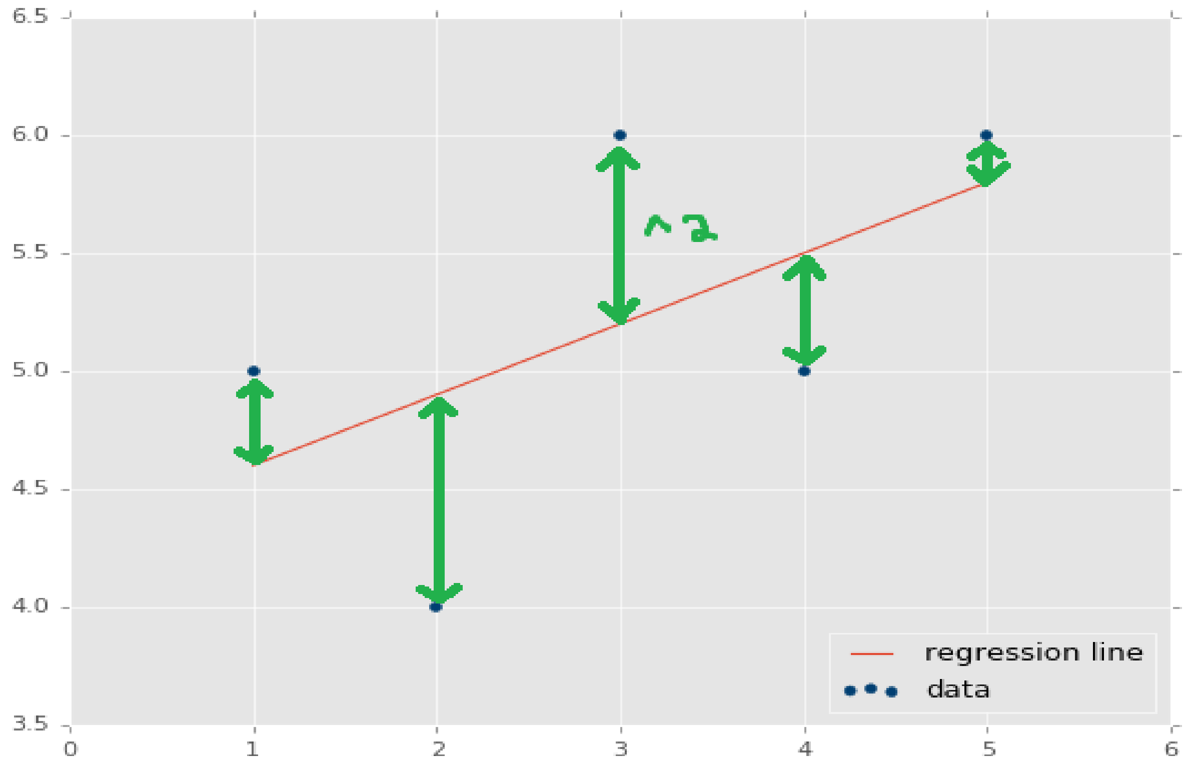
The value of b_0, also called the **intercept**, shows the point where the estimated regression line crosses the y axis. The value of b_1 determines the **slope** of the estimated regression line.

The **coefficient of determination** (denoted by R^2) is a key output of regression analysis. It is interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variable.

- The coefficient of determination is the square of the correlation (r) between predicted y scores and actual y scores; thus, it ranges from 0 to 1.
- With linear regression, the coefficient of determination is also equal to the square of the correlation between x and y scores.
- An R^2 of 0 means that the dependent variable cannot be predicted from the independent variable.

- An R^2 of 1 means the dependent variable can be predicted without error from the independent variable.
- An R^2 between 0 and 1 indicates the extent to which the dependent variable is predictable. An R^2 of 0.10 means that 10 percent of the variance in *Y* is predictable from *X*; an R^2 of 0.20 means that 20 percent is predictable; and so on.

