How to Create a Residual Plot in Python

Authored by

stats writer

December 24, 2025

Creating a residual plot in Python is an essential step in validating any statistical model. The process begins by importing core data science libraries such as **matplotlib** and **seaborn**, which handle visualization tasks. Following the import, the dataset must be prepared, which typically involves separating the data into independent (predictor) and dependent (response) variables. The core statistical procedure involves fitting a linear regression model to the data, followed by calculating the residuals--the differences between the observed and predicted values--for every data point. Finally, the visualization step uses a library like **seaborn** or specialized functions within **statsmodels** to graphically display these residuals against either the independent variable or the fitted values.

The Importance of Residual Plots in Statistical Modeling

A residual plot is a fundamental diagnostic tool in regression analysis, particularly when assessing the suitability of a fitted model. This visualization technique displays the difference between the observed response values and the response values predicted by the model (the residuals) against the fitted values or the predictor variables themselves. Analyzing the scatter pattern of these points is critical for confirming that the model assumptions hold true.

This type of diagnostic plot is often deployed to determine two crucial aspects of the model: first, whether a simple or multiple linear regression model is truly appropriate for the underlying dataset; and second, to check for potential violations of the core assumption of homoscedasticity. Violation of this assumption, known as heteroscedasticity, occurs when the variance of the residuals is not constant across all levels of the predictor variables. The goal is to see residuals scattered randomly around zero, showing no discernible pattern, which signifies that the variance is constant (homoscedastic).

This tutorial explains, step-by-step, how to implement and interpret a residual plot for both simple and multiple linear regression models using the powerful statistical capabilities available in Python.

Example: Setting Up the Dataset in Python

For this practical example, we will utilize a sample dataset that describes the attributes of ten basketball players. This dataset, organized using the **Pandas** library, contains variables like 'rating', 'points', 'assists', and 'rebounds'. Defining the data structure correctly is the prerequisite for any subsequent statistical modeling.

```
import numpy as np
import pandas as pd

#create dataset
```

df = pd.DataFrame({'rating': ,
'points': ,
'assists': ,
'rebounds': })

#view dataset
df

rating points assists rebounds
0 90 25 5 11
1 85 20 7 8
2 82 14 7 10
3 88 16 8 6
4 94 27 5 6
5 90 20 7 9
6 76 12 6 6
7 75 15 9 10
8 87 14 9 10
9 86 19 5 7

The dataset provides the foundation for our analysis, allowing us to proceed to fitting a regression model and subsequently generating the diagnostic visualizations.

Generating a Residual Plot for Simple Linear Regression

For our first model, we will fit a simple linear regression model using *points* as the independent variable (predictor) and *rating* as the dependent variable (response). We rely on the `ols` (Ordinary Least Squares) function available within the Statsmodels library, which simplifies the statistical estimation process.

#import necessary libraries
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols

#fit simple linear regression model
model = ols('rating ~ points', data=df).fit()

#view model summary
print(model.summary())

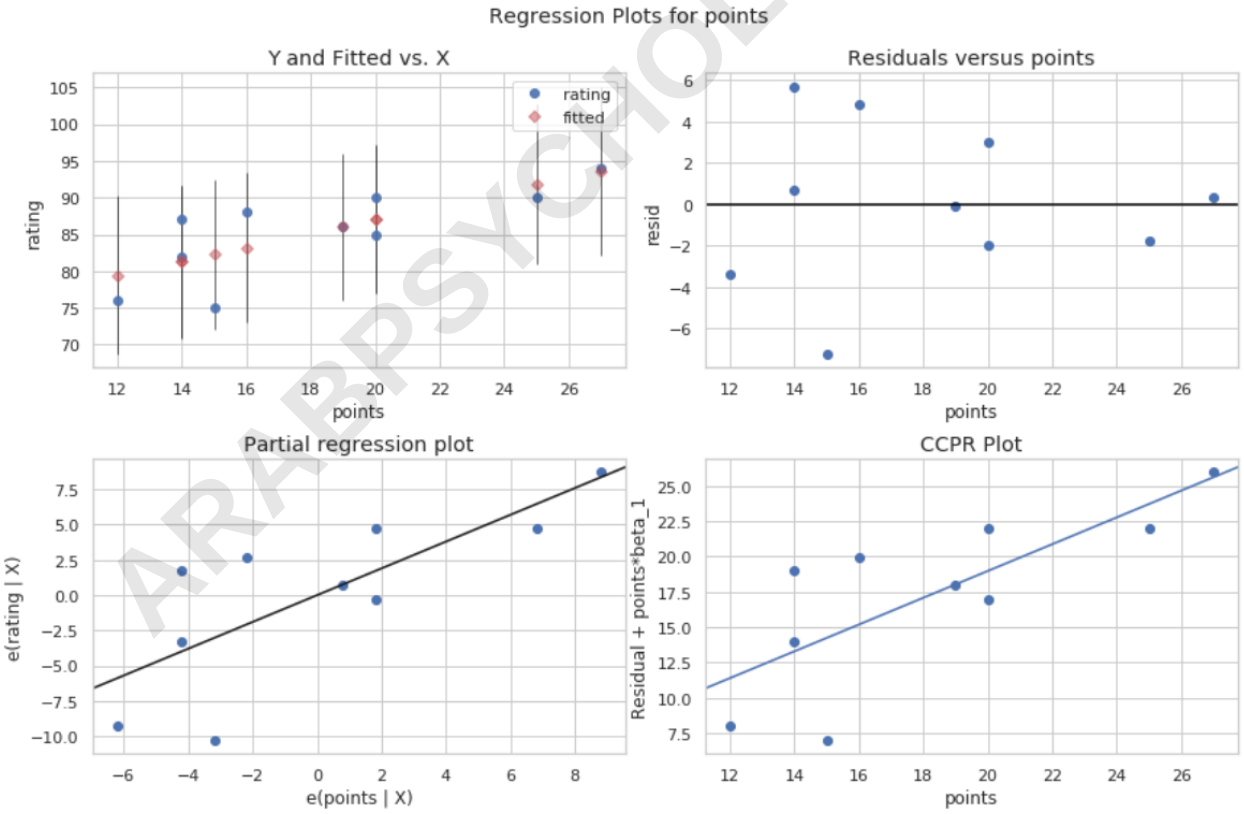
To create the diagnostic plot, we employ the `sm.graphics.plot_regress_exog` function, which is specifically designed to generate a set of four standardized regression diagnostic plots, including the vital residual vs. fitted plot for the specified predictor variable.

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

#produce regression plots
fig = sm.graphics.plot_regress_exog(model, 'points', fig=fig)

Interpreting the Simple Linear Regression Residual Plot

The output of the plotting function provides several visualizations, but we focus on the residual vs. fitted plot, typically found in the top right quadrant. The x-axis on this specific plot represents the actual values for the predictor variable *points*, and the corresponding y-axis shows the residual (error) for each observation relative to the regression line.



For a model to satisfy the assumptions of OLS, the residuals should be scattered randomly around the horizontal line at zero. Crucially, the spread of these points should remain relatively constant across all values of the predictor. Since the residuals in the visualization above appear to be

randomly scattered without exhibiting a funnel shape or any other distinct pattern, this is a strong indication that heteroscedasticity is not an issue with respect to the predictor variable *points*.

Residual Plots for Multiple Linear Regression

The methodology for residual diagnostics remains similar when transitioning to a Multiple Linear Regression (MLR) framework. For this second example, we hypothesize that *assists* and *rebounds* together predict the player's *rating*. We must fit the model using both predictors simultaneously.

```
#fit multiple linear regression model
model = ols('rating ~ assists + rebounds', data=df).fit()

#view model summary
print(model.summary())
```

