How to Easily Calculate Row and Column Sums in NumPy Arrays

Authored by

stats writer

November 21, 2025

As a foundational library for scientific computing in Python, NumPy provides powerful tools for efficient data manipulation, particularly when dealing with multi-dimensional datasets. One of the most common analytical tasks involves calculating aggregate statistics across specific dimensions of a data structure. This post focuses on the precise methodology for summing the rows and columns of a NumPy array, leveraging the highly optimized `np.sum()` function.

The core functionality for aggregation in NumPy resides within the `np.sum()` function. This universal function (ufunc) is designed to compute the sum of array elements over a specified axis. Understanding how to properly utilize the `axis` parameter is critical to achieving the desired summation across either the rows (axis=1) or the columns (axis=0) of a 2D NumPy array.

When invoked without specifying an axis (i.e., `axis=None`), `np.sum()` defaults to returning the sum of **all** elements within the array, flattening the structure into a single scalar value. However, for obtaining dimension-specific totals--such as row sums or column sums--explicitly defining the `axis` parameter guides the function on which dimension to collapse during the summation process. Mastery of this parameter is essential for performing sophisticated data analysis and transformation using NumPy.

Understanding the Role of the Axis Parameter

Before diving into specific code examples, it is imperative to clarify how the `axis` parameter dictates the behavior of summation functions in NumPy. In a two-dimensional array (often visualized as a matrix), we typically refer to two primary axes: the vertical axis and the horizontal axis. NumPy assigns numerical indices to these axes starting from zero.

In the context of a standard 2D array, the dimensions are structured as `(rows, columns)`. Therefore, **axis 0** corresponds to the rows dimension (the vertical direction), and **axis 1** corresponds to the columns dimension (the horizontal direction). When you specify an axis for summation, you are instructing NumPy to collapse that particular axis by summing all elements along it.

To sum the columns, you must specify the axis that represents the rows (axis=0) because the calculation proceeds down the columns, collapsing the row structure. Conversely, to sum the rows, you must specify the axis that represents the columns (axis=1) because the calculation proceeds across the rows, collapsing the column structure. Misinterpreting this concept is a common pitfall for new NumPy users, so a clear mental model is vital for effective array manipulation.

Core Syntax for Dimension-Specific Summation

The following two methods represent the standard and most efficient approaches for calculating

sums along the major dimensions of a 2D array. Whether you choose to call `np.sum(arr, axis=X)` or use the array method `arr.sum(axis=X)`, the underlying calculation mechanism is the same, relying on highly optimized routines.

These fundamental commands allow instant aggregation, forming the backbone of many vectorized operations in data science workflows. The selection of axis 0 or axis 1 directly controls the output structure and the orientation of the aggregation.

Method 1: Sum Rows of NumPy Array (Collapsing Axis 1)

arr.sum(axis=1)

Method 2: Sum Columns of NumPy Array (Collapsing Axis 0)

arr.sum(axis=0)

Initializing the 2D Demonstration Array

To illustrate these concepts practically, we will utilize a structured 2D NumPy array created using `np.arange()` and reshaped into a 6x3 matrix. This array provides a clear, sequential set of values, making it easier to manually verify the resulting sums for both row and column calculations.

The dimensions of this array are six rows and three columns, meaning its shape is `(6, 3)`. The elements range from 0 to 17. Pay close attention to the indices, as these will guide our manual verification steps later in the tutorial, ensuring we fully grasp how the axis parameter works.

import numpy as np

# Create NumPy array of shape (6, 3)
arr = np.arange(18).reshape(6,3)

# View the structure of the NumPy array
print(arr)

]

Example 1: Calculating Sums Across Rows (axis=1)

To calculate the sum of elements for each individual row, we must set the `axis` parameter equal to 1. This instructs the `np.sum()` function to iterate along the columns (the horizontal dimension) and aggregate the values, effectively collapsing the column axis. The resulting output is a 1D array

where each element corresponds to the total sum of one row from the original 2D structure.

When the summation occurs along axis 1, the number of resulting elements will match the number of rows in the input array. Since our demonstration array `arr` has six rows, the output array will contain six summed values. This operation is critical for normalizing data based on row totals or calculating per-sample statistics, as the shape reduction is mathematically precise and computationally fast.

import numpy as np

# Calculate sum of rows in NumPy array using axis=1
arr.sum(axis=1)

array()

Verification of Row Summation Results

We can manually verify the resulting summation array against our original 6x3 matrix to confirm the functionality of `axis=1`. This step provides confidence in the correct interpretation of the `np.sum()` operation and illustrates why the output array has a length equal to the number of rows.