In statistics, the **coefficient of determination**, denoted by *R2* or *r2* and pronounced "R squared", is a number that indicates the proportion of the variance in the dependent variable that is predictable from the independent variable(s). This value ranges from 0 to 1. Value '1' indicates predictor perfectly accounts for all the variation in Y. Value '0' indicates that predictor 'x' accounts for no variation in 'y'



- The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

- The regression sum of squares, also called the explained sum of squares:

$$SS_{\text{reg}} = \sum_i (f_i - \bar{y})^2,$$

- The sum of squares of residuals, also called the residual sum of squares:

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

The most general definition of the coefficient of determination is

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}.$$

There are five basic steps when you're implementing linear regression:

1. Import the packages and classes you need.
2. Provide data to work with and eventually do appropriate transformations.
3. Create a regression model and fit it with existing data.
4. Check the results of model fitting to know whether the model is satisfactory.
5. Apply the model for predictions.

The final step is to evaluate the performance of algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, three evaluation metrics are commonly used:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$$

Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

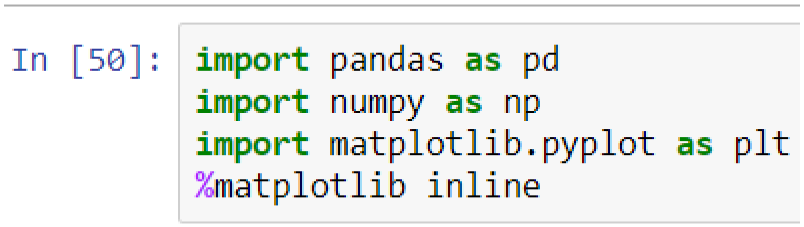
$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

Luckily, we don't have to perform these calculations manually. The Scikit-Learn library comes with pre-built functions that can be used to find out these values for us.

Step 1: Import packages and classes



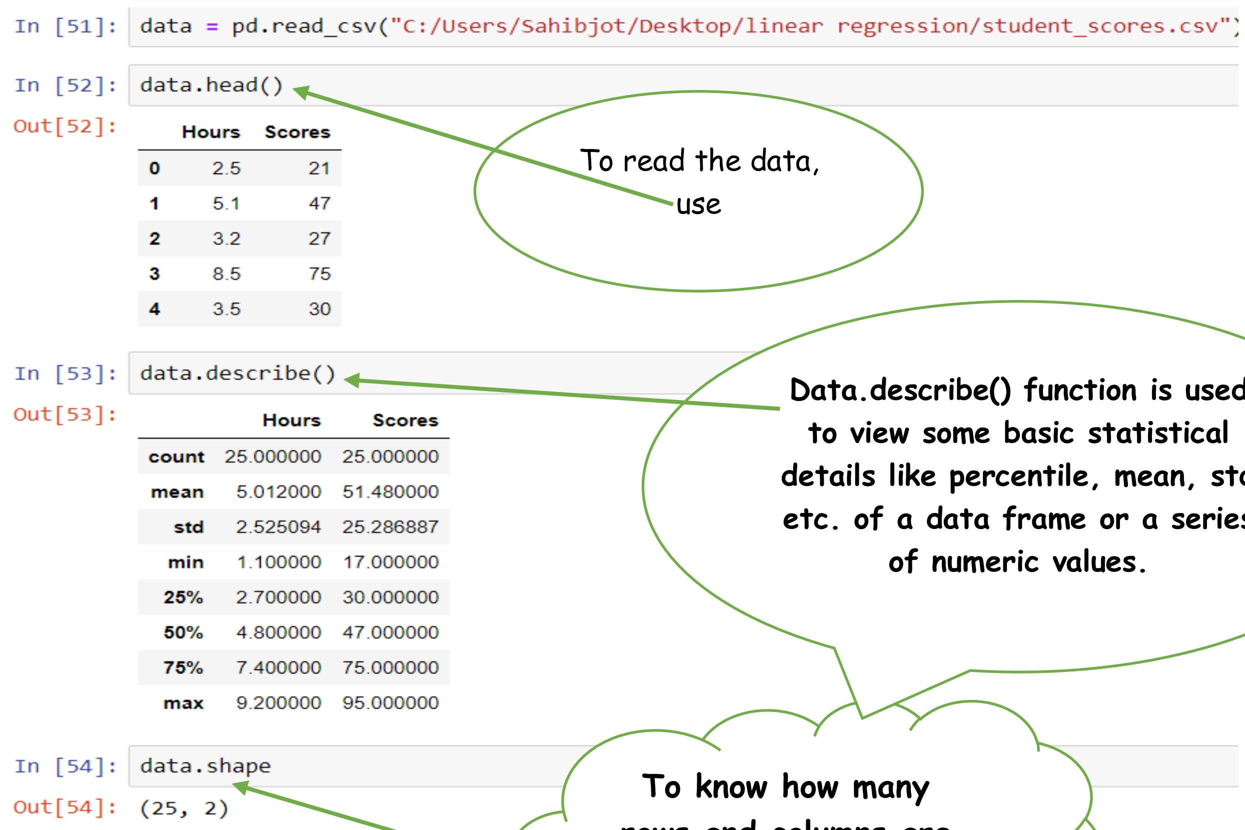
```
In [50]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline
```

