Learning Geometric Mean Calculation with Python: A Step-by-Step Guide

Authored by

Mohammed looti

November 3, 2025

The geometric mean is a type of average that indicates the central tendency of a set of numbers by using the product of their values. Unlike the arithmetic mean, the geometric mean is particularly useful for data sets involving growth rates, financial returns, or indices, as it effectively smooths out varying rates of change. When working in Python, there are two primary, reliable ways to calculate the geometric mean efficiently.

These two robust methods leverage powerful numerical libraries: using the dedicated function available in SciPy or crafting a custom function based on logarithms and exponentials using NumPy. Both approaches yield identical results, but developers often choose based on project dependencies and performance needs.

Here is an overview of the two main computational strategies for calculating the geometric mean in Python:

Method 1: Calculate Geometric Mean Using SciPy

from scipy.stats import gmean

#calculate geometric mean
gmean()

Method 2: Calculate Geometric Mean Using NumPy

import numpy as np

#define custom function
def g_mean(x):
a = np.log(x)
return np.exp(a.mean())

#calculate geometric mean
g_mean()

It is important to note that, given the same input data, both calculation methods will return the exact same numerical result, confirming their mathematical equivalence. The following expanded sections provide detailed instructions and practical examples for implementing these methods.

Understanding the Geometric Mean

The geometric mean is fundamentally defined as the N-th root of the product of N numbers. Mathematically, it is expressed as: $G = sqrt{x_1 x_2 cdots x_n}$. This distinct definition makes it

superior to the arithmetic mean in contexts where values are multiplied together, such as calculating compound interest or determining the average performance ratio across multiple periods.

In data science and statistics, the geometric mean is typically used when dealing with normalized data, index numbers, or data sets that exhibit exponential growth. For instance, averaging rates of return for investments requires the geometric mean to accurately reflect the true compound growth. Using the standard arithmetic mean in these scenarios would often lead to an inflated or misleading average.

A convenient property of the geometric mean, which informs the NumPy implementation, is that it can be calculated using logarithms. Specifically, the geometric mean is equal to the exponential of the arithmetic mean of the logarithms of the values. This transformation simplifies the calculation, especially for very large datasets or numbers, by converting multiplication operations into addition operations.

Method 1: Utilizing the SciPy Library

The simplest and most recommended approach for calculating the geometric mean in Python is by leveraging the specialized statistical functions provided by the SciPy library. SciPy is built upon NumPy and offers high-level scientific computing capabilities, including the dedicated `gmean()` function located within the `scipy.stats` module.

Using the built-in function abstracts away the complex mathematical implementation (whether it uses the direct product method or the logarithmic transformation), allowing the user to focus solely on the input data. This method is preferred for its readability, reliability, and optimized performance, especially when dealing with large arrays of numerical data.

To use this method, you only need to ensure that SciPy is installed in your environment and then import the specific function. The function handles various iterable inputs, such as standard Python lists or NumPy arrays, returning a single floating-point number representing the result.

Method 2: Implementing a Custom NumPy Function

For scenarios where the SciPy dependency is undesirable, or if a deeper understanding of the calculation method is required, the geometric mean can be calculated using core NumPy functions. This custom approach relies on the logarithmic property mentioned earlier: the geometric mean is the exponent of the average of the natural logarithms of the data points.

This method requires defining a small function that first takes the natural logarithm of the input array (using `np.log()`), then calculates the arithmetic mean of those transformed values, and

finally applies the exponential function (`np.exp()`) to the result. This mathematical sequence ensures numerical stability, especially when multiplying many small or large numbers, which can sometimes lead to underflow or overflow errors if calculated via direct multiplication.

While slightly more verbose than the SciPy method, this approach demonstrates the fundamental mathematical operation underpinning the geometric mean calculation and relies only on the widely available NumPy library. It is a powerful illustration of how complex statistics can be derived from basic numerical primitives.

Practical Example: Calculating the Geometric Mean

Let us apply both methods to a sample array of values: $X = $. We will verify that both the built-in SciPy function and the custom NumPy function produce identical results.

First, we use the `gmean()` function from SciPy:

from scipy.stats import gmean

#calculate geometric mean
gmean()

4.81788719702029

The computation using the SciPy library yields a geometric mean of approximately **4.8179**. This result is achieved quickly and reliably, requiring minimal code setup.

Next, we confirm this result using our custom function built entirely with NumPy primitives:

import numpy as np

#define custom function
def g_mean(x):
a = np.log(x)
return np.exp(a.mean())

#calculate geometric mean
g_mean()

4.81788719702029

As expected, the geometric mean calculated using the logarithmic transformation approach also results in **4.8179**, which perfectly matches the result obtained from the dedicated SciPy function.

This numerical equivalence reinforces the mathematical validity of both techniques.

Crucial Consideration: Handling Zero Values

A critical point to remember when calculating the geometric mean is the effect of zero values within the dataset. Since the geometric mean involves the product of all elements, if even a single element in the array is zero, the product of the entire set will be zero, causing the geometric mean itself to be zero.

Furthermore, if you are using the logarithmic method (the custom NumPy function), attempting to take the natural logarithm of zero is mathematically undefined and will typically result in a runtime warning (e.g., "divide by zero encountered in log") and return a result of negative infinity or Not a Number (NaN), followed by an exponential function returning zero.

Depending on the context of your data--for instance, if the zeros represent missing data, non-existent growth, or valid measurements--you may need to preprocess the data. A common procedure is to filter out the zeros before calculation if the geometric mean is intended to measure proportional change among only the positive elements. The following Python code demonstrates how to filter zeros from a list before proceeding with the calculation:

#create array with some zeros
x =

#remove zeros from array
x_new =

#view updated array
print(x_new)

After filtering, the new list `x_new` contains only the positive values, allowing for a meaningful geometric mean calculation that reflects the central tendency of the non-zero components.

Summary of Approaches and Best Practices

When selecting a method for calculating the geometric mean in Python, developers generally have a straightforward choice based on project needs. For standard statistical analysis and environments where the SciPy library is already in use, the `scipy.stats.gmean()` function is the clearest and most concise option.

If the project must minimize dependencies or if the calculation is being performed within a restricted environment where only NumPy is available, defining the custom logarithmic function is

the appropriate path. This method ensures high performance and numerical stability using established mathematical properties.

Regardless of the chosen method, always ensure your data is appropriately cleaned and preprocessed. Verifying that the input array contains strictly positive values--or deciding how to treat zeros (e.g., filtering them out or replacing them with a tiny epsilon value)--is the most critical step for obtaining an accurate and meaningful geometric mean.

Additional Resources

For further study on statistical methods and numerical computing in Python, consult the following authoritative documentation:

SciPy Documentation on gmean
NumPy Documentation on Logarithmic Functions
Understanding the Exponential Function