The original array rows are defined as:

Row 0:
Row 1:
Row 2:
Row 3:
Row 4:
Row 5:

The calculation proceeds across the columns for each row index:

The sum of values in the first row (index 0) is 0 + 1 + 2 = **3**.
The sum of values in the second row (index 1) is 3 + 4 + 5 = **12**.
The sum of values in the third row (index 2) is 6 + 7 + 8 = **21**.
The sum of values in the fourth row (index 3) is 9 + 10 + 11 = **30**.
The sum of values in the fifth row (index 4) is 12 + 13 + 14 = **39**.
The sum of values in the sixth row (index 5) is 15 + 16 + 17 = **48**.

This verification confirms that the resulting array accurately reflects the total sum for each row when `axis=1` is specified.

Example 2: Calculating Sums Down Columns (axis=0)

To calculate the sum of elements for each individual column, we must specify `axis=0`. Recall that axis 0 represents the rows dimension. By collapsing this dimension, we instruct NumPy to iterate vertically down the columns, accumulating the values from the top row to the bottom row for each column index, effectively giving us the feature totals.

This operation is essential when calculating statistics across features or variables, where each column typically represents a specific attribute in a dataset. Because our demonstration array has three columns, the resulting output will be a 1D array containing three aggregated values, corresponding to the total sum of Column 0, Column 1, and Column 2. This represents a reduction in the number of rows, leaving only the column dimension.

import numpy as np

# Calculate sum of columns in NumPy array using axis=0
arr.sum(axis=0)

array()

Verification of Column Summation Results

We must confirm the results of the column summation (`axis=0`) by manually adding the elements that constitute each vertical column in the original array structure. This is crucial for verifying that the axis parameter was correctly applied to aggregate elements vertically.

The elements forming the columns are structured as:

Column 0 elements (index 0): 0, 3, 6, 9, 12, 15
Column 1 elements (index 1): 1, 4, 7, 10, 13, 16
Column 2 elements (index 2): 2, 5, 8, 11, 14, 17

The resulting sums are calculated by aggregating down the rows:

The sum of values in the first column (index 0) is 0 + 3 + 6 + 9 + 12 + 15 = **45**.
The sum of values in the second column (index 1) is 1 + 4 + 7 + 10 + 13 + 16 = **51**.
The sum of values in the third column (index 2) is 2 + 5 + 8 + 11 + 14 + 17 = **57**.

This comprehensive verification confirms that the output array correctly represents the aggregation of values along the vertical dimension of the input NumPy array.

Calculating the Grand Total Sum of All Elements

While row and column aggregations are frequently necessary, there are times when the objective is to find the single, scalar sum of every single element contained within the array. This is achieved by omitting the `axis` parameter or explicitly setting it to `None`.

When `axis=None`, NumPy effectively flattens the multi-dimensional array into a single dimension before performing the summation. This process is generally faster and more memory-efficient than manually iterating through nested loops in standard Python, demonstrating the power of NumPy's underlying C implementation.

Using our array `arr`, which contains sequential integers from 0 to 17, we can calculate the expected total sum. The sum of the first *n* elements (including 0) is calculated efficiently by the `np.sum()` function.

import numpy as np

# Calculate the sum of all elements
total_sum = arr.sum(axis=None)
print(total_sum)

153

As a final check, we can verify this result by summing the previously calculated row sums (3 + 12 + 21 + 30 + 39 + 48) or the column sums (45 + 51 + 57), both of which must yield the final result of 153, confirming the mathematical consistency of NumPy's aggregation functions across different dimensions.

Summary and Advanced Considerations

The ability to efficiently sum across specific axes is fundamental to effective scientific computing with NumPy. By correctly utilizing the `axis` parameter--where `axis=1` collapses columns to give row sums, and `axis=0` collapses rows to give column sums--users can precisely control the dimensionality and focus of their data aggregations.

For those dealing with higher-dimensional arrays (e.g., 3D or 4D tensors), the principle remains the same: the specified axis is the dimension that is collapsed and summed over. For instance, in a 3D array of shape (D1, D2, D3), specifying `axis=0` sums across the first dimension, leaving an output array of shape (D2, D3). This consistent logic allows for scaling these aggregation techniques to vastly complex datasets.

It is highly recommended that users consult the complete official documentation for the `np.sum()` function to explore additional parameters such as `dtype` (data type) and `out` (output array specification), which offer further control over performance and memory management in large-scale applications.

The following tutorials explain how to perform other common operations in NumPy: