Learning Simple Linear Regression with Python: A Step-by-Step Guide

Authored by

Mohammed looti

November 6, 2025

Introduction to Simple Linear Regression

Statistical modeling provides powerful tools essential for understanding complex relationships hidden within data. Among the fundamental techniques in this field is Simple Linear Regression (SLR). SLR is a robust statistical method used specifically when the goal is to quantify the linear association between two continuous variables: a single **explanatory variable** (or predictor) and a single **response variable** (or outcome). Understanding this relationship is crucial for prediction and inference in many scientific and business domains.

The primary objective of SLR is to identify the **line of best fit**--a straight line that minimizes the sum of squared vertical distances between the actual data points and the line itself. This derived regression line serves as a model that allows analysts to interpret the nature, direction, and strength of the association between the variables. Provided that the relationship is statistically sound, the resulting model can be effectively utilized to generate reliable predictions for the response variable based on new values of the explanatory variable.

The mathematical foundation of this linear relationship is defined by a straightforward equation:

? = b0 + b1x

A clear comprehension of the components within this formula is vital for accurate interpretation of the model's output:

?: This represents the **estimated response value**, which is the model's prediction.
b0: This is the **intercept** of the regression line. It defines the predicted value of ? when the explanatory variable x is equal to zero.
b1: This is the **slope** coefficient. It quantifies the average change in the predicted response value (?) for every one-unit increase in the explanatory variable (x).

This tutorial offers a practical, detailed, and step-by-step methodology for performing a Simple Linear Regression analysis entirely within the **Python** programming environment. We will leverage industry-standard statistical and visualization libraries to ensure both accuracy and comprehensive model validation.

Step 1: Preparing and Loading the Data in Python

The success of any regression analysis hinges upon the quality and structure of the input data. To illustrate the process, we will begin by constructing a synthetic dataset designed to explore the relationship between the effort invested in studying and the resulting academic performance. Our dataset comprises 15 observations, detailing the total hours a student studied for an exam alongside their corresponding final score.

Our goal is to fit an SLR model using *hours* as the explanatory variable and *exam score* as the **response variable**. The central hypothesis driving this analysis is that an increase in study time is positively correlated with a resultant increase in exam scores.

The Python library Pandas is the industry-standard tool for efficient data manipulation and structuring. The following code snippet demonstrates how to define our data points and convert them into a robust Pandas DataFrame, establishing the necessary structure for all subsequent statistical calculations:

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

This foundational step ensures that our data is correctly imported, labeled, and structured, which is a prerequisite for effective exploratory data analysis (EDA) and model fitting.

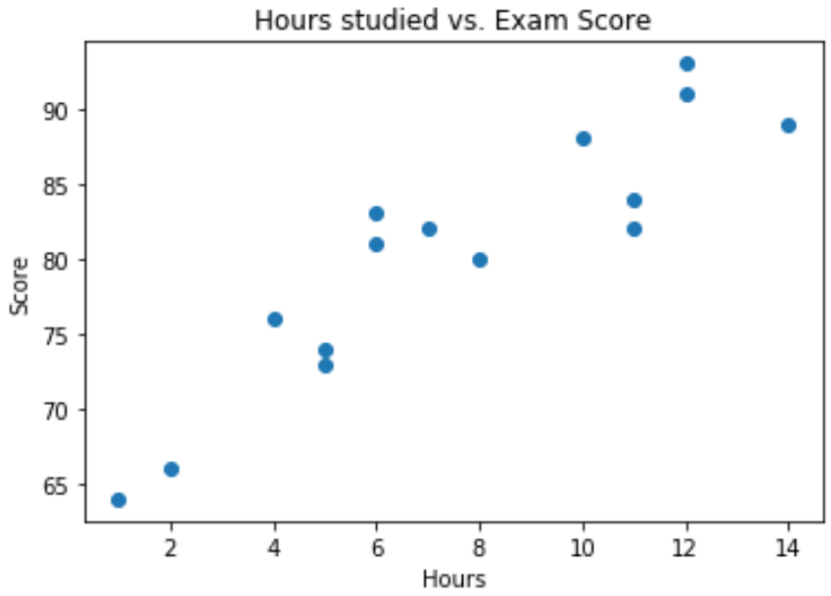
Step 2: Exploratory Data Analysis and Assumption Checks

Before committing to fitting the regression model, performing a thorough visual inspection of the data is a critical intermediate step. This exploratory phase helps us verify if the fundamental assumptions underlying Simple Linear Regression are plausibly met. The most crucial assumption is that the relationship between the two variables is reasonably **linear**.

