

How do you perform simple linear regression in Python step-by-step?

Authored by
stats writer

April 21, 2024

RECOMMENDED CITATION

stats writer (2024). *How do you perform simple linear regression in Python step-by-step?*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=137719>

Simple linear regression is a statistical technique used to model the relationship between two continuous variables, with one being the independent variable and the other being the dependent variable. In Python, this can be performed using the "statsmodels" library. The following are the step-by-step instructions for performing simple linear regression in Python:

1. Import the necessary libraries: Begin by importing the "statsmodels" library, along with any other relevant libraries such as "numpy" and "pandas".
2. Load the data: Next, load the data set that contains the two variables of interest. This can be done using the "read_csv" function from the "pandas" library.
3. Define the variables: Once the data is loaded, define the independent variable (X) and the dependent variable (y).
4. Create the regression model: Use the "ols" function from the "statsmodels" library to create a linear regression model. This function takes in the independent and dependent variables as inputs.
5. Fit the model: Use the "fit" method on the regression model to fit the data and obtain the coefficients for the regression equation.
6. View the results: The "summary" method can be used to view the results of the regression, including the coefficient values, p-values, and R-squared value.
7. Make predictions: With the fitted model, predictions can be made by passing new values for the independent variable to the "predict" method.
8. Visualize the results: Finally, the results can be visualized using a scatter plot of the data points and the regression line.

By following these steps, simple linear regression can be easily performed in Python, providing valuable insights into the relationship between two variables.

Perform Simple Linear Regression in Python (Step-by-Step)

Simple linear regression is a technique that we can use to understand the relationship between a single explanatory variable and a single response variable.

This technique finds a line that best "fits" the data and takes on the following form:

$$y = b_0 + b_1x$$

where:

y : The estimated response value
 b_0 : The intercept of the regression line
 b_1 : The slope of the regression line

This equation can help us understand the relationship between the explanatory and response variable, and (assuming it's statistically significant) it can be used to predict the value of a response variable given the value of the explanatory variable.

This tutorial provides a step-by-step explanation of how to perform simple linear regression in Python.

Step 1: Load the Data

For this example, we'll create a fake dataset that contains the following two variables for 15 students:

Total hours studied for some exam
Exam score

We'll attempt to fit a simple linear regression model

using *hours* as the explanatory variable and *exam score* as the response variable.

The following code shows how to create this fake dataset in Python:

```
import pandas as pd
```

```
#create dataset
```

```
df = pd.DataFrame({'hours': ,  
'score': })
```

```
#view first six rows of dataset
```

```
df
```

```
hours score
```

```
0 1 64
```

```
1 2 66
```

```
2 4 76
```

```
3 5 73
```

```
4 5 74
```

```
5 6 81
```

Step 2: Visualize the Data

Before we fit a simple linear regression model, we

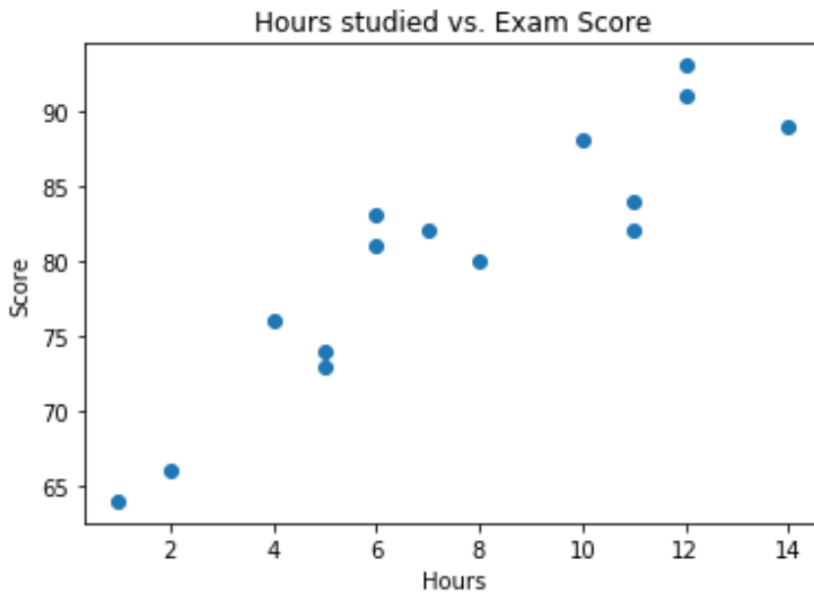
should first visualize the data to gain an understanding of it.

