How to Create a Pandas Pivot Table with Multiple Aggregation Functions

Authored by

stats writer

November 22, 2025

Mastering Multi-Function Aggregation in Pandas Pivot Tables

The Pandas library provides highly efficient tools for data analysis, and the `pivot_table()` function is fundamental for summarizing and reshaping data. A common requirement in data visualization and reporting is the ability to apply several different statistical calculations simultaneously to the same data columns. Fortunately, Pandas allows users to pass multiple values to the **aggfunc** argument when defining a pivot table, greatly simplifying complex data summarization tasks. This technique avoids the need to run multiple pivot operations separately and then merge the results, leading to cleaner, more readable, and more efficient code.

When utilizing the `pivot_table()` method, the **aggfunc** parameter accepts a list, tuple, or dictionary of aggregation function names (as strings) or actual function objects. This capability is essential for generating comprehensive summaries that require measures of centrality (like mean or median) alongside measures of spread (like standard deviation) and counts. Understanding how to structure this input correctly is key to unlocking advanced data analysis capabilities within the Pandas framework.

The Core Syntax for Defining Multiple Aggregation Functions

To create a pivot table using multiple aggregation functions, you simply enclose the desired function names in a tuple and assign this tuple to the **aggfunc** parameter. The `pivot_table()` method will then execute each specified function against the designated **values** column, grouped by the criteria defined in the **index** argument. This results in a multi-indexed column structure where each aggregation function forms a header level.

The general syntax demonstrates this structure clearly. Here, we specify a primary grouping column (`col1`) and the data we wish to aggregate (`col2`). The **aggfunc** argument receives both `sum` and `mean`, instructing Pandas to calculate both metrics for the values in `col2` for every unique entry in `col1`.

df.pivot_table(index='col1', values='col2', aggfunc=('sum', 'mean'))

This powerful yet concise command produces a resulting table that simultaneously displays both the total (sum) and the average (mean) of the values found in **col2**, meticulously grouped according to the categories present in **col1**. This immediate side-by-side comparison is invaluable for exploratory data analysis, providing different perspectives on the centralized tendencies and overall magnitude of the data set. The structure automatically handles the necessary data grouping and calculation optimization inherent to the Pandas workflow.

Setting Up the Demonstration Data: A Basketball Example

To illustrate this functionality in a practical context, we will utilize a sample DataFrame containing fictional statistics for several basketball players across three different teams (A, B, and C). This dataset includes columns for the `team` affiliation, `points` scored, and `assists` made. The goal is to calculate summary statistics for the `points` column based on the grouping defined by the `team` column.

First, we must import the necessary Pandas library and then initialize our sample DataFrame. Notice how the data is structured: each row represents a player's performance in a particular game or observation, and we have multiple observations per team, making it ideal for grouping and aggregation operations using the `pivot_table()` method.

import pandas as pd

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

#view DataFrame
print(df)

team points assists
0 A 4 2
1 A 4 2
2 A 2 5
3 A 8 5
4 B 9 4
5 B 5 7
6 B 5 5
7 B 7 3
8 C 8 9
9 C 8 8
10 C 4 4
11 C 3 4

Upon reviewing the output, we confirm that our dataset contains 12 records, equally distributed among the three teams. This foundational DataFrame is now prepared for complex aggregation operations, allowing us to derive meaningful insights into the scoring performance of each team by

leveraging the multiple functions available via the **aggfunc** parameter.

Practical Application: Summation and Mean Aggregation

Our first concrete task is to summarize the total scoring power (sum of points) and the average efficiency (mean of points) for each team. By passing `('sum', 'mean')` as a tuple to the **aggfunc** parameter, we instruct Pandas to perform both calculations simultaneously on the `points` column, indexed by `team`. This single operation produces a clean, two-dimensional summary.

#create pivot table to summarize sum and mean of points by team
df.pivot_table(index='team', values='points', aggfunc=('sum', 'mean'))

mean sum
team
A 4.50 18
B 6.50 26
C 5.75 23

The generated pivot table is highly informative. It clearly summarizes two distinct, yet related, performance metrics for the `points` scored by each team. This output structure, where the aggregation functions become the column headers, is a hallmark of using multiple functions in **aggfunc**. The ability to see both the cumulative score (sum) and the average score per observation (mean) side-by-side provides a nuanced understanding of team dynamics, allowing analysts to quickly compare total output versus average unit performance.