We rely on a scatterplot to immediately visualize the joint distribution and relationship between the independent variable (*hours*) and the dependent variable (*score*). If the resulting plot shows data points forming a discernible pattern that generally follows a straight line, the linearity assumption can be considered satisfied:

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()
```

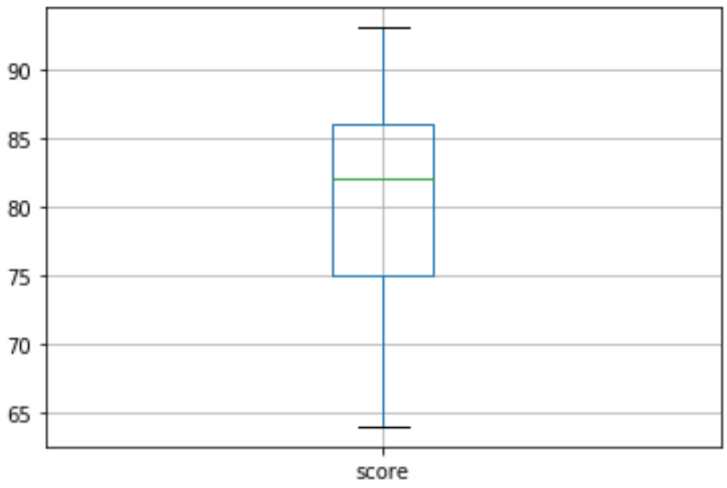


The visual evidence presented by the scatterplot strongly suggests a clear, positive linear trend. As the total number of study hours increases, the corresponding exam score tends to rise consistently, confirming that a simple linear model is indeed an appropriate choice for analyzing this particular dataset.

Furthermore, we must proactively check for the presence of influential data points, commonly known as outliers. These unusual observations possess the potential to severely skew the fit of the Ordinary Least Squares (OLS) regression line, compromising the reliability of the parameters. A boxplot of the response variable (score) is an excellent diagnostic tool for identifying these anomalies. Python's standard boxplot definition flags an observation as a potential outlier if it falls 1.5 times the Interquartile Range (IQR) outside the central box.

If any observation were deemed an outlier, it would appear as a small, isolated circle or symbol outside the whiskers of the resulting boxplot:

df.boxplot(column=)



The generated boxplot successfully confirms that there are no isolated points, indicating that our dataset is free from significant outliers that might negatively influence the model fitting process or distort the coefficient estimates.

Step 3: Fitting the OLS Model and Interpreting Results

With the necessary checks for linearity and the absence of influential outliers complete, we are ready to fit the simple linear regression model using the **Ordinary Least Squares** (OLS) estimation method. For this critical step, we utilize the `statsmodels` library, which is highly regarded within the data science community for its statistical rigor and provision of comprehensive output summaries, similar to those found in R or SAS.

A key technical detail when employing `statsmodels` is the necessity of explicitly including the intercept term (b_0) within the explanatory variable matrix. We achieve this by using the highly functional `sm.add_constant(x)` method. Following this adjustment, we fit the core OLS model, designating *score* (y) as the dependent variable and the constant-adjusted *hours* (x) as the independent variable.

The following sequence of Python code executes the regression analysis and prints the resulting statistical summary table, which contains all the necessary metrics for interpretation:

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
==============================================================================
```

By examining the coefficients table within the summary, we can extract the estimated parameters necessary to construct our final fitted regression equation:

Score = 65.334 + 1.9824 * (Hours Studied)

The estimated **slope** coefficient for *hours* (1.9824) provides the core insight of the model. This value signifies that for every single additional hour a student devotes to studying, their expected exam score increases by approximately 1.9824 points. Conversely, the **intercept** (65.334) suggests that a hypothetical student who studies zero hours is predicted to achieve an average score of 65.334. This derived equation is now fully operational for prediction; for example, if a student studies for 10 hours, their expected score is 85.158 (calculated as 65.334 + 1.9824 * 10).

