How to Conduct a Two Sample T-Test in Python?

Authored by

stats writer

December 25, 2025

The two sample t-test is a fundamental tool in inferential statistics, designed to ascertain whether the unknown true population means of two independent groups are significantly different from one another. This test is crucial when analyzing experimental results or comparative studies where data collection results in two distinct, non-overlapping datasets. Leveraging the power of the SciPy library in Python, data scientists can efficiently execute this procedure using the specialized `ttest_ind()` function.

Understanding how to implement the two-sample t-test in a coding environment like Python is essential for modern data analysis. The primary output of this statistical operation includes the calculated **t-statistic** and the corresponding two-tailed p-value. These metrics provide the empirical evidence necessary to decide whether to reject or fail to reject the **null hypothesis**, thereby drawing a formal conclusion regarding the equality of the two population means.

Prerequisites and Assumptions for the T-Test

Before proceeding with the practical execution of a two sample t-test, it is critical to confirm that the underlying assumptions of the test are met. Failure to meet these assumptions can compromise the validity of the statistical conclusion. The primary assumptions for the standard (Student's) independent two-sample t-test include the independence of observations within and between the groups, the approximate normality of the data distributions, and perhaps most importantly, the homogeneity of population **variances**.

While the t-test is relatively robust against minor deviations from normality, particularly with larger sample sizes, the assumption regarding equal variances often dictates which specific version of the test must be applied--Student's t-test (equal variances assumed) or Welch's t-test (unequal variances assumed). In this guide, we will first demonstrate how to test for this assumption before selecting the appropriate test type in Python.

Setting Up the Experiment Scenario: The Plant Height Study

To provide a clear, practical illustration, we will use a common research scenario involving biological measurements. Suppose researchers are interested in determining if two distinct species of plants exhibit the same average height. To investigate this hypothesis, they meticulously collected a representative sample, obtaining 20 individual plant height measurements for each species.

The core objective of this study is to compare the population mean height (μ) for Species 1 versus Species 2. If the statistical test provides sufficient evidence against the **null hypothesis** (H0: μ1 = μ2), the researchers can confidently conclude that the mean heights are statistically different. We will now proceed through the necessary steps to conduct this analysis using the collected sample

data within a Python environment.

Step 1: Preparing and Defining the Sample Data

The initial phase of any data analysis pipeline involves structuring the data correctly. For the two-sample t-test in Python, the data for each group must be structured as separate numerical arrays. We utilize the **NumPy** library for efficient array handling and mathematical operations. Below, we define the height measurements collected from the 20 plants of Group 1 and Group 2, representing the two distinct species.

The following code block demonstrates the setup required to load the sample observations into NumPy arrays, preparing them for the statistical test function call:

import numpy as np

group1 = np.array()
group2 = np.array()

Evaluating the Assumption of Equal Variances

Before running the primary t-test, we must address the assumption of equal population **variances**. If the population variances are deemed unequal, we must apply Welch's t-test instead of the standard Student's t-test. A simple rule of thumb often employed is to calculate the ratio of the larger sample variance to the smaller sample variance; if this ratio is less than 4:1, we can generally proceed assuming equal population variances.

We use NumPy's `np.var()` function to calculate the variance for each group independently. This preliminary check ensures we select the correct parameters for the SciPy function in the subsequent step:

#find variance for each group
print(np.var(group1), np.var(group2))

7.73 12.26

Upon calculation, the variance for Group 1 is 7.73, and for Group 2 is 12.26. We calculate the ratio: 12.26 / 7.73 = **1.586**. Since 1.586 is significantly less than the threshold of 4, we conclude that we can safely assume that the population variances are equal for the purpose of this analysis. Therefore, we will use the standard independent two-sample t-test.

The `ttest_ind()` Function and Parameterization

The `ttest_ind()` function, housed within the `scipy.stats` module, is the primary mechanism for conducting the independent two-sample t-test in Python. Understanding its required inputs is crucial for accurate implementation. The function syntax is straightforward but requires careful consideration of the `equal_var` parameter based on our preliminary variance check.

The standard syntax for calling the function is:

ttest_ind(a, b, equal_var=True)

The function accepts the following key parameters:

a: This represents the array of sample observations for the first group.
b: This represents the array of sample observations for the second group.
equal_var: This boolean parameter determines the type of test performed. If set to **True**, it runs the standard Student's t-test, assuming equal population **variances**. If set to **False**, it automatically switches to perform Welch's t-test, which is appropriate when variances are unequal. Since our variance check confirmed a ratio less than 4, we will proceed using `equal_var=True`.

Step 2: Executing the Two-Sample T-Test in Python

With the data defined and the necessary assumption regarding variance confirmed, we can now execute the statistical test. We import the `scipy.stats` module (conventionally aliased as `stats`) and invoke the `ttest_ind()` function, passing our two data arrays (`group1` and `group2`) and explicitly setting `equal_var=True`.

The execution of this function immediately returns a tuple containing two critical values: the calculated t-statistic and the two-sided p-value, which are essential for drawing our final statistical conclusion:

import scipy.stats as stats

#perform two sample t-test with equal variances
stats.ttest_ind(a=group1, b=group2, equal_var=True)

(statistic=-0.6337, pvalue=0.53005)

The resulting output provides a **t-statistic** of **-0.6337** and a corresponding p-value of **0.53005**.

Step 3: Interpreting the Statistical Outcomes (Hypothesis Testing)

The final and most crucial step in any statistical test is interpreting the results relative to the established hypotheses. The two competing hypotheses for this two-sample t-test are formalized as follows:

The two hypotheses for this particular two sample t-test are as follows:

Null Hypothesis (H0): μ1 = μ2. This states that the population means of the two plant species are equal.
Alternative Hypothesis (HA): μ1 ? μ2. This states that the population means are *not* equal.

To make a definitive decision, we compare the calculated p-value against a predetermined significance level, typically denoted as alpha (α). Conventionally, α is set at 0.05. The rule is simple: if the p-value is less than or equal to α, we reject the **null hypothesis**; otherwise, we fail to reject it.

In our example, the calculated p-value is **0.53005**. Since 0.53005 is substantially greater than the significance level of 0.05, we **fail to reject the null hypothesis** (H0). Statistically, we do not possess sufficient evidence to conclude that the true mean heights of the two plant species are significantly different from one another.

Conclusion and Next Steps

The implementation of the two-sample t-test in Python, utilizing the robust capabilities of the SciPy library, provides a clear and repeatable method for comparing two independent population means. Our analysis demonstrated the necessary steps, from data preparation and variance checking to function execution and final hypothesis interpretation.

Although the test suggested no significant difference in this specific dataset, mastering this technique is foundational for more complex statistical modeling. For those interested in exploring related statistical procedures, the following resources offer further guidance on t-tests in Python:

How to Conduct a One Sample T-Test in Python
How to Conduct a Paired Samples T-Test in Python