How to Perform Grubbs' Test in Python

Authored by

stats writer

December 23, 2025

The identification and management of abnormal data points are critical steps in any robust data analysis pipeline. These anomalous observations, commonly referred to as outliers, can drastically skew statistical results and lead to erroneous conclusions if not handled correctly. Among the most trusted methods for statistically validating the presence of these anomalies is Grubbs' Test, also known as the maximum normalized residual test.

Grubbs' Test is a powerful statistical procedure designed specifically to detect a single univariate outlier in a sample drawn from a normally distributed population. It calculates a G statistic, which is essentially a measure of how far the suspected outlier is from the sample mean, normalized by the standard deviation. This test is essential for ensuring the integrity of datasets used for modeling and inference, especially when working with highly sensitive data.

While other general statistical packages like SciPy may offer related functions, performing a dedicated Grubbs' Test in the Python ecosystem is often most straightforward using specialized third-party libraries. This detailed guide demonstrates how to leverage the `outlier_utils` package to execute this test efficiently, providing clear examples for both two-sided and one-sided outlier detection.

Grubbs' Test is specifically designed to identify the presence of a single univariate outlier in a dataset. To utilize this test reliably, the dataset should be approximately normally distributed and must contain at least seven observations ($n ge 7$).

This tutorial will now proceed with practical implementation details using Python.

Implementing Grubbs' Test in Python

To implement the test, we rely on the convenient `smirnov_grubbs()` function provided by the outlier_utils package. This function simplifies the often complex statistical computations required by the test, providing a clean interface for data scientists and analysts.

The primary function we will use adheres to the following general syntax:

smirnov_grubbs.test(data, alpha=.05)

This function accepts two primary parameters that govern its operation and sensitivity:

data: This parameter requires a numeric vector or array containing the data values to be tested for outliers. This is typically a NumPy array.
alpha: This defines the significance level (or $alpha$ level) for the test. The default value is 0.05. This value represents the probability of rejecting the null hypothesis (that no outliers exist) when it is actually true (Type I error). A smaller alpha makes the test more stringent.

Before proceeding with the examples, ensure that the necessary package is installed in your Python environment. Use the following command in your terminal or command prompt:

pip install outlier_utils

Once the `outlier_utils` package is successfully installed, you can proceed to import the necessary functions and apply **Grubbs' Test** to your datasets. The following examples demonstrate various practical scenarios for outlier detection.

Example 1: Performing a Two-Sided Test

A two-sided **Grubbs' Test** is the standard application of the procedure, designed to detect a potential outlier at either the minimum (lower tail) or the maximum (upper tail) end of the dataset. This approach is recommended when you have no prior hypothesis about the direction of the potential anomaly.

The code below demonstrates how to initialize a dataset using **NumPy**, which is the preferred structure for numerical operations in Python, and then apply the general `test` function from the imported `smirnov_grubbs` module. We set the alpha level to the conventional 0.05.

```
import numpy as np
from outliers import smirnov_grubbs as grubbs

# Define the dataset containing 17 observations, including the suspected outlier (40)
data = np.array()

# Perform the two-sided Grubbs' test using the default alpha = 0.05
grubbs.test(data, alpha=.05)
```

```
array()
```

Upon execution, this function returns a new array representing the original dataset after the statistically identified outlier has been removed. In this case, the maximum value of 40 was determined to be an outlier at the 0.05 significance level, thus it was successfully removed from the resulting array.

Example 2: Executing One-Sided Tests

While the two-sided test is useful for general detection, situations often arise where we only suspect an outlier exists in one specific direction--either abnormally high or abnormally low. For these directional hypotheses, we utilize one-sided versions of the **Grubbs' Test**, specifically

`min_test` and `max_test`.

The `min_test` function is employed to specifically check if the minimum value in the dataset is a statistically significant outlier. Conversely, the `max_test` function checks only the maximum value. This targeted approach can increase the statistical power for detecting an outlier in the hypothesized tail.

import numpy as np
from outliers import smirnov_grubbs as grubbs

# Define the data array
data = np.array()

# Perform Grubbs' test to see if the minimum value (5) is an outlier
grubbs.min_test(data, alpha=.05)

array()

# Perform Grubbs' test to see if the maximum value (40) is an outlier
grubbs.max_test(data, alpha=.05)

array()

The results confirm the directional nature of the test. When running `min_test`, the minimum value (5) is retained, indicating it is not significantly distant enough from the mean to be classified as an outlier at the 0.05 alpha level. Conversely, running `max_test` correctly identifies and removes the value 40, reinforcing the conclusion that the anomaly resides in the upper tail of the distribution.

Example 3: Extract the Index of the Outlier

While the previous functions return the cleaned dataset, data cleaning often requires knowing exactly where the outlier was located (its index). The `outlier_utils` package provides specialized functions for isolating this information, which is crucial for referencing back to database records or larger data structures.

To pinpoint the exact location of the outlier within the original array, we use the `max_test_indices` function (or `min_test_indices` if looking for the minimum outlier). This returns the zero-based index of the identified anomaly.

import numpy as np
from outliers import smirnov_grubbs as grubbs

```
# Define the data array
data = np.array()

# Perform Grubbs' max test and identify the index (if any) of the outlier
grubbs.max_test_indices(data, alpha=.05)
```

The output tells us that the statistically significant outlier is located at the 17th position (index 16) of the array. Knowing the index allows the analyst to cross-reference the original source data for verification or manual inspection.

Example 4: Extract the Value of the Outlier

If the goal is simply to record the value that was flagged as an anomaly, the `max_test_outliers` function (or `min_test_outliers`) should be used. This function returns a list containing the actual value(s) identified as statistically significant outliers based on the criteria set by the alpha level.

```
import numpy as np
from outliers import smirnov_grubbs as grubbs

# Define the data array
data = np.array()

# Perform Grubbs' max test and identify the actual value (if any) of the outlier
grubbs.max_test_outliers(data, alpha=.05)
```

This tells us that there is one outlier with a value of 40. These targeted extraction functions are invaluable for automated reporting and debugging processes, allowing data scientists to isolate and record anomalies before making any changes to the dataset.

How to Handle Outliers

Identifying an outlier using **Grubbs' Test** is only the first step; the most critical phase involves deciding how to handle the anomaly. The treatment strategy depends heavily on the context of the data and the suspected cause of the outlier. It is essential to approach outlier remediation cautiously, as removing valid data can introduce bias.

If **Grubbs' Test** successfully identifies a statistical outlier in your dataset, you have several primary courses of action, each with its own implications for the subsequent analysis:

Investigate Data Entry and Measurement Errors.

Before any modification or removal, the detected value must be scrutinized. Often, values that appear as outliers are simply due to typos, faulty sensor readings, or data entry errors made by an individual. First, verify that the value was entered correctly by reviewing the raw data source before you make any further decisions.

Impute or Assign a New Value (Data Imputation).

If the outlier turns out to be a result of a typo or data entry error and the original correct value cannot be recovered, you may decide to assign a new value to it using a measure of central tendency. Common imputation strategies include replacing the outlier with the mean or the median of the dataset. Using the median is often preferred, as it is less sensitive to extreme values than the mean, ensuring the imputed value doesn't introduce new distortions.

Remove the Outlier Entirely.

If the value is a true outlier (not an error) and it is determined that this extreme value will disproportionately skew critical statistical parameters, you may choose to remove it. Removal is typically justified when the outlier represents a rare event that is not representative of the population you wish to model, or when the goal is to establish a robust model based on the typical behavior of the data.