Interpreting the Results of Multi-Aggregation

The resulting pivot table allows for rapid comparative analysis between the teams based on their scoring metrics. Interpreting these aggregated values helps us understand not only which team scored the most overall but also which team performed most consistently or efficiently on average. The immediate visualization of these statistics makes complex comparisons straightforward, confirming that Team B leads both in total points and average points per observation.

A detailed breakdown of the output reveals the following specific insights drawn directly from the aggregated data:

Players on team **A** registered a mean points value of **4.50** (indicating their average performance per observation) and accumulated a total sum points value of **18**.
Players on team **B** demonstrated higher average performance, achieving a mean points value of **6.50**, and also led in overall scoring with a total sum points value of **26**.
Players on team **C** showed solid, balanced performance with a mean points value of **5.75** and a

total sum points value of **23**.

This example highlights how combining aggregation functions is beneficial. While Team B has the highest sum and mean, if we had only looked at the sum, we might miss subtle differences in per-game performance relative to other teams. The combined view ensures a robust initial assessment of the dataset, providing a complete picture of both total magnitude and typical scoring rate.

Exploring Other Useful Aggregation Functions

While mean and sum are essential, the utility of **aggfunc** extends far beyond basic arithmetic. Analysts often require metrics that describe the distribution, spread, or extreme values within the dataset. The `pivot_table()` function supports a wide array of built-in statistical functions that can be included in the aggregation tuple.

Key metrics commonly requested for comprehensive data profiling include:

count: Determines the number of non-null observations for the group. Essential for verifying sample size consistency.
min: Identifies the lowest recorded value in the group, indicating the floor of performance.
max: Identifies the highest recorded value in the group, showing the ceiling of performance.
median: Calculates the 50th percentile (the middle value), which is less sensitive to outliers than the mean.
std (**standard deviation**): Measures the amount of variation or dispersion from the average. A high standard deviation indicates that the data points are spread out over a wider range of values.

By incorporating these diverse functions, we transition from simply summarizing totals to performing a full statistical description of the grouped data. This capability is paramount in professional data science workflows where understanding data variance is as critical as understanding central tendency. The following example demonstrates how to aggregate the values in the `points` column using this broader set of metrics for each team:

#create pivot table to summarize several metrics for points by team
df.pivot_table(index='team', values='points',
aggfunc=('count', 'min', 'max', 'median', 'std'))

count max median min std
team
A 4 8 4.0 2 2.516611
B 4 9 6.0 5 1.914854
C 4 8 6.0 3 2 .629956

Analyzing this extended output provides deeper insights into performance variability. For instance, Team C, despite having an intermediate mean score, exhibits the highest standard deviation (2.63), suggesting their individual performances are the most spread out, indicating inconsistency. Conversely, Team B has the lowest standard deviation (1.91), indicating more consistent scoring across their observations, reinforcing their statistical leadership. The `count` column confirms that all teams have an equal number of data points (4), ensuring the statistics are comparable across groups.

Conclusion and Resources for Advanced Pivot Table Usage

The flexibility of the `pivot_table()` function, especially when utilizing multiple aggregation functions, represents a significant efficiency gain in data analysis. Not only does this method streamline code by consolidating multiple operations into a single command, but it also produces a highly readable, self-documenting output suitable for immediate reporting. Furthermore, the **aggfunc** parameter is versatile enough to handle custom functions (including those defined using `lambda`) or functions imported from libraries like NumPy, allowing analysts to perform virtually any statistical operation during the pivoting process.

Mastering the utilization of multiple **aggfunc** parameters is a critical skill for efficient data manipulation in Pandas. For developers seeking a comprehensive understanding of all supported parameters, customization options, and edge cases related to complex data summarization, consulting the official documentation is highly recommended.

Note: You can find the complete documentation for the Pandas **pivot_table()** function, which covers detailed usage of the **aggfunc** parameter, on the Pandas website.