Step 4: Assessing Model Fit and Significance

While interpreting the regression coefficients is essential, it is equally important to rigorously evaluate the statistical quality and overall effectiveness of the model using the diagnostic metrics provided in the summary table. These metrics confirm whether the relationships observed are reliable and not due to random chance.

We first examine the **P>|t|** column, which reports the p-value associated with each coefficient. Since the p-value for the *hours* variable is reported as 0.000, it is substantially smaller than the conventional significance threshold of 0.05. This result allows us to confidently reject the null hypothesis and conclude that the linear relationship between hours studied and exam score is **statistically significant**.

Next, the R-squared value (0.831) serves as a measure of the model's explanatory power. This metric indicates that **83.1%** of the total variation observed in the exam scores can be successfully accounted for or explained solely by the variation in the number of hours studied. This high R-squared value confirms a strong overall fit, strongly suggesting that *hours* is a highly effective predictor of the final academic outcome.

Finally, the **F-statistic** (63.91) and its corresponding p-value (2.25e-06) assess the overall significance of the entire regression model. Because this p-value is extremely minute, we decisively conclude that the model as a whole is statistically significant, validating that the explanatory variable (hours studied) is indeed valuable for modeling the response variable (exam score).

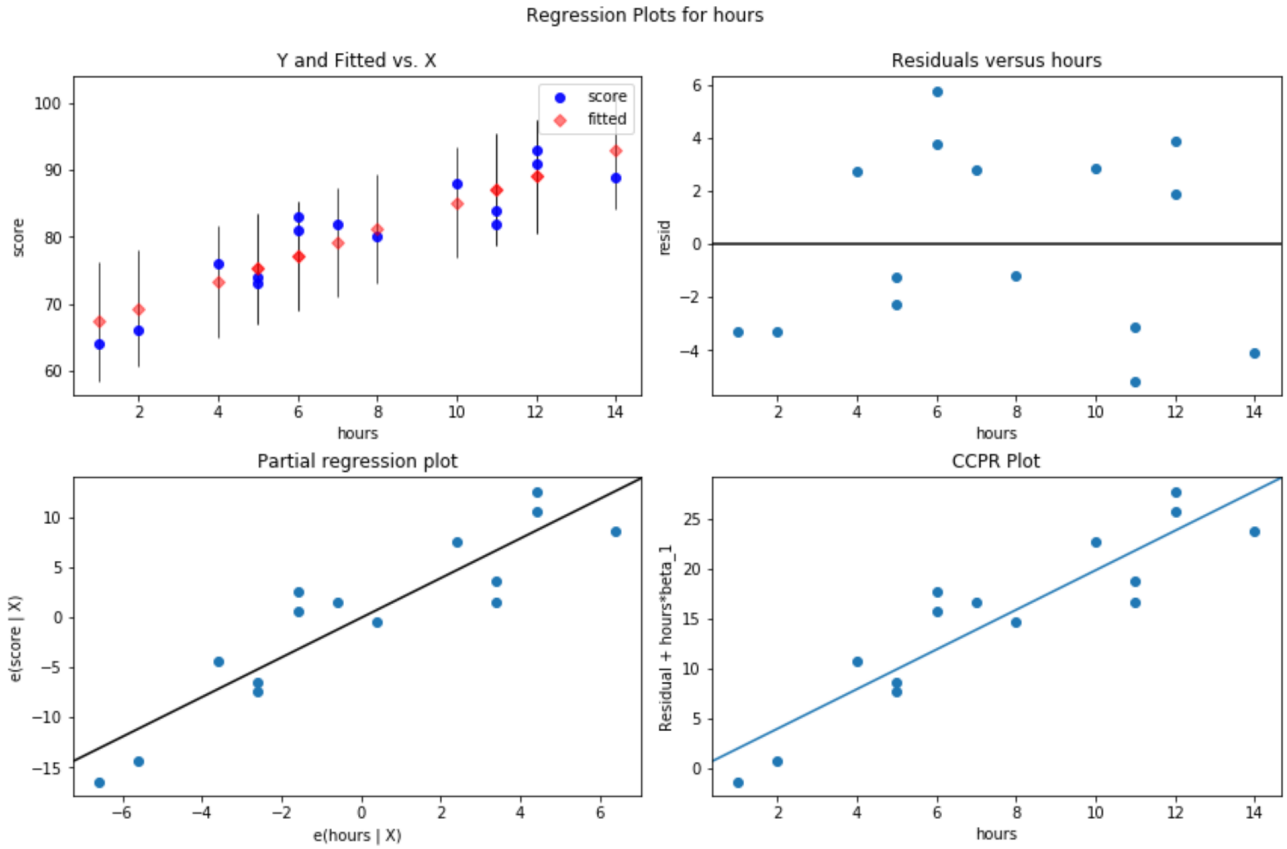
Step 5: Validating Model Assumptions with Residual Analysis

