Evaluating Linear Regression Models: A Practical Guide to Residual Plot Analysis in Python

Authored by

Mohammed looti

November 7, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Evaluating Linear Regression Models: A Practical Guide to Residual Plot Analysis in Python*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=12665

A Residual Plot is a fundamental diagnostic tool in statistics, specifically designed to help practitioners evaluate the appropriateness and validity of a fitted Linear Regression model. This visualization plots the fitted values (the predictions made by the model) against the corresponding Residuals (the difference between the observed and predicted values). Understanding this relationship is crucial for ensuring the reliability of statistical inferences drawn from the model.

This type of plot is often used to assess whether or not a linear relationship is appropriate for a given dataset and, critically, to check for violations of core regression assumptions, particularly the assumption of constant error variance, known as homoscedasticity. Detecting patterns in the residuals--such as funnel shapes or curves--alerts the analyst to potential model misspecification.

This comprehensive tutorial explains how to create, interpret, and leverage a residual plot for both simple and multiple linear regression models using the powerful statistical capabilities available in Python, focusing on the Statsmodels library for robust diagnostics.

Setting Up the Environment and Sample Data in Python

To demonstrate the creation of residual plots, we must first prepare our environment and load a sample dataset. We rely on standard Python libraries such as NumPy for numerical operations, Pandas for efficient data manipulation, and Matplotlib for plotting capabilities. The dataset we will use describes the attributes of 10 hypothetical basketball players, where we intend to model the relationship between performance statistics and an overall player rating.

The process begins by importing the necessary libraries and constructing the DataFrame. This preliminary step ensures that our data is correctly structured and ready for the statistical modeling phase. The resulting structure provides clear variables such as 'rating', 'points', 'assists', and 'rebounds', which will serve as our response and predictor variables in the subsequent regression analysis.

The following code block initializes the data structure and provides a view of the dataset we will be analyzing:

```
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

Residual Plot for Simple Linear Regression

Our first modeling exercise involves fitting a Simple Linear Regression model, which establishes a relationship between a single predictor and the response variable. For this example, we will suppose that we want to predict the player's overall *rating* using only the *points* scored, setting *rating* as the response variable and *points* as the predictor variable. This initial analysis serves as a straightforward application of OLS regression before we introduce the complexity of multiple predictors.

To execute the regression and obtain the necessary diagnostic information, we utilize the `ols` function from the Statsmodels library, specifying the formula `rating ~ points`. The model is then fitted to the DataFrame. This step calculates the coefficients and provides the foundation for generating the residuals that we need to visualize.

The code below imports the necessary plotting and statistical modules and then fits the model, followed by printing the summary output (though the summary itself is not displayed here, the command is retained for completeness):

```
#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())

Generating the Residual Plots for Diagnostics

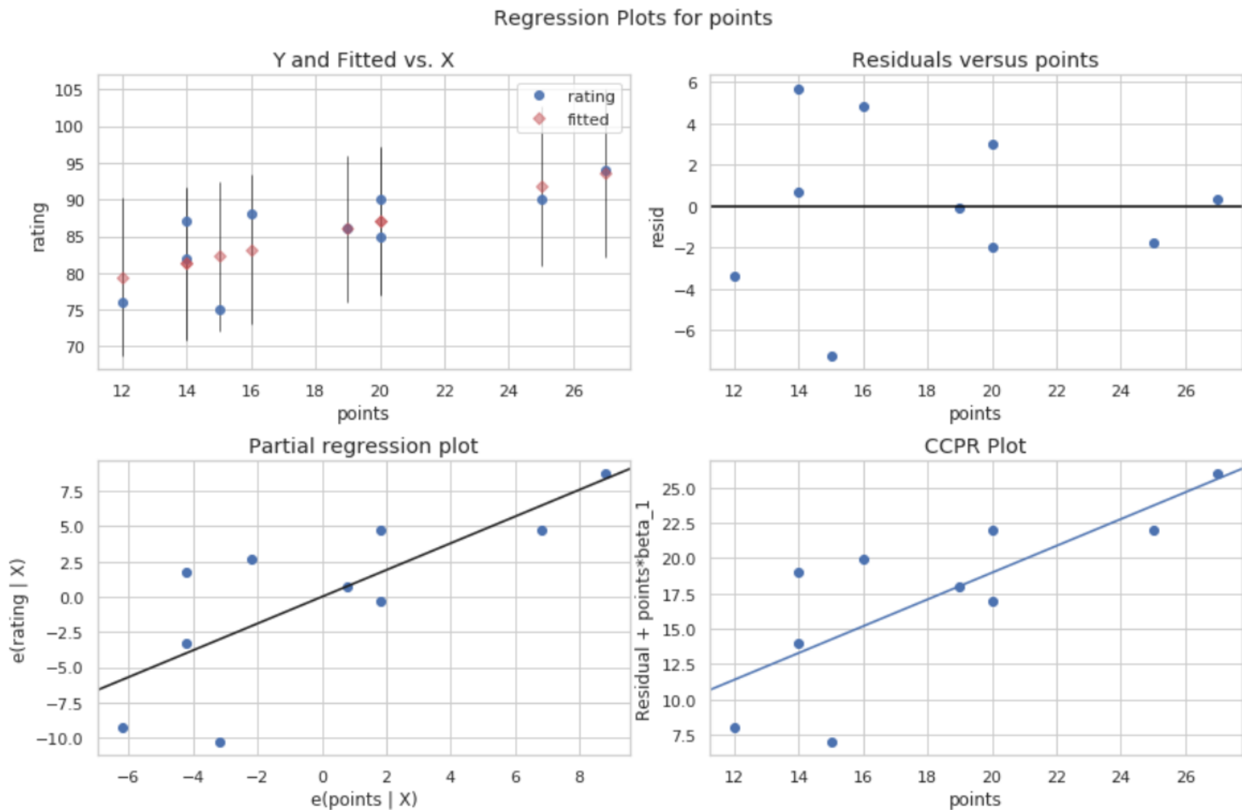
Once the model has been fitted, we can generate the diagnostic visualizations. We create a residual vs. fitted plot--along with several other useful diagnostic graphs--by using the `plot_regress_exog` function from the Statsmodels graphics module. This function is particularly valuable because it generates a suite of four plots tailored to assess the relationship between the residuals and a specific predictor variable.

We must specify the fitted model (`model`) and the predictor variable we are examining (`'points'`). By setting the figure size, we ensure the plots are clearly legible for interpretation. The resulting output provides a visual snapshot of how well the model satisfies key assumptions regarding the errors associated with the 'points' variable.

The following code generates the diagnostic plots for the predictor variable *points*:

#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 Output

The `plot_regress_exog` function generates four distinct plots. The plot located in the top right corner is the most critical for our assumption checks: the residual vs. fitted plot. In this visualization, the x-axis displays the actual values for the predictor variable (*points*), and the y-axis represents the corresponding Residuals (the difference between the observed and predicted ratings).

The ideal scenario for a valid linear model is a pattern of random scatter centered around the zero line on the y-axis. A random scatter indicates that the model has captured the underlying linear relationship effectively, and there is no systematic bias remaining in the errors. If we were to observe a curved pattern, it would suggest non-linearity; if we saw a widening or narrowing of the scatter (a funnel shape), it would indicate a violation of homoscedasticity.

Since the residuals in the plot above appear to be randomly scattered around the horizontal zero line, this is a strong indication that the assumption of homoscedasticity holds true. The absence of a systematic pattern or structure suggests that Heteroscedasticity (non-constant variance of errors) is not a problem associated with the predictor variable *points* in this Linear Regression model, validating our choice of a simple linear fit.

Extending the Analysis to Multiple Linear Regression

Next, we expand our model to Multiple Linear Regression by incorporating additional predictor variables. Suppose we fit a model using *assists* and *rebounds* as predictor variables, still aiming to model the *rating* as the response variable. While the overall model diagnostics (such as the R-squared value) inform us about explanatory power, the residual plots are essential for checking the internal validity assumptions for each individual predictor.

We again use the `ols` function, but this time, the formula explicitly includes both predictors: `rating ~ assists + rebounds`. After fitting this new, more complex model, we must sequentially examine the relationship between the model's residuals and each predictor variable to ensure that the constant variance assumption holds true across all dimensions of the predictor space.