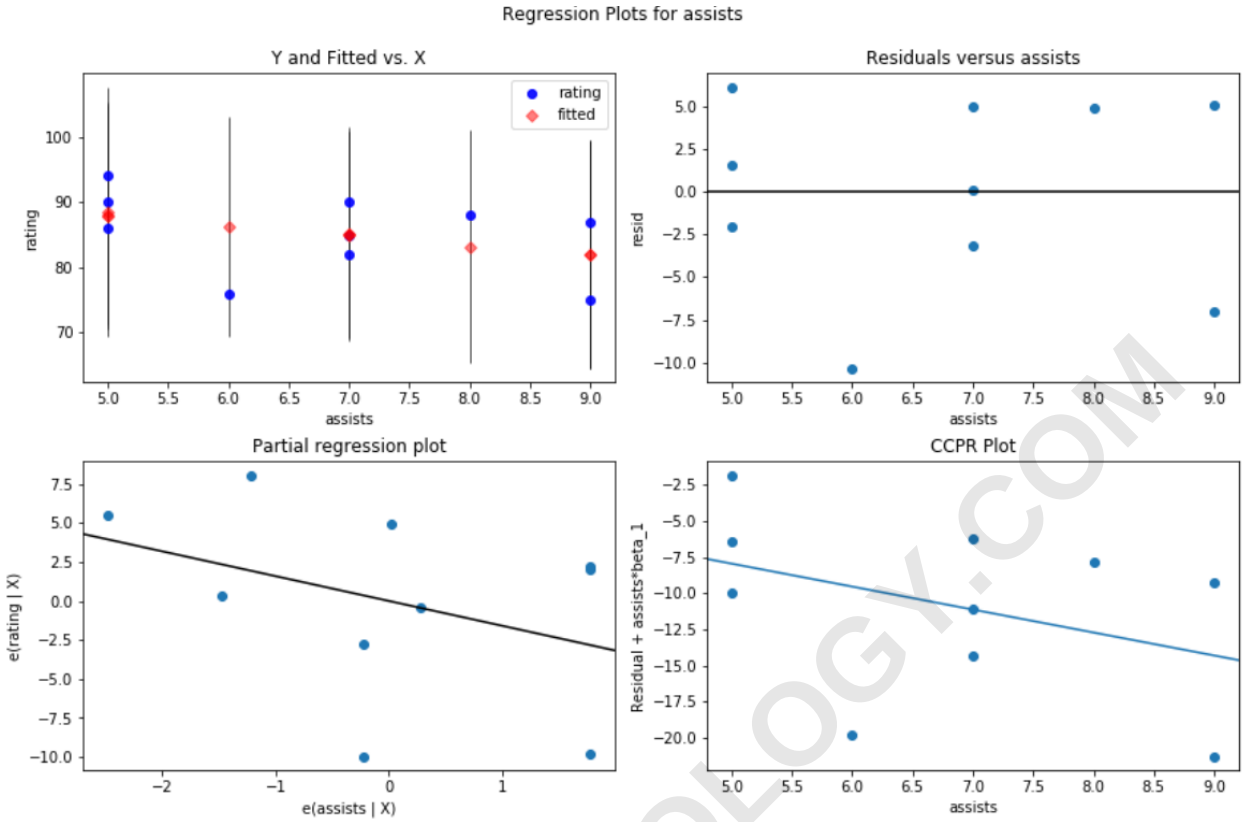
In MLR, diagnostic plots must be generated for each individual predictor variable to ensure that the assumptions are not violated by any single independent variable. We must run `sm.graphics.plot_regress_exog` once for each predictor we want to assess diagnostically.

Analyzing Individual Predictor Residual Plots in MLR (Assists)

We begin by creating the residual vs. predictor plot specifically for the *assists* variable. This visualization allows us to isolate the influence of *assists* on the model's errors, providing a granular view of assumption compliance.

```
#create residual vs. predictor plot for 'assists'
fig = plt.figure(figsize=(12,8))
fig = sm.graphics.plot_regress_exog(model, 'assists', fig=fig)
```

The plot generated for *assists* is shown below. We observe that the points are randomly dispersed around the zero line, confirming that the variance of the errors is stable across the range of *assists* values. This indicates that *assists* does not introduce systematic bias or non-constant variance into the model.