For the inferences derived from an OLS model to be considered valid and unbiased, the residuals (the differences between the observed and predicted values, or the prediction errors) must satisfy two crucial statistical assumptions: they must be approximately normally distributed, and they must exhibit homoscedasticity (meaning they must have a constant variance across all levels of the predictor variable). A violation of these assumptions renders the standard errors and p-values generated by the model untrustworthy.

We begin our validation process by using a **Residual vs. Fitted Values Plot** to confirm the assumption of homoscedasticity. In this diagnostic plot, the x-axis displays the fitted (predicted) values while the y-axis displays the calculated residuals. For the assumption to hold, the points must appear randomly scattered above and below the zero line, demonstrating no systematic widening, narrowing, or curved pattern:

#define figure size
fig = plt.figure(figsize=(12,8))

#produce residual plots
fig = sm.graphics.plot_regress_exog(model, 'hours', fig=fig)



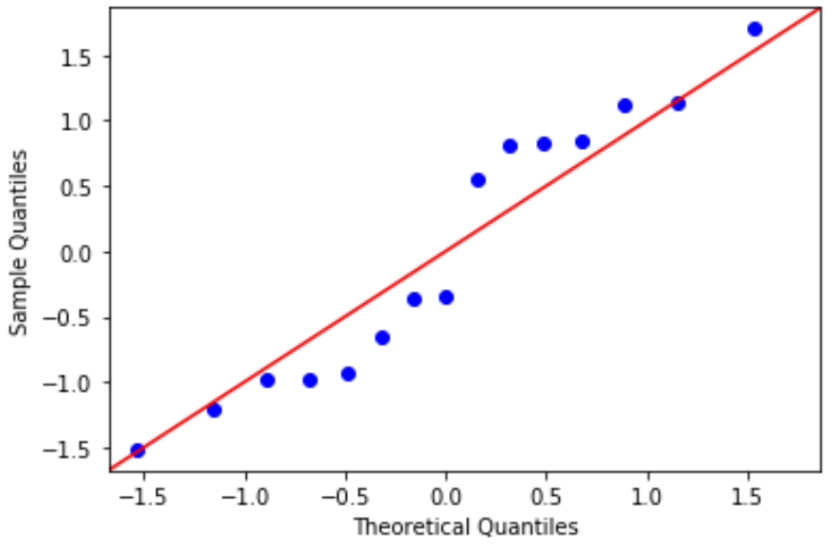
Observing the top-right panel (Residuals vs. Fitted Plot), the residuals are indeed randomly scattered around the zero line without any discernible systematic pattern or funnel shape. This confirms that the variance is constant across the range of scores, and the critical assumption of homoscedasticity is successfully met.

Next, we generate a **Q-Q Plot** (Quantile-Quantile Plot) to assess the assumption regarding the normal distribution of the residuals. Data that is normally distributed will show its points falling

tightly along the theoretical diagonal 45-degree line displayed on the plot:

#define residuals
res = model.resid

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



While minor deviations are noticeable at the extreme tails of the distribution, the vast majority of the data points closely align with the 45-degree line. We can therefore reasonably conclude that the normality assumption for the residuals is satisfied. Because both key model assumptions--constant variance and approximate normality--have been met, we can assert with high confidence that the results of our Ordinary Least Squares analysis are statistically robust and reliable for making inferences about the population relationship.

The complete Python code utilized in this tutorial is available for download and review here.