By using the **import function**, you can import the NumPy and Pandas package in the notebook.

Like, import NumPy as np

import Pandas as pd

Step 2: Provide data



```
In [51]: data = pd.read_csv("C:/Users/Sahibjot/Desktop/linear regression/student_scores.csv")

In [52]: data.head()
Out[52]:
   Hours  Scores
0    2.5      21
1    5.1      47
2    3.2      27
3    8.5      75
4    3.5      30

In [53]: data.describe()
Out[53]:
           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 [54]: data.shape
Out[54]: (25, 2)
```

To read the data, use

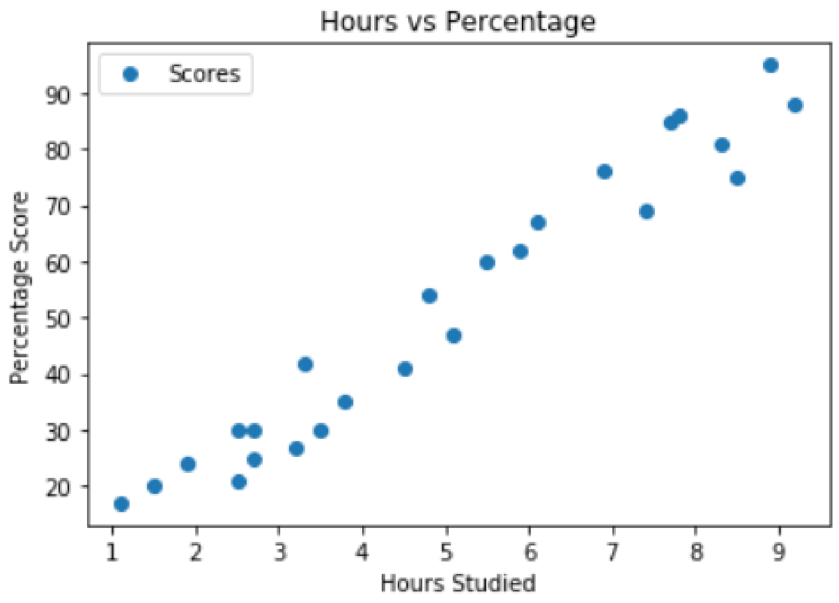
Data.describe() function is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values.

To know how many rows and columns are there in the data, use data.shape

Pd.read_csv() function is used to import the data. In the brackets, you need to mention the location of the data file in your computer/laptop and it must be in quotes.