First, we want to make sure that the relationship between *hours* and *score* is roughly linear, since that is an underlying assumption of simple linear regression.

We can create a simple scatterplot to view the relationship between the two variables:

```
import matplotlib.pyplot as plt

plt.scatter(df.hours, df.score)
plt.title('Hours studied vs. Exam Score')
plt.xlabel('Hours')
plt.ylabel('Score')
plt.show()
```

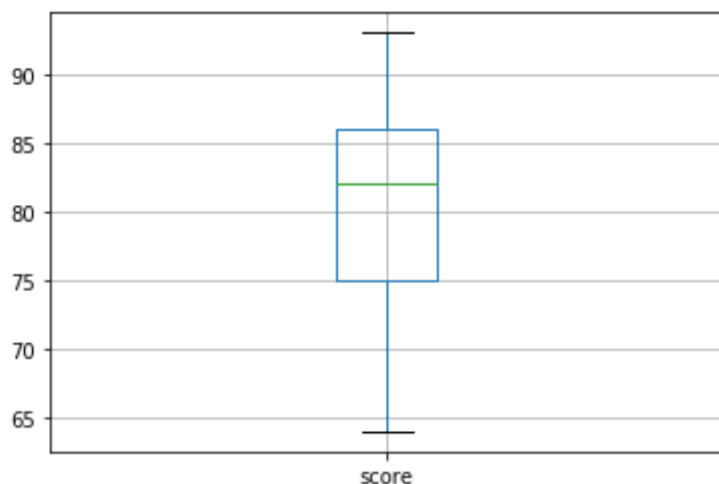


From the plot we can see that the relationship does appear to be linear. As *hours* increases, *score* tends to increase as well in a linear fashion.

Next, we can create a boxplot to visualize the distribution of exam scores and check for outliers. By default, Python defines an observation to be an outlier if it is 1.5 times the interquartile range greater than the third quartile (Q3) or 1.5 times the interquartile range less than the first quartile (Q1).

If an observation is an outlier, a tiny circle will appear in the boxplot:

```
df.boxplot(column=)
```



There are no tiny circles in the boxplot, which means there are no outliers in our dataset.

Step 3: Perform Simple Linear Regression

Once we've confirmed that the relationship between our variables is linear and that there are no outliers present, we can proceed to fit a simple linear regression model using *hours* as the explanatory variable and *score* as the response variable:

Note: We'll use the OLS() function from the statsmodels library to fit the regression model.

```
import statsmodels.api as sm
```

```
#define response variable
```

```
y = df
```

```
#define explanatory variable
```

```
x = df]
```

```
#add constant to predictor variables
```

```
x = sm.add_constant(x)
```

```
#fit linear regression model
```

```
model = sm.OLS(y, x).fit()
```

```
#view model summary
```

```
print(model.summary())
```

```
OLS Regression Results
```

```
=====
```

```
=====
```

```
Dep. Variable: score R-squared: 0.831
```

```
Model: OLS Adj. R-squared: 0.818
```

```
Method: Least Squares F-statistic: 63.91
```

```
Date: Mon, 26 Oct 2020 Prob (F-statistic): 2.25e-06
```

```
Time: 15:51:45 Log-Likelihood: -39.594
```

```
No. Observations: 15 AIC: 83.19
```

```
Df Residuals: 13 BIC: 84.60
```

```
Df Model: 1
```

Covariance Type: nonrobust

```
=====
=====
coef std err t P>|t|
-----
const 65.3340 2.106 31.023 0.000 60.784 69.884
hours 1.9824 0.248 7.995 0.000 1.447 2.518
=====
=====
Omnibus: 4.351 Durbin-Watson: 1.677
Prob(Omnibus): 0.114 Jarque-Bera (JB): 1.329
Skew: 0.092 Prob(JB): 0.515
Kurtosis: 1.554 Cond. No. 19.2
=====
=====
```

From the model summary we can see that the fitted regression equation is:

$$\text{Score} = 65.334 + 1.9824 * (\text{hours})$$

This means that each additional hour studied is associated with an average increase in exam score of 1.9824 points. And the intercept value of 65.334 tells us

the average expected exam score for a student who studies zero hours.

