How to Easily Calculate the Average of Specific Columns in Pandas

Authored by

stats writer

December 2, 2025

In the world of data analysis using Pandas, calculating descriptive statistics is a fundamental operation. While calculating column averages (the mean of an entire feature) is straightforward, analysts often need to compute the average value across specific columns for each individual row. This row-wise calculation is essential for creating composite scores, normalizing data, or performing feature engineering where the relationship between variables needs to be summarized horizontally across observations. Mastering the application of the built-in mean() function alongside the crucial axis parameter allows for precise control over these aggregation techniques, ensuring that the results accurately reflect the desired statistical summary for your DataFrame.

Understanding Axis Control in Pandas

To successfully calculate row averages, it is critical to understand how Pandas handles dimensional operations. Most aggregation functions, including the mean() function, operate along an axis. By default, when you call an aggregation function on a DataFrame without specifying the axis, it assumes `axis=0`. The value `axis=0` refers to the index (rows) and dictates that the operation should be performed down the rows, resulting in a single output value per column. This calculates the mean of each column.

Conversely, to calculate row averages, we must specify the `axis` as 1 (`axis=1`). This parameter tells Pandas to apply the aggregation function horizontally across the columns. When we use `axis=1` with the `mean() function`, the result is a Series where each element represents the average value derived from the corresponding row's features. This simple parameter shift unlocks the ability to generate powerful, row-specific metrics crucial for complex analytical tasks. Understanding this distinction is the foundation of efficient data manipulation within Python using the Pandas library.

Core Methods for Row-Wise Averaging

The calculation of average row values can be achieved using two principal methods, depending entirely on whether the analysis requires all numerical columns to be included or just a select subset. Both methods rely on the proper application of the `axis=1` parameter to ensure horizontal computation. Below are the foundational code snippets outlining these two distinct approaches:

The core syntax for calculating average row values for selected columns in a pandas DataFrame is based on two fundamental approaches:

Method 1: Calculate Average Row Value for All Applicable Columns

df.mean(axis=1)

This command operates directly on the entire DataFrame, calculating the average for every row across all available columns that contain numerical data types. Columns with non-numeric data are automatically ignored in this calculation, simplifying the process when all features are relevant.

Method 2: Calculate Average Row Value for Specific Columns

df].mean(axis=1)

This method requires intermediate column selection using double square brackets (`]`). This subsetting creates a new temporary DataFrame containing only the desired columns, upon which the mean() function is applied with `axis=1`. This is the preferred method for generating weighted scores or composite metrics.

Setting Up the Demonstration DataFrame

To practically illustrate these aggregation techniques, we will utilize a sample DataFrame containing sports statistics. This dataset comprises eight rows, representing eight different player performances, with columns detailing performance metrics such as 'points', 'assists', and 'rebounds'. Using a concise Python script, we first import the Pandas library and then initialize the DataFrame structure, providing a clear foundation for the subsequent averaging calculations.

The code below defines and displays the initial structure of our dataset. Note that all columns contain numerical data, which simplifies the averaging process as no columns need to be explicitly excluded due to incompatible data types. Understanding this initial structure is key to interpreting the row-wise calculations that follow. This DataFrame serves as the benchmark for both Method 1 and Method 2 demonstrations.

The following example demonstrates how to use each method in practice with this standardized Pandas DataFrame:

import pandas as pd

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

#view DataFrame
df

points assists rebounds

0 14 5 11
1 19 7 8
2 9 7 10
3 21 9 6
4 25 12 6
5 29 9 5
6 20 9 9
7 11 4 12

Method 1: Calculating the Average Row Value Across All Columns

The first and simplest technique involves computing the mean across all numerical columns for every row. This is particularly useful when all features contribute equally to a desired summary statistic, such as an overall performance index. By setting axis=1 and assigning the result to a new column named `average_all`, we enrich the existing DataFrame without altering the original feature columns. This process is highly efficient and requires minimal syntax, making it ideal for a preliminary assessment of central tendency per observation.

The following code snippet executes this calculation, creating the new column and displaying the updated DataFrame. Notice how the computation considers all three columns ('points', 'assists', and 'rebounds') for each row calculation, resulting in a single mean value per observation. This provides a holistic measure of performance for each player entry in the dataset. Since three columns are being averaged, the count of elements used in the denominator is always three.