If you want to plot the graph:

```
data.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



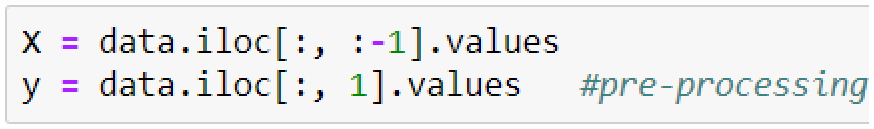
Step3: Data Preprocessing

• Pre-processing refers to the transformations applied to our data before feeding it to the algorithm.
• Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

Need of Data Preprocessing

• For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner.

• Another aspect is that data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithms are executed in one data set, and best out of them is chosen.



```
X = data.iloc[:, :-1].values
y = data.iloc[:, 1].values   #pre-processing
```

Pandas provide a unique method to retrieve rows from a Data frame. **Data.iloc[]** method is used when the index label of a data frame is something other than numeric series of 0, 1, 2, 3….n or in case the user doesn't know the index label. Rows can be extracted using an imaginary index position which isn't visible in the data frame. iloc is used for indexing or selecting based on position i.e., by row number and column number

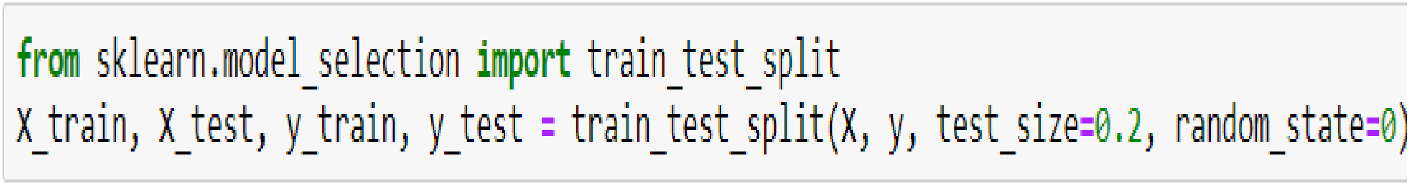
Step4: split the data

You can't possibly manually split the dataset into two. And you also have to make sure you split the data in a random manner. To help us with this task, the SciKit library provides a tool, called the Model Selection library. There's a class in the library which is, aptly, named 'train_test_split.' Using this we can easily split the dataset into the training and the testing datasets in various proportions.

- **test_size** — This parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.2 as the value, the dataset will be split 20% as the test dataset.
- **train_size** — You have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but

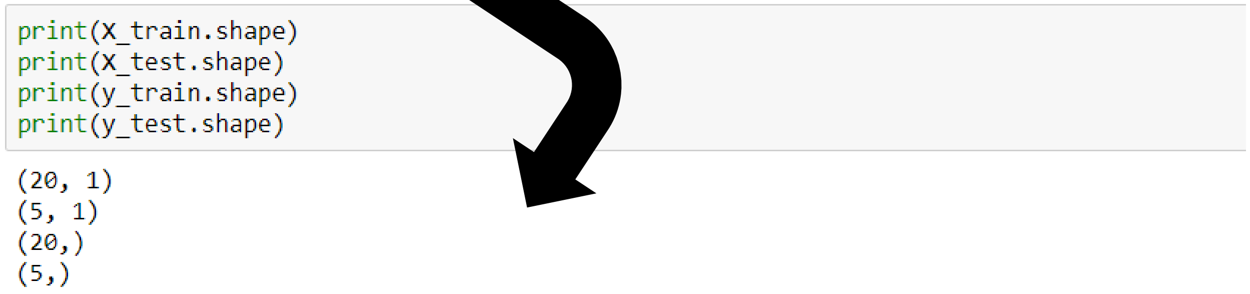
instead you tell the class what percent of the dataset you want to split as the training set.

- **random_state** — Here you pass an integer, which will act as the seed for the random number generator during the split.