We can also use this equation to find the expected exam score based on the number of hours that a student studies. For example, a student who studies for 10 hours is expected to receive an exam score of 85.158:

$$\text{Score} = 65.334 + 1.9824*(10) = 85.158$$

Here is how to interpret the rest of the model summary:

P>|t|: This is the p-value associated with the model coefficients. Since the p-value for *hours* (0.000) is significantly less than .05, we can say that there is a statistically significant association between *hours* and *score*.
R-squared: This number tells us the percentage of the variation in the exam scores can be explained by the number of hours studied. In general, the larger the R-squared value of a regression model the better the explanatory variables are able to predict the value of the response variable. In this case, 83.1% of the variation in scores can be explained by hours studied.
F-statistic & p-value: The F-statistic (63.91) and the corresponding p-

value (2.25e-06) tell us the overall significance of the regression model, i.e. whether explanatory variables in the model are useful for explaining the variation in the response variable. Since the p-value in this example is less than .05, our model is statistically significant and *hours* is deemed to be useful for explaining the variation in *score*.

Step 4: Create Residual Plots

After we've fit the simple linear regression model to the data, the last step is to create residual plots.

One of the key assumptions of linear regression is that the residuals of a regression model are roughly normally distributed and are homoscedastic at each level of the explanatory variable. If these assumptions are violated, then the results of our regression model could be misleading or unreliable.

To verify that these assumptions are met, we can create the following residual plots:

Residual vs. fitted values plot: This plot is useful for confirming homoscedasticity. The x-axis displays the fitted values and the y-axis displays the residuals. As

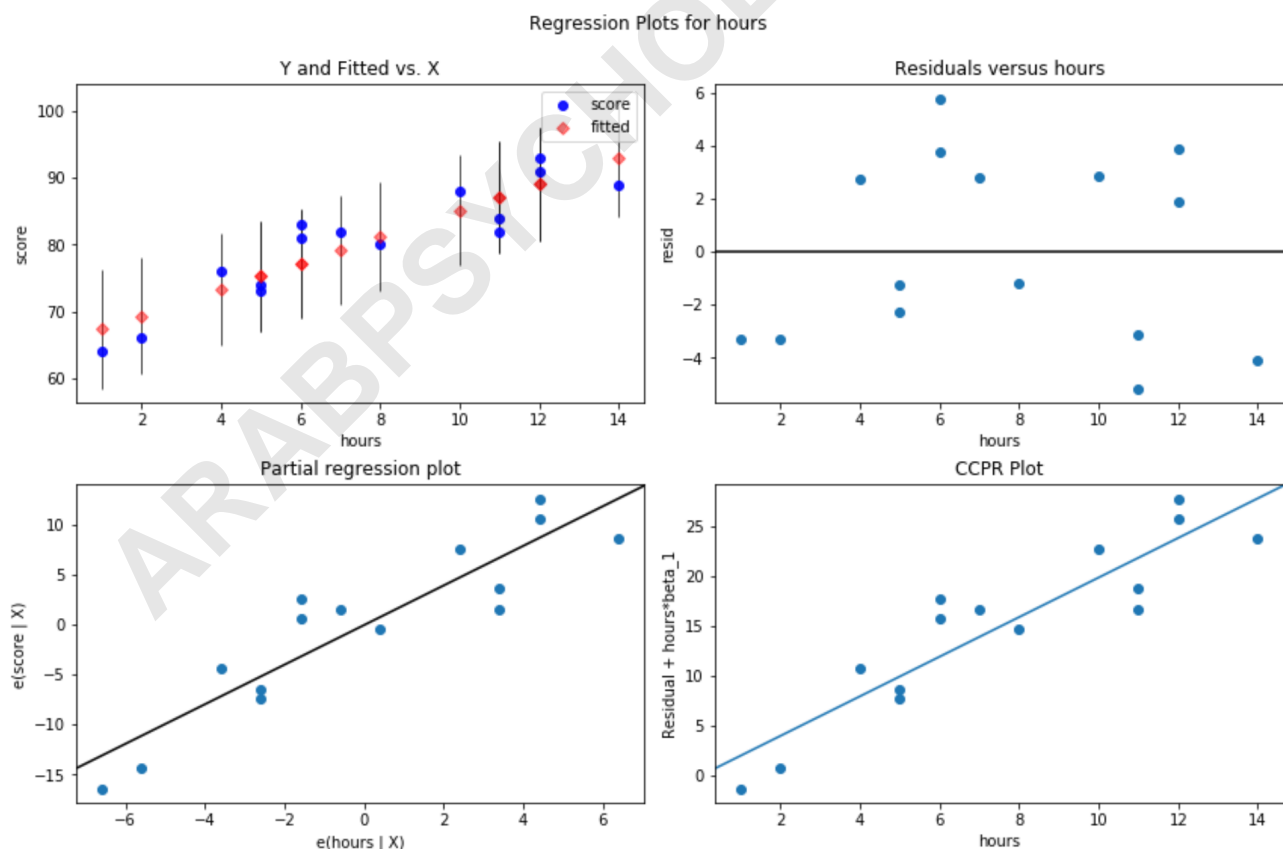
long as the residuals appear to be randomly and evenly distributed throughout the chart around the value zero, we can assume that homoscedasticity is not violated:

```
#define figure size
```

```
fig = plt.figure(figsize=(12,8))
```

```
#produce residual plots
```

```
fig = sm.graphics.plot_regress_exog(model, 'hours',
fig=fig)
```



