How to Find the Sum of Rows in a Pandas DataFrame

Authored by

stats writer

December 24, 2025

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

stats writer (2025). *How to Find the Sum of Rows in a Pandas DataFrame*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=108642

Calculating summaries of datasets is a fundamental task in data analysis, and when working with Pandas DataFrames, determining the sum of values across rows is a common requirement. The powerful and flexible DataFrame.sum() method provides an efficient way to achieve this aggregation. Understanding how to correctly apply this method, particularly by specifying the computation axis, is crucial for accurate results.

The primary function of the sum() method is to return the sum of all values within the specified dimension of the DataFrame. By default, it sums across rows (producing column totals, using `axis=0`). However, to calculate the sum of values horizontally (row-wise), we must explicitly set the `axis` parameter to `1`. The output of the row summation operation is typically a Pandas Series, where the index corresponds to the original DataFrame's row indices.

Understanding the Axis Parameter for Row Aggregation

Mastering the use of the axis argument is essential for leveraging Pandas effectively for statistical aggregations. In the context of the DataFrame.sum() method, `axis=0` refers to the index, causing aggregation down the rows (producing column totals). Conversely, setting `axis=1` instructs the function to operate across the columns, calculating a single aggregated value (the sum) for each distinct row.

This distinction is critically important when summarizing data. If the axis is omitted, `axis=0` is generally assumed, resulting in column totals. Since our goal is to find the sum of rows, specifying `axis=1` is mandatory. This allows data professionals to quickly generate meaningful row-level summaries for various analytical purposes, such as creating new features or validating data completeness.

Setting Up the Sample DataFrame

To demonstrate the different row summation techniques, we first establish a sample DataFrame. This dataset, created using Python's Pandas and NumPy libraries, simulates performance metrics across several categories: 'rating', 'points', 'assists', and 'rebounds'. The inclusion of a `NaN` value in the 'rebounds' column allows us to illustrate how Pandas handles missing data during aggregation.

The following code initializes the DataFrame structure. We will be applying all subsequent summation examples to this dataset:

```
import pandas as pd
import numpy as np

#create DataFrame
```

ARABPSYCHOLOGY.COM

df = pd.DataFrame({'rating': ,
'points': ,
'assists': ,
'rebounds': })

#view DataFrame
df

rating points assists rebounds
0 90 25 5 8.0
1 85 20 7 NaN
2 82 14 7 10.0
3 88 16 8 6.0
4 94 27 5 6.0
5 90 20 7 9.0
6 76 12 6 6.0
7 75 15 9 10.0
8 87 14 9 10.0
9 86 19 5 7.07

Example 1: Finding the Aggregate Sum of Every Row

The simplest application of row summation is calculating the total value across all numerical columns for each observation. We achieve this by invoking the sum() method and setting the `axis` parameter to `1`. This operation returns a Pandas Series, where each element represents the aggregate sum of its corresponding row.

Note how the calculation handles the missing value in Index 1. By default, Pandas ignores `NaN` values, treating the missing entry as zero for the purpose of summation. This is crucial behavior that often saves time on data cleaning but must be understood clearly to avoid misinterpretation.

df.sum(axis=1)

0 128.0
1 112.0
2 113.0
3 118.0
4 132.0
5 126.0
6 100.0

ARABPSYCHOLOGY.COM

7 109.0
8 120.0
9 117.0
dtype: float64

From this output, we can deduce the total contribution of all numerical fields for each data point:

The sum of values in the first row (Index 0) is **128.0**.
The sum of values in the second row (Index 1) is **112.0** (85 + 20 + 7, as the NaN value is skipped).
The sum of values in the third row (Index 2) is **113.0**.

This technique provides a fast, initial summary statistic for validating the distribution and magnitude of row totals.

Example 2: Storing Row Sums in a New Column

For most analytical workflows, simply viewing the sums is not enough; the calculated row totals need to be integrated back into the original DataFrame as a new feature. This new column can then be used for subsequent analysis, ranking, or filtering operations.

Since the result of `df.sum(axis=1)` is a Pandas Series that preserves the index alignment of the DataFrame, we can directly assign this Series to a new column name. We will name this column `'row_sum'`.