```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

if you want to check how many rows and columns are there in train and test data for X and Y

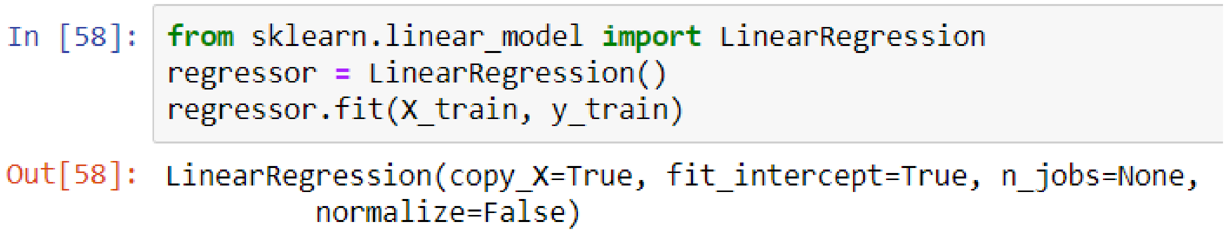


```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(20, 1)
(5, 1)
(20,)
(5,)
```

As you can see, x has two dimensions, and x_test.shape is (5, 1), while y has a single dimension, and y_test.shape is (5,).

Step 5: Create a model and fit it



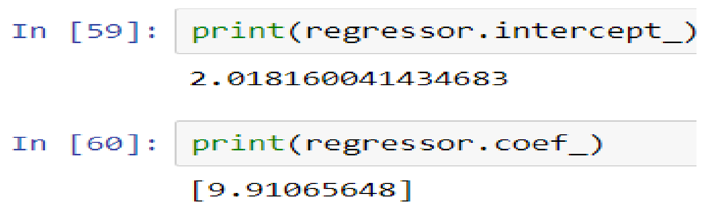
```
In [58]: from sklearn.linear_model import LinearRegression
         regressor = LinearRegression()
         regressor.fit(X_train, y_train)

Out[58]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                  normalize=False)
```

- **fit_intercept** is a Boolean (True by default) that decides whether to calculate the intercept b_0 (True) or consider it equal to zero (False).

- **normalize** is a Boolean (False by default) that decides whether to normalize the input variables (True) or not (False).
- **copy_X** is a Boolean (True by default) that decides whether to copy (True) or overwrite the input variables (False).
- **n_jobs** is an integer or None (default) and represents the number of jobs used in parallel computation. None usually means one job and -1 to use all processors.

With .fit(), you calculate the optimal values of the weights b_0 and b_1, using the existing input and output (x and y) as the arguments. In other words, .fit() **fits the model**.

If you want to find the value of intercept and coefficient:

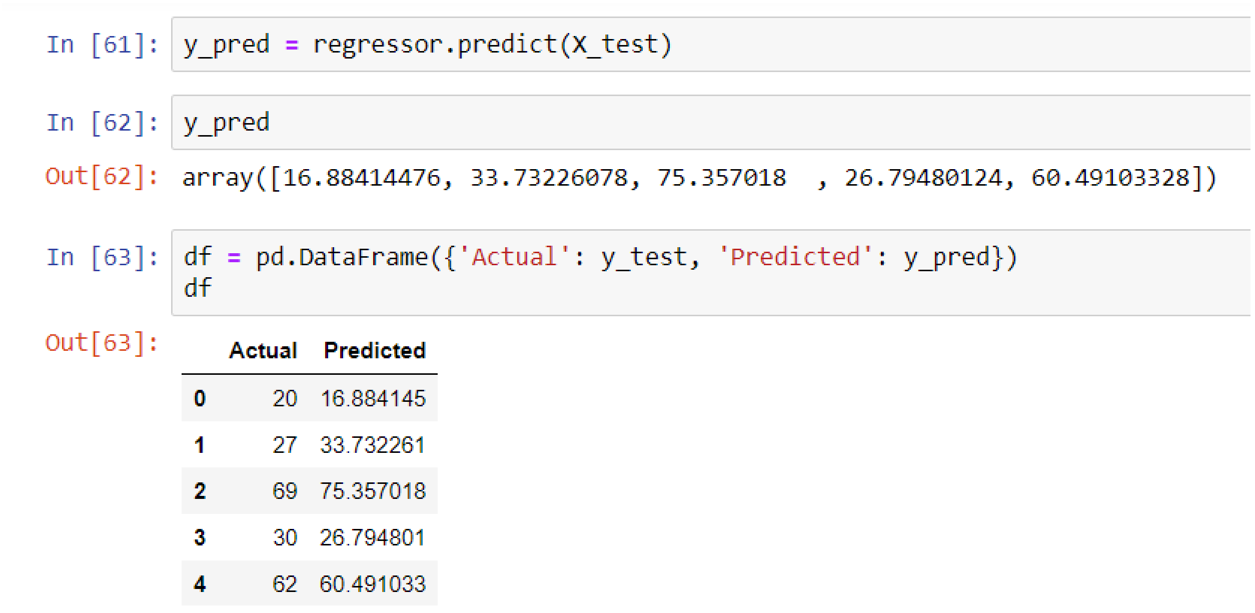


```
In [59]: print(regressor.intercept_)
         2.018160041434683

In [60]: print(regressor.coef_)
         [9.91065648]
```