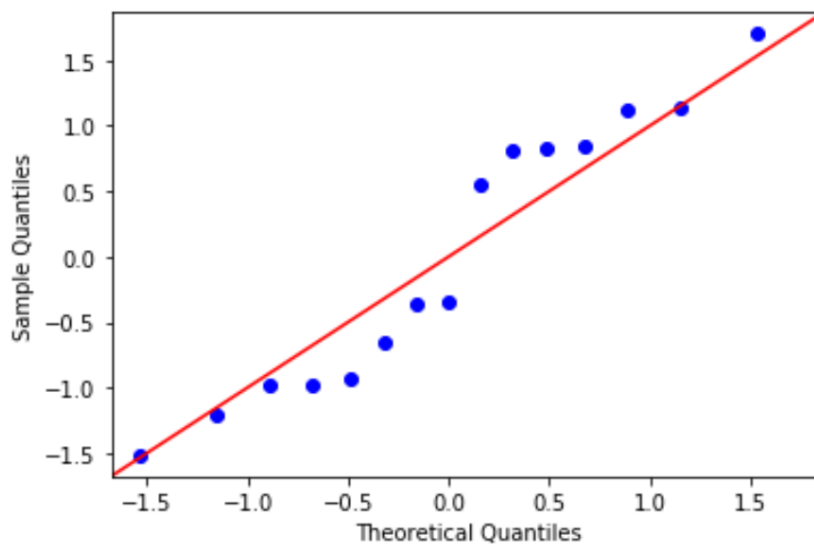
Four plots are produced. The one in the top right corner

is the residual vs. fitted plot. The x-axis on this plot shows the actual values for the predictor variable *points* and the y-axis shows the residual for that value.

Since the residuals appear to be randomly scattered around zero, this is an indication that heteroscedasticity is not a problem with the explanatory variable.

Q-Q plot: This plot is useful for determining if the residuals follow a normal distribution. If the data values in the plot fall along a roughly straight line at a 45-degree angle, then the data is normally distributed:

```
#define residuals
res = model.resid#create Q-Q plot
fig = sm.qqplot(res, fit=True, line="45")
plt.show()
```



The residuals stray from the 45-degree line a bit, but not enough to cause serious concern. We can assume that the normality assumption is met.

Since the residuals are normally distributed and homoscedastic, we've verified that the assumptions of the simple linear regression model are met. Thus, the output from our model is reliable.

The complete Python code used in this tutorial can be found [here](#).