#define new column that shows the average row value for all columns
df = df.mean(axis=1)

#view updated DataFrame
df

points assists rebounds average_all
0 14 5 11 10.000000
1 19 7 8 11.333333
2 9 7 10 8.666667
3 21 9 6 12.000000
4 25 12 6 14.333333
5 29 9 5 14.333333
6 20 9 9 12.666667
7 11 4 12 9.000000

Interpreting the output confirms that the calculation correctly divides the sum of the three values in each row by three. For instance, the calculation for the first and second rows clearly demonstrates this method:

The average value of the first row (Index 0) is calculated as: (14 + 5 + 11) / 3 = **10.00**.

The average value of the second row (Index 1) is calculated as: (19 + 7 + 8) / 3 ≈ **11.33**.

This result, stored in the new `average_all` column, provides a single metric summarizing the player's overall input across all measured variables.

Method 2: Calculating the Average Row Value for Specific Columns

Often, data analysis requires calculating an average only among a select group of features, effectively ignoring others. To achieve this selective row-wise aggregation, we must first subset the Pandas DataFrame to include only the necessary columns. This is accomplished by passing a list of column names--hence the requirement for double square brackets--before applying the mean() function with `axis=1`. In our example, we choose to average 'points' and 'rebounds', deliberately excluding 'assists' from the calculation.

The syntax below isolates the 'points' and 'rebounds' columns. Since we are only selecting two columns, the divisor in our mean calculation will automatically become two, regardless of how many other columns exist in the original DataFrame. This targeted approach is essential for deriving metrics where not all variables should contribute to the final composite score, allowing for focused analysis on specific feature relationships.

#define new column that shows average of row values for points and rebounds columns
df = df].mean(axis=1)

#view updated DataFrame
df

points assists rebounds avg_points_rebounds
0 14 5 11 12.5
1 19 7 8 13.5
2 9 7 10 9.5
3 21 9 6 13.5
4 25 12 6 15.5
5 29 9 5 17.0
6 20 9 9 14.5
7 11 4 12 11.5

The resulting column, `avg_points_rebounds`, now contains the mean calculated exclusively from the specified pair of columns. The interpretation of the first two rows clarifies this two-variable calculation, noting the change in the denominator from 3 to 2:

The average value of "points" and "rebounds" in the first row (Index 0) is calculated as: (14 + 11) / 2 = **12.5**.

The average value of "points" and "rebounds" in the second row (Index 1) is calculated as: (19 + 8) / 2 = **13.5**.

This demonstrates the power of column selection combined with the axis parameter for granular data aggregation.

Handling Missing Data (NaN) During Row Aggregation

A crucial consideration when performing row-wise aggregation is the handling of missing data, typically represented as NaN (Not a Number) in Pandas. By default, the `mean()` function in Pandas automatically excludes NaN values from the numerator (the sum) and the denominator (the count of values). This behavior ensures that the mean is calculated only based on the valid, non-missing values present in that specific row subset, preventing skewed results due to data gaps.

If, for example, a row had three columns selected for averaging, but one column contained NaN, the mean would be calculated based on the sum of the two valid values divided by two, not three. This is generally the desired outcome for calculating unbiased statistical averages. However, if you need the presence of a missing value to force the resulting average to also be NaN, indicating that the row is incomplete, you would need to adjust the function using the `skipna=False` argument within the mean() function call.

Advanced Applications and Feature Engineering

Calculating row averages is a foundational technique in feature engineering, where analysts derive new, meaningful metrics from existing data. By aggregating multiple features into a single column, we effectively reduce dimensionality and create composite scores that might be more predictive or easier to interpret than the raw features alone. For instance, in financial datasets, averaging metrics like "Price-to-Earnings Ratio" and "Price-to-Sales Ratio" across a row can yield a new "Valuation Score" for a company, simplifying complex financial health assessments.

Furthermore, this technique is not limited to simple averages. Other aggregation functions, such as `sum()`, `min()`, `max()`, and `median()`, can be applied identically along `axis=1`. The ability to select specific columns and apply precise functions row-wise provides immense flexibility for

cleaning, transforming, and preparing complex datasets for machine learning models or advanced statistical reporting, allowing data scientists to tailor their feature sets precisely to model requirements.

Conclusion: Mastering Row-Wise Data Summarization

Mastering row-wise data aggregation in Pandas is a cornerstone skill for any data professional working with tabular data. By correctly utilizing the axis parameter set to `1`, you gain complete control over whether the averaging function operates on all columns or a carefully curated subset. Whether generating a holistic performance index (Method 1) or creating specialized composite metrics (Method 2), the principles remain consistent: select your data, specify the aggregation function, and define the axis of operation.

This expertise ensures that your data analysis scripts are robust, readable, and capable of generating the precise statistical summaries required for informed decision-making, significantly enhancing your efficiency in data manipulation and feature engineering tasks.

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