The following code fits this multiple regression model using the two new variables:

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

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

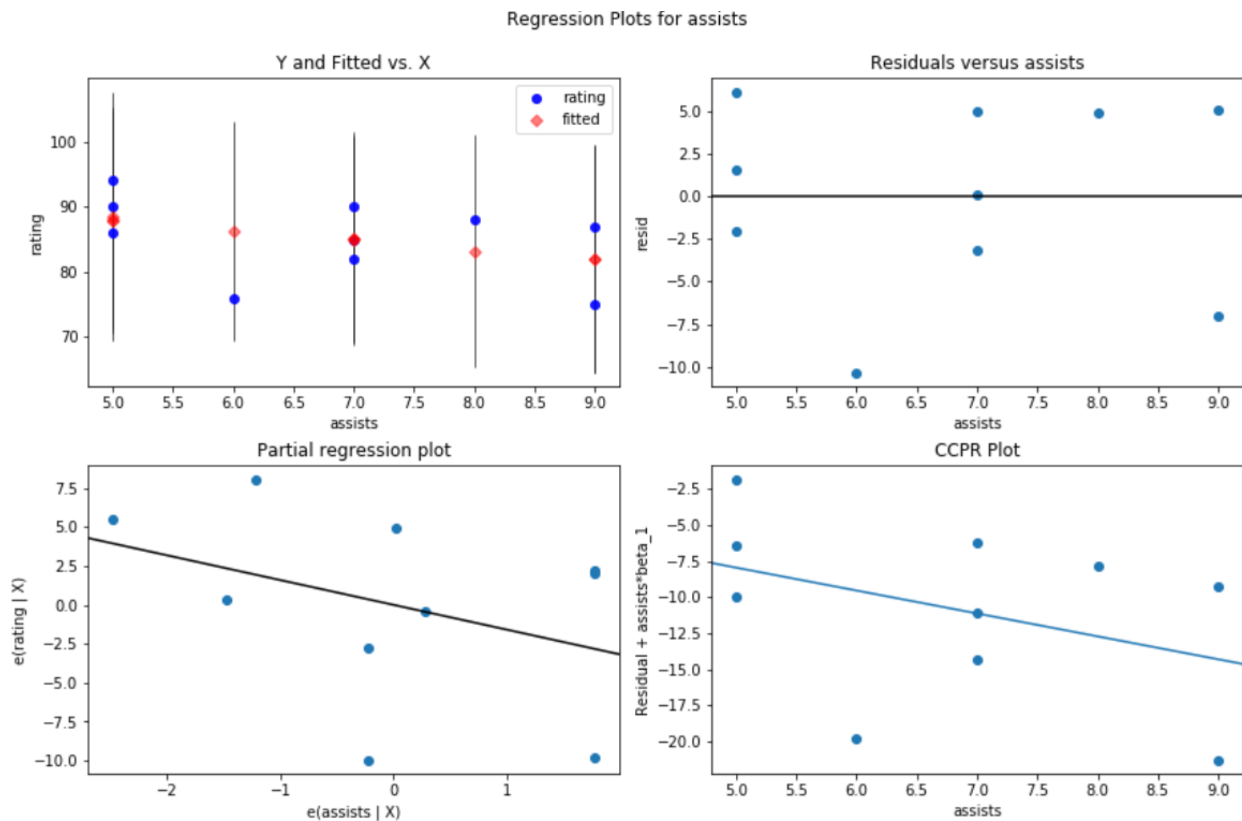
Evaluating Individual Predictor Residuals

In the context of multiple regression, we generate separate sets of diagnostic plots for each included predictor. This allows us to isolate potential issues, such as non-linearity or Heteroscedasticity, that might be specific to one variable and masked by the presence of others. We repeat the use of `sm.graphics.plot_regress_exog`, changing the specified predictor each time.

For example, we first examine the relationship between the residuals of the model and the *assists* variable. The visualization provides the necessary detail to confirm whether the variance of the errors is stable as the number of assists increases or decreases.

