How to Easily Perform Logistic Regression with Statsmodels

Authored by

stats writer

November 27, 2025

Logistic regression is a fundamental statistical analysis technique utilized primarily to model the relationship between one or more independent variables and a categorical dependent variable, specifically when the outcome is binary (e.g., success/failure, pass/fail, true/false). Unlike standard linear regression, which predicts a continuous outcome, logistic regression estimates the probability of an event occurring by fitting data to a sigmoid function. This approach makes it indispensable in fields ranging from medical diagnostics to finance and social science.

To execute this powerful analysis efficiently in Python, data scientists frequently rely on the Statsmodels package. Statsmodels is designed to provide comprehensive tools for exploring data, estimating statistical models, and performing statistical tests. This guide will walk you through the precise steps required to perform logistic regression using the Statsmodels API, covering everything from data preparation to fitting the model, interpreting the results, and evaluating its overall performance using metrics like Pseudo R-squared.

The Statsmodels Framework for Statistical Modeling

The Python ecosystem offers several robust libraries for statistical modeling, and Statsmodels is highly valued for its strong alignment with traditional statistical software packages like R and Stata. The module provides an extensive collection of functions and classes that facilitate the fitting of various statistical models, including generalized linear models (GLMs), which encompass logistic regression.

Using Statsmodels often involves following a standardized workflow: defining the model using formula notation (similar to R), fitting the model to the data using maximum likelihood estimation, and then generating a detailed summary output that includes all necessary diagnostic statistics. This approach ensures transparency and ease of interpretation for those familiar with classical statistics and rigorous hypothesis testing.

The following detailed walkthrough demonstrates how to implement logistic regression using the functions provided within the Statsmodels package, focusing on a practical example of predicting educational outcomes based on study variables.

Step 1: Preparing and Structuring the Dataset (Data Creation)

Before fitting any statistical model, the initial and most critical step is structuring the input data appropriately. For this demonstration, we will use the pandas DataFrame structure, which is the standard mechanism for data handling and manipulation in Python. Our synthetic dataset aims to predict whether a student will pass an exam based on two distinct predictors: the time spent studying and the method of study employed.

We define three crucial variables for our analysis, ensuring the dependent variable is correctly encoded as a binary outcome:

Hours Studied: An **integer value** representing the total time dedicated to preparation. This is a continuous quantitative independent variable.
Study Method: A categorical variable (Method A or B) indicating the approach used by the student. This is a qualitative independent variable that will be handled automatically by Statsmodels using dummy coding.
Exam Result: A **binary dependent variable** (Pass coded as 1, Fail coded as 0) that we aim to predict.

The overall modeling objective is to use the 'hours studied' (quantitative) and 'study method' (qualitative) variables to predict the probability of the 'exam result' being a pass. The creation of the pandas DataFrame is necessary to structure this data efficiently for the Statsmodels formula API, which requires well-organized tabular data.

The following code snippet illustrates the process of generating this DataFrame and visualizing the first few observations:

import pandas as pd

#create DataFrame
df = pd.DataFrame({'result': ,
'hours': ,
'method': })

#view first five rows of DataFrame
df.head()

result hours method
0 0 1 A
1 1 2 A
2 0 2 A
3 0 2 B
4 0 3 B

Step 2: Implementing the Logit Model in Statsmodels

With the data correctly prepared, the next step involves fitting the logistic regression model itself. In Statsmodels, this is achieved using the `logit()` function, which stands for Logistic Model. We access this function through the `statsmodels.formula.api` submodule, which streamlines the

modeling process by allowing input through R-style formulas.

The formula `result ~ hours + method` specifies that 'result' is the dependent variable predicted by 'hours' and 'method'. The model fitting process employs the **Maximum Likelihood Estimation (MLE)** method, which is mathematically robust and standard for discrete choice models like logistic regression. MLE iteratively searches for the coefficient values that maximize the likelihood of observing the actual pass/fail outcomes given the predictors.

Once the model object is created and the `.fit()` method is executed, we generate a comprehensive summary. This summary provides technical details about the estimation process (like convergence status and iteration count) alongside the critical statistical results, including coefficient estimates and model fit diagnostics.

import statsmodels.formula.api as smf

#fit logistic regression model using the formula API
model = smf.logit('result ~ hours + method', data=df).fit()

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

Optimization terminated successfully.
Current function value: 0.557786
Iterations 5
Logit Regression Results
==========
===
Dep. Variable: result No. Observations: 20
Model: Logit Df Residuals: 17
Method: MLE Df Model: 2
Date: Mon, 22 Aug 2022 Pseudo R-squ.: 0.1894
Time: 09:53:35 Log-Likelihood: -11.156
converged: True LL-Null: -13.763
Covariance Type: nonrobust LLR p-value: 0.07375
==========
====
coef std err z P>|z|

Intercept -2.1569 1.416 -1.523 0.128 -4.932 0.618
method 0.0875 1.051 0.083 0.934 -1.973 2.148
hours 0.4909 0.245 2.002 0.045 0.010 0.972

==========

Step 3: Interpreting the Regression Output (Coefficients and Significance)

The most important section of the output is the coefficients table. The values listed in the **coef** column represent the estimated change in the log odds of the dependent variable (passing the exam) for a one-unit change in the respective predictor, assuming all other variables remain constant. These log odds can be exponentiated to obtain the odds ratio, which is often easier to interpret.

Analyzing the coefficients from our fitted model provides specific insights:

Study Method (method): The coefficient of **0.0875** indicates that using study method B is associated with an average increase of 0.0875 in the log odds of passing the exam, relative to the baseline method (Method A). Since this value is close to zero, the practical difference between the two methods appears negligible in this sample.
Hours Studied (hours): The coefficient of **0.4909** reveals that for every additional hour studied, the average log odds of passing the exam increase by 0.4909. This strong positive value suggests a clear relationship where dedicating more time significantly increases the likelihood of success.

To assess the statistical reliability of these findings, we examine the **P>|z|** column, which contains the p-values resulting from the Wald Z-test for each coefficient. This test determines if the coefficient is significantly different from zero. We typically compare the p-value against a predetermined significance level ($alpha$), often set at 0.05.

Evaluating the statistical significance of our predictors:

Study Method: The p-value for `method` is **0.934**. Since this value is substantially greater than 0.05, we conclude that we cannot reject the null hypothesis. Therefore, there is no statistically significant evidence suggesting that the choice of study method (A vs. B) affects the probability of passing the exam in this model.
Hours Studied: The p-value for `hours` is **0.045**. Since $0.045 < 0.05$, we reject the null hypothesis. This confirms a statistically significant and positive relationship between the number of hours studied and the likelihood of passing the exam.

Step 4: Assessing Overall Model Utility (Pseudo R-Squared and LLR Test)

To assess the global quality of the fitted model, we turn to metrics designed specifically for likelihood-based models. Unlike linear regression, which uses R-squared, logistic regression relies on measures that quantify how much better the model is compared to a model with no predictors

(the null model).

We analyze two critical metrics provided in the Statsmodels summary output for overall model assessment:

1. Pseudo R-Squared

The **Pseudo R-squared** value is utilized as an analogous measure to the R-squared in linear modeling. It is calculated based on the maximized log-likelihood functions of the full model and the null model. This metric gauges the proportion of improvement in model fit achieved by including the predictors.

The most common formulation, the McFadden Pseudo R-squared (which is often reported by Statsmodels), is derived from the log-likelihood values: $R^2_{McFadden} = 1 - (LL_{Full} / LL_{Null})$. This value ranges from 0 to 1, with values closer to 1 indicating a much stronger model fit relative to the null model. However, note that a "good" Pseudo R-squared value in logistic regression is typically much lower than in linear regression, often ranging from 0.2 to 0.4 for an excellent fit.

In this example, the Pseudo R-squared value is **0.1894**. This value suggests a modest improvement over the null model. While it indicates that the predictor variables are contributing to the model's explanatory power, it simultaneously suggests that the variables included do not fully account for the total variability in the exam results, implying potential omitted variable bias or complex underlying relationships.

2. LLR p-value (Likelihood Ratio Test)

The **LLR p-value** is generated by the Likelihood Ratio Test (LRT), which provides a global test of model significance. This test evaluates the overall hypothesis that all independent variables simultaneously have zero effect on the outcome (i.e., comparing the Log-Likelihood of the full model to the Log-Likelihood of the null model).

If the LLR p-value is below the chosen significance threshold (e.g., $alpha = 0.05$), we conclude that the fitted model, as a whole, is significantly better at predicting the response variable than a model without any predictors. This confirms the model's overall utility.

In this example, the LLR p-value is **0.07375**. If we adhere strictly to the traditional 0.05 significance level, we would conclude that the model is not statistically useful overall, as the p-value slightly exceeds the threshold. However, given that it is close to the threshold (and one predictor is individually significant), this result warrants caution. It reinforces the finding from the Pseudo R-squared that the model, while guided by the significant 'hours' variable, has limited explanatory capacity in this specific dataset.

Conclusion: Summarizing Key Findings and Further Work

This tutorial successfully demonstrated the rigorous methodology for performing logistic regression using the Statsmodels package in Python. We emphasized crucial steps, including defining the binary dependent variable, utilizing the formula API for model fitting, and interpreting the output in terms of p-values and coefficients.

Our analysis confirmed a strong, statistically significant link between the number of hours a student studies and their probability of passing the exam. However, the specific study method did not prove to be a significant factor. Furthermore, the evaluation metrics, specifically the low Pseudo R-squared (0.1894) and the marginally significant LLR p-value (0.07375), suggest that while the 'hours' variable is a powerful predictor, the model's overall predictive power remains modest, and researchers should consider incorporating additional explanatory variables to enhance its performance.

Mastering this workflow provides the foundational expertise for conducting more complex statistical analyses and predictive modeling tasks within the Python data science environment.