How to calculate Residual Sum of Squares in Python?

Authored by

stats writer

December 13, 2025

The Residual Sum of Squares (RSS) is one of the most fundamental metrics in statistical analysis, particularly within the field of regression models. It serves as a crucial measure of the discrepancy between the observed data and the values predicted by the model, essentially quantifying the overall error of the fit. A lower RSS value generally signifies that the chosen model explains the variance in the response variable more effectively, demonstrating superior predictive power and tighter alignment with the underlying data structure. Understanding how to accurately calculate and interpret this metric is paramount for anyone involved in statistical modeling and machine learning.

While statistical software packages often compute the RSS automatically, it is highly beneficial for data scientists to grasp the mathematical principles and the practical implementation steps, especially when working with flexible programming languages like Python. Although libraries like NumPy or scikit-learn offer functions for calculating squared errors, in the context of advanced linear modeling using `statsmodels`, the RSS can be accessed directly as a property of the fitted model object, streamlining the workflow considerably. This comprehensive tutorial will guide you through the process of defining the data, fitting a multiple linear regression, and extracting the accurate RSS value using powerful Python libraries.

Understanding the Concept of Residuals

At the core of the Residual Sum of Squares calculation lies the concept of a residual. A residual represents the vertical distance between a specific observed data point in your dataset and the corresponding point on the fitted regression line or surface. In essence, it is the error for that particular observation--the portion of the response variable that the model failed to explain. Formally, a residual is defined as the difference between the actual (observed) value and the predicted value in a regression model, reflecting the imperfection inherent in any statistical estimation.

The primary goal of fitting a linear regression model, especially through the Ordinary Least Squares (OLS) method, is to minimize these errors. If a model fits the data perfectly, all residuals would be zero. However, in real-world scenarios, variation is inevitable, and we seek the line that produces the smallest average error. Because errors can be positive (underestimation) or negative (overestimation), simply summing them would result in cancellation, often yielding a total sum close to zero, which is not an effective measure of overall model error.

To overcome the cancellation issue and ensure that larger errors contribute more significantly to the overall penalty, statisticians square each residual before summing them. This squaring process guarantees that all contributions are positive and penalizes larger deviations much more heavily than smaller ones. This mechanism is crucial because minimizing the sum of squared errors--the RSS--is exactly what the OLS estimation procedure is designed to accomplish. The RSS is, therefore, the fundamental quantity being minimized during the model fitting phase, providing the

best possible linear unbiased estimates (BLUE) under standard assumptions.

It is calculated as:

Residual = Observed value - Predicted value

The Mathematical Definition of Residual Sum of Squares (RSS)

Understanding the specific formulation of the Residual Sum of Squares (RSS) provides clarity on how model performance is mathematically judged. Once the individual residuals are calculated for every data point in the sample, the next step is the aggregation of these squared errors. This aggregation results in a single, scalar value that summarizes the total unexplained variance. This metric is also sometimes referred to as the Sum of Squared Errors (SSE) or Sum of Squared Residuals (SSR), although the terminology can sometimes overlap with the Sum of Squares due to Regression. For clarity in the context of model fit assessment, we will adhere to RSS.

The critical insight provided by RSS is its direct relationship to model quality: the smaller the RSS, the closer the predicted values are to the observed values, indicating a superior fit to the training data. Conversely, a very high RSS suggests that the model is performing poorly, and the estimated relationship does not adequately capture the underlying patterns within the data. It is important to note that RSS is scale-dependent, meaning it cannot be used alone to compare models across different datasets or models predicting vastly different variables.

The formula for the Residual Sum of Squares solidifies this relationship:

Residual sum of squares = Σ(ei)2

where the components are defined precisely:

Σ: This is the uppercase Greek letter Sigma, representing the operation of summation, indicating that we sum the resulting values across all observations.
ei: This term denotes the ith residual, which is the difference between the observed response value (Yi) and the predicted response value (Y?i) for the ith data point.

This formula confirms that every data point contributes to the total error based on how far it deviates from the model's prediction, with the squaring operation ensuring that all deviations, regardless of direction, increase the total sum. The lower the resulting RSS value, the better a model is deemed to fit the dataset, serving as a fundamental benchmark for model evaluation in regression analysis.