Here is the code and the resulting residual vs. predictor plot for *assists*:

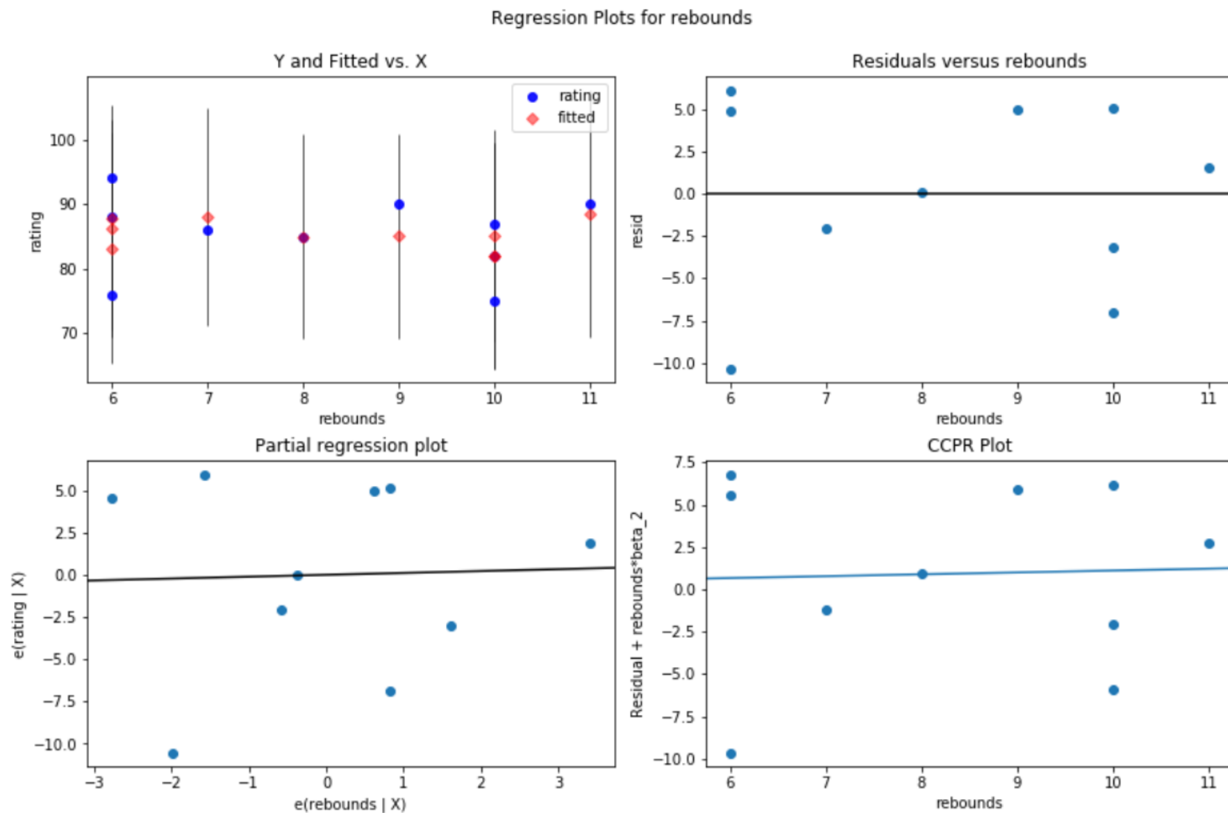
```
#create residual vs. predictor plot for 'assists'
fig = plt.figure(figsize=(12,8))
fig = sm.graphics.plot_regress_exog(model, 'assists', fig=fig)
```



Similarly, we must perform the diagnostic check for the *rebounds* predictor. This second visualization is critical to confirm that the model's assumptions hold across all inputs. If one plot showed a non-random pattern while the other did not, it would suggest that the issue is localized to the variable exhibiting the structure, potentially requiring transformation or re-evaluation of that specific predictor.

The code and resulting plot for *rebounds* are shown below:

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



In both diagnostic visualizations--the residual vs. *assists* plot and the residual vs. *rebounds* plot--the residuals appear to be randomly scattered around the zero line. This consistent random pattern across all predictors in the model is a strong indicator that homoscedasticity is maintained and that the model is adequately specified. Such results confirm that the Multiple Linear Regression model is robust and suitable for inferential analysis concerning player ratings based on assists and rebounds.