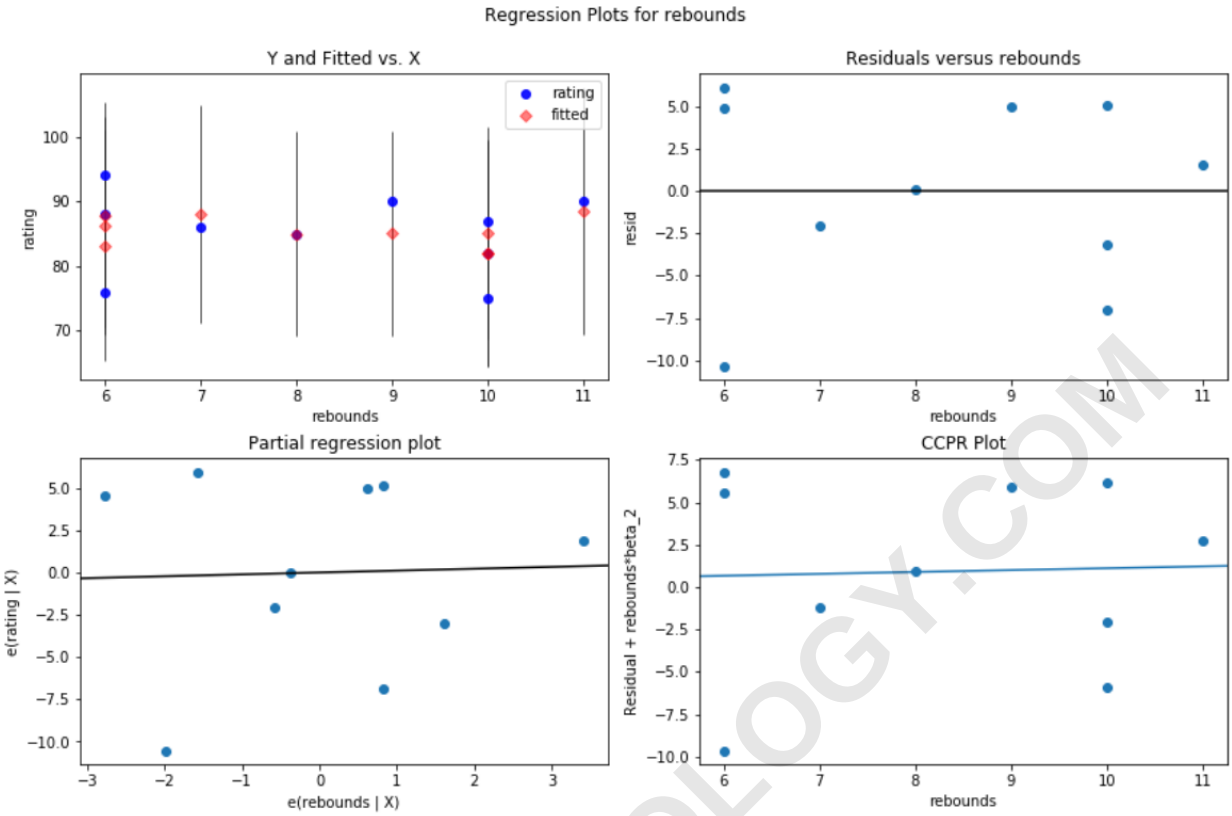


Analyzing Individual Predictor Residual Plots in MLR (Rebounds)

Finally, we generate the corresponding residual plot for the second predictor, *rebounds*. This step is crucial for completing the diagnostic check across all independent variables in the MLR model, ensuring the robustness of the overall statistical estimation.

#create residual vs. predictor plot for 'assists'
fig = plt.figure(figsize=(12,8))
fig = sm.graphics.plot_regress_exog(model, 'rebounds', fig=fig)

As seen in the plot below, the residuals corresponding to *rebounds* also demonstrate a random scattering pattern centered around zero.



In both the *assists* and *rebounds* diagnostic plots, the residuals appear to be randomly scattered around zero. This visual confirmation ensures that the assumption of homoscedasticity is met for both predictors, meaning the variance of the error terms is constant throughout the model. Consequently, we can proceed with interpreting the results of the multiple linear regression model with confidence in the validity of its underlying statistical assumptions.