The code below demonstrates this assignment and displays the updated DataFrame, now enriched with the aggregated data:

#define new DataFrame column 'row_sum' as the sum of each row
df = df.sum(axis=1)

#view DataFrame
df

rating points assists rebounds row_sum
0 90 25 5 8.0 128.0
1 85 20 7 NaN 112.0
2 82 14 7 10.0 113.0
3 88 16 8 6.0 118.0
4 94 27 5 6.0 132.0
5 90 20 7 9.0 126.0
6 76 12 6 6.0 100.0

ARABPSYCHOLOGY.COM

7 75 15 9 10.0 109.0
8 87 14 9 10.0 120.0
9 86 19 5 7.0 117.0

The new `'row_sum'` column now provides a comprehensive horizontal total for each record, serving as a powerful derived metric for comparative analysis.

Example 3: Summing a Short List of Specific Columns Using Arithmetic Operators

Sometimes, only a select few columns need to be aggregated per row. If the number of columns is small (e.g., two or three), the most direct way to calculate their sum is through element-wise addition using the standard arithmetic operator (+). This method is highly readable and performs the necessary row-wise summation implicitly because Pandas aligns the Series objects by their shared index before performing the calculation.

To find the combined score of 'points' and 'assists', we can use the following concise syntax:

#define new DataFrame column as sum of points and assists columns
df = df + df

#view DataFrame
df

rating points assists rebounds sum_pa
0 90 25 5 8.0 30
1 85 20 7 NaN 27
2 82 14 7 10.0 21
3 88 16 8 6.0 24
4 94 27 5 6.0 32
5 90 20 7 9.0 27
6 76 12 6 6.0 18
7 75 15 9 10.0 24
8 87 14 9 10.0 23
9 86 19 5 7.0 24

The resulting `'sum_pa'` column holds the combined value of only the specified columns, offering a targeted metric for analysis. While elegant for short lists, this method is not advisable when dealing with larger, variable column sets.

ARABPSYCHOLOGY.COM

Example 4: Summing a Long List of Specific Columns Using Subsetting

For scenarios requiring the summation of many columns, or when the column selection needs to be dynamic (e.g., summing all columns except one), relying on the addition operator becomes unwieldy. The robust solution is to first subset the DataFrame to include only the desired columns and then apply the standard `.sum(axis=1)` method to this temporary view.

In this example, we demonstrate how to calculate the sum of all columns DataFrame except for `'rating'`. We manage the column selection using a list, which provides programmatic control over the aggregation:

#define col_list as a list of all DataFrame column names
col_list= list(df)

#remove the column 'rating' from the list
col_list.remove('rating')

#define new DataFrame column as sum of rows in col_list
df = df.sum(axis=1)

#view DataFrame
df

rating points assists rebounds new_sum
0 90 25 5 8.0 38.0
1 85 20 7 NaN 27.0
2 82 14 7 10.0 31.0
3 88 16 8 6.0 30.0
4 94 27 5 6.0 38.0
5 90 20 7 9.0 36.0
6 76 12 6 6.0 24.0
7 75 15 9 10.0 34.0
8 87 14 9 10.0 33.0
9 86 19 5 7.0 31.0

This technique is highly scalable and ensures that the sum() method is only applied to the relevant numerical fields, providing superior control for complex feature engineering.

Handling Missing Data (NaN) During Summation

A crucial aspect of calculating row sums involves understanding how the sum() method interacts

ARABPSYCHOLOGY.COM

with missing values, represented by NaN. By default, the `skipna` parameter is set to `True`, meaning Pandas automatically excludes any NaN entries from the summation calculation, allowing the sum of the remaining non-null values to be calculated.

If, however, the business or analytical requirement dictates that a row should result in a missing value if even one of its components is missing, the `skipna` parameter must be explicitly set to `False`. This forces the propagation of the missingness to the aggregated total:

df.sum(axis=1, skipna=False)

0 128.0
1 NaN
2 113.0
...
dtype: float64

In the example above, if `skipna=False` were used, the result for Index 1 would become `NaN`, reflecting the incomplete data. Analysts must consciously choose the appropriate `skipna` setting based on whether they want to impute zero for missing values during summation or strictly require complete data for a total.

Summary of Row Aggregation Techniques

Finding row sums in a Pandas DataFrame is a foundational data manipulation skill achieved primarily through the use of the `.sum(axis=1)` method. This article has detailed multiple strategies, from calculating the total aggregate for all columns to employing targeted summation on specific subsets using both arithmetic operations and DataFrame subsetting.

Consistent use of `axis=1` ensures row-wise calculation, transforming horizontal data into vertical summaries. By combining this knowledge with the ability to manage column selection and control the handling of NaN values, data practitioners can efficiently derive meaningful features for advanced statistical modeling and reporting.

You can find the complete documentation for the pandas sum() function here.

ARABPSYCHOLOGY.COM