How do I Find a P-Value from a t-Score in Python?

Authored by

stats writer

December 24, 2025

In the field of data science and classical statistics, calculating the p-value associated with a computed t-score is a fundamental requirement for formal decision-making. The p-value serves as a critical measure of statistical significance, quantifying the probability of observing data as extreme as, or more extreme than, the observed data, assuming the null hypothesis is true. When working within the Python ecosystem, the efficient and reliable way to perform this calculation is through the utilization of the powerful SciPy library, specifically its statistical module.

This process requires two key inputs: the calculated t-score (the observed test statistic) and the corresponding number of degrees of freedom (df), which defines the specific shape of the t-distribution. By feeding these parameters into the appropriate function within the **scipy.stats** sub-module, practitioners can instantaneously retrieve the precise p-value necessary to conclude their hypothesis test. This detailed guide demonstrates exactly how to implement these calculations across various types of tests--left-tailed, right-tailed, and two-tailed--ensuring clarity and statistical rigor in your analyses.

Understanding the Relationship Between T-Scores and P-Values

In statistical inference, the primary goal of a hypothesis test is to determine whether there is enough evidence in a sample to reject a predefined assumption about a population, known as the null hypothesis. The t-score is a standardized value representing how many standard errors an observed sample mean is from the hypothesized population mean. A larger absolute value of the t-score suggests that the observed difference is less likely to be due to random chance, pushing the analysis toward rejecting the null hypothesis.

The crucial next step is calculating the p-value associated with this computed t-score. The p-value essentially provides the probability that, if the null hypothesis were true, we would observe a test statistic (like our t-score) as extreme or more extreme than the one calculated from our sample data. If this p-value falls below a predetermined significance level (often denoted as $alpha$, typically 0.05), we conclude that the evidence strongly contradicts the null assumption, leading us to reject the null hypothesis in favor of the alternative hypothesis.

Therefore, knowing the precise p-value is essential for making a formal statistical decision. The magnitude of the t-score and the corresponding degrees of freedom dictate the exact location on the Student's t-distribution curve, and calculating the area under the curve beyond that point provides the required p-value. This calculation, while complex mathematically, is handled effortlessly using specialized functions available in modern statistical programming libraries like SciPy.

Introducing the SciPy Statistical Function

To efficiently find the p-value associated with any given t-score in Python, we leverage the capabilities of the **scipy.stats** module. Specifically, we utilize the `t.sf()` function. This function stands for the Survival Function, which is equivalent to 1 minus the Cumulative Distribution Function (CDF). In the context of probability, the survival function calculates the area under the probability density curve starting from a specific point (our t-score) out to infinity, providing the required tail probability.

The primary reason we employ the `.sf()` (survival function) is that it directly calculates the area in the tail of the distribution, which is precisely the definition of the p-value for one-tailed tests. Moreover, to accommodate both positive and negative t-scores (which determine the left or right tail), we typically pass the **absolute value** of the t-score to the function. This ensures that we are always calculating the probability associated with the magnitude of the deviation, regardless of direction.

The general syntax for using the survival function from the Student's t-distribution within SciPy is structured as follows:

scipy.stats.t.sf(abs(x), df)

where the required parameters are clearly defined:

x: Represents the observed t-score, passed as its absolute value.
df: Represents the degrees of freedom, calculated based on the sample size(s) used in the hypothesis test.

The following examples illustrate how to find the p-value associated with a t-score for a left-tailed test, right-tailed test, and a two-tailed test, demonstrating the minor adjustments required for each scenario to ensure accurate statistical reporting.

Calculating the P-Value for a Left-Tailed Test

A left-tailed hypothesis test is used when the alternative hypothesis suggests that the true population parameter is **less than** the value stated in the null hypothesis. For instance, testing if a new fertilizer results in a yield significantly lower than the standard method. In such cases, a negative t-score (falling far into the left tail of the distribution) supports the alternative hypothesis.

Suppose we have conducted a test and calculated a t-score of **-0.77**, and the sample size dictated that the corresponding degrees of freedom (df) are **15**. Since the t-distribution is symmetrical around zero, the probability of observing a score less than -0.77 is identical to the probability of observing a score greater than +0.77. We can therefore utilize the `t.sf()` function with the

absolute value of the t-score to find the area in the relevant tail.

The calculation is performed in Python as follows:

import scipy.stats

#find p-value. Use abs() to ensure calculation is done on the right side of the distribution for sf()
scipy.stats.t.sf(abs(-.77), df=15)

0.2266283049085413

The resulting p-value is approximately **0.2266**. If we establish a common significance level ($alpha$) of 0.05, we must compare the calculated p-value to this threshold. Since 0.2266 is significantly greater than 0.05, we lack sufficient statistical evidence to reject the null hypothesis of our test. This indicates that the observed result of -0.77 is not unusual enough to warrant a definitive conclusion about the population.

Interpreting Results for a Right-Tailed Test

A right-tailed hypothesis test is conducted when the alternative hypothesis posits that the true population parameter is **greater than** the value specified by the null hypothesis. For example, testing if a new marketing strategy yields an average sales increase that is significantly higher than the previous average. This scenario typically results in a positive t-score.

Consider a scenario where the calculated t-score is **1.87**, based on a sample that provides degrees of freedom (df) equal to **24**. Since we are interested in the area above this positive t-score, which corresponds directly to the right tail, the `t.sf()` function naturally computes the exact probability we need. Because 1.87 is already positive, passing its absolute value remains 1.87, simplifying the call.

The Python computation proceeds as follows, yielding the tail probability:

import scipy.stats

#find p-value for right-tailed test
scipy.stats.t.sf(abs(1.87), df=24)

0.036865328383323424

The resulting p-value is found to be **0.0368**. When utilizing a significance level ($alpha$) of 0.05, we observe that the calculated p-value (0.0368) is demonstrably less than the threshold (0.05).

This critical finding means the observed data is sufficiently improbable under the assumption of the null hypothesis, leading us to reject the null hypothesis and conclude that the alternative hypothesis (that the true mean is greater than hypothesized) is statistically supported.

Applying the Two-Tailed Test Correction

A two-tailed hypothesis test is employed when the alternative hypothesis simply states that the true population parameter is **not equal to** the hypothesized value, without specifying a direction (greater than or less than). Because the difference could manifest in either direction (positive or negative deviation from the mean), the p-value must account for both tails of the distribution.

Since the Student's t-distribution is symmetric, the probability of observing a deviation in the positive direction (the right tail) is equal to the probability of observing the same magnitude of deviation in the negative direction (the left tail). Therefore, the standard procedure for a two-tailed test involves calculating the probability in one tail using `t.sf(abs(x), df)` and then multiplying the result by two to cover both extreme regions.

Suppose we calculate a t-score of **1.24** with **22** degrees of freedom (df). To find the correct two-tailed p-value, we perform the one-tail calculation and then multiply the result by 2, as shown in the code block below:

import scipy.stats

#find p-value for two-tailed test (must multiply the single-tail result by 2)
scipy.stats.t.sf(abs(1.24), df=22)*2

0.22803901531680093

The final two-tailed p-value is calculated as **0.2280**. Comparing this result to the standard significance level of $alpha = 0.05$, we find that 0.2280 is substantially larger than 0.05. Consequently, we must fail to reject the null hypothesis of our two-tailed hypothesis test. This implies that the observed deviation represented by the t-score of 1.24 is not statistically unusual enough to claim a difference from the hypothesized mean.

For verification or analyses outside of a programmatic environment, specialized online statistical calculators can also be used to find p-values from t-scores and degrees of freedom.