Setting Up the Python Environment for Regression

To demonstrate the calculation of RSS, we utilize Python, leveraging its robust ecosystem of statistical and data manipulation libraries. Before performing any analysis, it is essential to prepare the environment by importing the necessary libraries--specifically, **Pandas** for efficient data handling and **Statsmodels** for statistical modeling. These libraries are industry standards for data science workflows and provide specialized functions that make regression analysis intuitive and reliable. We will ensure our data is structured correctly within a Pandas DataFrame, which is the preferred format for statistical operations in Python.

For this tutorial, we will work with a hypothetical dataset designed to model student performance. Our scenario involves analyzing how two predictor variables--the number of hours spent studying and the total number of preparatory exams taken--influence the final exam score received by a group of students. By using multiple predictors, we are setting up a multiple linear regression scenario, which is common in real-world data analysis and provides a richer context for calculating the Residual Sum of Squares. The initial step focuses entirely on inputting and structuring this data.

The data preparation phase involves creating a DataFrame where each row corresponds to a student observation, and columns represent the variables: 'hours', 'exams', and 'score'. Proper data entry and verification are critical, as errors in the source data will propagate throughout the modeling process. The following Python code snippet initiates this process, utilizing the Pandas library to construct the structured dataset for 14 different students.

Step 1: Enter the Data

We define the dataset containing observations on study hours, preparatory exams taken, and the resulting exam score for 14 students. This data will be structured using the Pandas library, which is fundamental for handling tabular data in Python:

import pandas as pd

#create DataFrame
df = pd.DataFrame({'hours': ,
'exams': ,
'score': })

After executing this code, the data is ready for the subsequent statistical analysis. This DataFrame, `df`, now holds all the necessary variables to define and fit our multiple linear regression model, with 'score' designated as the dependent variable and 'hours' and 'exams' serving as the independent

variables.

Fitting the Ordinary Least Squares (OLS) Model

The next crucial step is to estimate the parameters of the linear relationship between our predictors and the response variable. We employ the **Ordinary Least Squares (OLS)** method, which is the standard technique for linear regression. OLS works by minimizing the Residual Sum of Squares (RSS), thereby finding the unique set of coefficients that best fits the hyperplane to the data. We utilize the powerful `statsmodels` library in Python, which is favored for its statistical rigor and its ability to provide detailed model diagnostics comparable to those found in specialized statistical software packages.

Before fitting the model, we must properly define our variables and, critically, ensure the inclusion of an intercept term. In `statsmodels`, unlike some other libraries, the user must explicitly add a constant term to the predictor variables if they wish the regression line to have an intercept (which is almost always required in standard OLS). The `sm.add_constant(x)` function handles this requirement automatically, transforming our predictor matrix to include a column of ones, thus allowing the model to estimate the baseline score when hours and exams are zero.

The `OLS()` function is instantiated with the response variable (`y`) and the augmented predictor variables (`x`). The subsequent `.fit()` method executes the actual minimization of the RSS, calculating the optimal coefficients (Beta values). The final step in this stage involves printing the model summary, which provides exhaustive detail regarding the model's performance, including coefficients, standard errors, T-statistics, P-values, and overall fit statistics like R-squared, allowing for comprehensive diagnostic assessment.

Step 2: Fit the Regression Model

We will use the OLS() function from the `statsmodels` library to perform the regression. We define "hours" and "exams" as the independent variables and "score" as the dependent variable, following the essential practice of adding a constant for the intercept:

import statsmodels.api as sm

#define response variable
y = df

#define predictor variables
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: 0.722
Model: OLS Adj. R-squared: 0.671
Method: Least Squares F-statistic: 14.27
Date: Sat, 02 Jan 2021 Prob (F-statistic): 0.000878
Time: 15:58:35 Log-Likelihood: -41.159
No. Observations: 14 AIC: 88.32
Df Residuals: 11 BIC: 90.24
Df Model: 2
Covariance Type: nonrobust
==========
===
coef std err t P>|t|
---
const 71.8144 3.680 19.517 0.000 63.716 79.913
hours 5.0318 0.942 5.339 0.000 2.958 7.106
exams -1.3186 1.063 -1.240 0.241 -3.658 1.021
==========
===
Omnibus: 0.976 Durbin-Watson: 1.270
Prob(Omnibus): 0.614 Jarque-Bera (JB): 0.757
Skew: -0.245 Prob(JB): 0.685
Kurtosis: 1.971 Cond. No. 12.1
==========
===

Calculating RSS Directly Using Statsmodels

One of the primary advantages of utilizing the `statsmodels` library for regression is the ease with which key statistical metrics can be extracted immediately after model fitting. Because the OLS

algorithm is inherently designed to minimize the Residual Sum of Squares (RSS), this value is computed and stored as a fundamental attribute of the fitted model object. In `statsmodels`, this attribute is conveniently named `ssr`, which stands for Sum of Squared Residuals--an equivalent term for RSS.

Accessing the RSS is achieved simply by calling the `.ssr` property on the `model` object that resulted from the `.fit()` operation. This streamlined approach eliminates the need for manual calculation, which would otherwise involve obtaining the predicted values, subtracting them from the actual values to find the residuals, squaring each residual, and finally summing the resulting values across all 14 observations. The efficiency of using the built-in property is highly valuable in iterative model development and large-scale data analysis projects in Python.

The immediate availability of the RSS allows data scientists to quickly verify the model's performance. For instance, if you were to compare two different models--perhaps one using 'hours' and 'exams', and another using only 'hours'--you could directly compare their respective `.ssr` values. The model yielding the smaller RSS is the one that has achieved a tighter fit to the data, assuming all other structural assumptions remain constant. This ease of access makes RSS a primary benchmark metric.

Step 3: Calculate the Residual Sum of Squares

We retrieve the residual sum of squares directly from the fitted `model` object using the `.ssr` attribute:

print(model.ssr)

293.25612951525414

Upon execution, the residual sum of squares turns out to be **293.256** (when rounded to three decimal places). This figure quantifies the total unexplained variance in the exam scores after accounting for the influence of study hours and preparatory exams.

Interpreting the Significance of the RSS Value

The calculated Residual Sum of Squares (RSS) of 293.256 is not interpreted in isolation but rather in context. As previously established, this numerical value represents the sum of the squared errors, meaning it is a measure of the variability in the dependent variable (score) that is **not** captured by the linear relationship defined by the predictors (hours and exams). A value of 293.256 seems relatively small for a score range between 69 and 96, suggesting a reasonably good fit, which is corroborated by the high R-squared value (0.722) displayed in the OLS summary table.

In practical terms, the primary utility of the RSS is for comparing competing models that have been fitted to the exact same dataset using the same number of observations. For instance, if a second model, perhaps one that includes a quadratic term for 'hours', resulted in an RSS of 250, we would objectively conclude that the second model provides a tighter fit to the data because it has a smaller total unexplained variance. It is crucial to remember that minimizing the RSS is the mechanical objective of the OLS methodology, making this metric an essential yardstick in the model selection process.

However, analysts must exercise caution when using RSS, particularly when comparing models with a different number of predictors. Adding more independent variables will almost always reduce the RSS, even if those variables are statistically insignificant, simply because the model gains more flexibility. Therefore, when comparing models of different complexities, adjusted metrics like Adjusted R-squared, AIC (Akaike Information Criterion), or BIC (Bayesian Information Criterion) are typically preferred, as they incorporate a penalty for the increased number of parameters utilized in the model.

Alternative Calculation: Manual and NumPy Implementation

While the `.ssr` attribute in `statsmodels` provides the most straightforward method for retrieving the RSS, it is highly instructive to understand how the calculation can be performed manually or using other fundamental Python libraries like NumPy. This approach reinforces the mathematical definition of RSS and allows for calculation in environments where a specific regression library like `statsmodels` might not be the primary tool, such as when using scikit-learn for modeling or when dealing with custom statistical algorithms.

To calculate the RSS manually, we first need the predicted values (Y?). In the context of our fitted `statsmodels` model, these predictions can be accessed via the `model.predict(x)` function. Once we have the predictions, we can calculate the residuals by subtracting the predicted values from the actual values (`y - model.predict(x)`). Finally, we square these residuals and sum them up using NumPy operations, or even Python's built-in summation functions if preferred.

A common alternative, especially in machine learning contexts, involves utilizing the NumPy library. We can calculate the residuals and then compute the sum of squares. Furthermore, libraries like scikit-learn often provide a dedicated function for Mean Squared Error (MSE), which is directly related to RSS. MSE is calculated as RSS divided by the number of observations (or degrees of freedom). Therefore, multiplying the MSE by the number of observations allows us to recover the original Residual Sum of Squares. This demonstrates the versatility of the Python data science stack when tackling core statistical computations.