The attributes of model are .intercept_, which represents the coefficient, b_0 and .coef_, which represents b_1. You can notice that .intercept_ is a scalar, while .coef_ is an array.

The value b_0 = 2.0181(approximately) illustrates that your model predicts the response 2.018 when x is zero. The value b_1 = 9.91 means that the predicted response rises by 9.91 when x is increased by one.

Step 6: Predictions

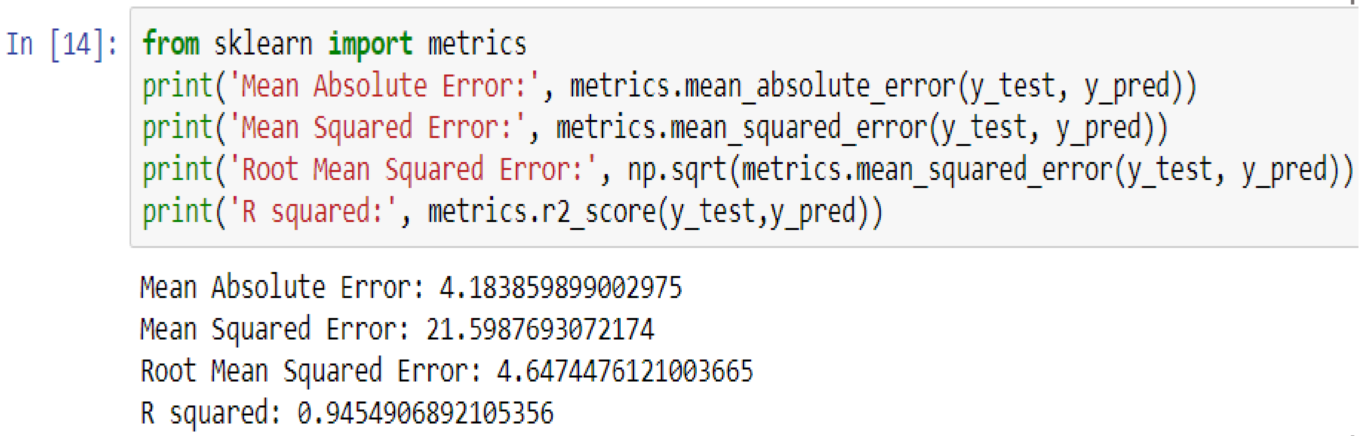


```
In [61]: y_pred = regressor.predict(X_test)

In [62]: y_pred

Out[62]: array([16.88414476, 33.73226078, 75.357018  , 26.79480124, 60.49103328])

In [63]: df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
         df

Out[63]:
```

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

.predict is used to get the predictions

Step7: evaluation



```
In [14]: from sklearn import metrics
         print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
         print('R squared:', metrics.r2_score(y_test,y_pred))

         Mean Absolute Error: 4.183859899002975
         Mean Squared Error: 21.5987693072174
         Root Mean Squared Error: 4.6474476121003665
         R squared: 0.9454906892105356
```

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