How to Find the Minimum Value by Group in Pandas Easily

Authored by

stats writer

November 28, 2025

Efficiently performing statistical operations on segmented subsets of data is a fundamental requirement in modern data analysis. When working with the powerful Pandas library in Python, determining the minimum value within defined groups--such as finding the lowest sales per region or the youngest employee per department--is accomplished through a straightforward yet highly effective combination of methods. The primary approach involves using the essential groupby() function, followed directly by the aggregation function, min(), applied to the selected value column(s).

This technique leverages the classic "Split-Apply-Combine" paradigm, which is central to how Pandas handles complex statistical calculations. First, the groupby() method splits the source DataFrame into distinct groups based on the values in one or more specified columns. Second, the minimum calculation (the "Apply" step) is performed independently on the relevant numerical column within each segment. Finally, the results are combined into a new output structure, typically a Pandas Series or a DataFrame, where the grouping column becomes the index.

While the standard chain of groupby() and min() is the most direct solution, advanced users may also employ the versatile agg() function. The agg() function allows for greater control, enabling the application of minimum calculation alongside other statistics (like maximum, mean, or count) in a single, organized operation, providing a comprehensive view of the group statistics. Mastering these methods is crucial for efficient data manipulation in Pandas.

Core Methods for Calculating Group Minimums

To identify the minimum values segmented by categorical features within your DataFrame, Pandas provides two primary syntactical pathways based on whether you are analyzing a single metric or multiple metrics simultaneously. Both methods rely on the efficiency of the GroupBy object generated internally by Pandas.

Understanding these foundational structures is essential for writing robust and readable data processing scripts. The method is always applied to a Pandas DataFrame object, where you specify the column used for grouping and the column(s) containing the numerical data you wish to aggregate using the min() function.

The following methods detail the standard syntax used to find the minimum value by group in a Pandas DataFrame:

Method 1: Groupby minimum of one column

This syntax is suitable when you are interested in the minimum value of a single numerical column segmented by a single grouping column. This operation returns a Pandas Series where the index is the grouping column.

df.groupby('group_column').min()

Method 2: Groupby minimum of multiple columns

When multiple metrics require simultaneous minimum calculation against the same group, you must pass a list of column names using double brackets (`[]`) to the GroupBy object. This returns a DataFrame.

df.groupby('group_column').min()

Preparing the Sample DataFrame for Demonstration

To illustrate these aggregation techniques effectively, we will utilize a simple dataset representing hypothetical athletic statistics. This dataset contains information on three different teams (A, B, and C) and their corresponding performance metrics: `points` scored and `rebounds` achieved. The goal is to determine the lowest performance metric for each team, isolating the group minimums.

We begin by importing the Pandas library, which is standard practice for any data manipulation task in Python. Following the import, we construct the DataFrame, ensuring the data types are appropriate for aggregation--specifically, the `team` column must be categorical for grouping, while `points` and `rebounds` must be numerical (integer type, `int64`).

The creation process involves defining a dictionary where keys serve as column names and values are lists of corresponding data points. This resulting structure provides a clear, tabular format that allows us to apply the groupby() function and calculate the required minimums based on the `team` column.

The following code initializes the sample Pandas DataFrame used throughout the subsequent examples:

import pandas as pd

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

#display DataFrame
print(df)

team points rebounds

0 A 24 11
1 A 23 8
2 B 27 7
3 B 11 6
4 B 14 6
5 C 8 5
6 C 13 12

Example 1: Groupby Minimum of One Column (Points)

Our first example focuses on calculating the minimum value of a single metric: `points`. We aim to find the lowest recorded score for each distinct `team` within the dataset. This is the most straightforward application of grouped aggregation in Pandas, requiring the selection of the grouping column, the creation of the GroupBy object, the selection of the target value column, and the application of the min() function.

When the operation is executed, Pandas internally iterates through the groups (A, B, C). For every instance belonging to Team A, it finds the smallest value in the `points` column (23). It repeats this process for Team B (11) and Team C (8). The result is a concise Pandas Series, where the original grouping column (`team`) is automatically promoted to the index of the resulting output, providing an immediate summary of minimum performance per group.

This method is highly efficient, especially when dealing with very large datasets, as Pandas operations are optimized for speed through vectorized processing. It prevents the need for manual loops or conditional filtering, providing a clean, one-line solution for segmented minimum finding.

The following code shows how to find the minimum value of the **points** column, grouped by the **team** column:

#find minimum value of points, grouped by team
df.groupby('team').min()

team
A 23
B 11
C 8
Name: points, dtype: int64

From the output, we can clearly interpret the aggregated results:

The minimum value of points recorded for team A is **23**.
The minimum value of points recorded for team B is **11**.
The minimum value of points recorded for team C is **8**.

Example 2: Aggregating Minimum Values Across Multiple Columns

Often, data analysts need to derive multiple summary statistics--in this case, minimums for both `points` and `rebounds`--for the same set of groups. This requires a slight modification to the syntax used in Example 1. Instead of selecting a single column, we pass a list of column names (e.g.,) immediately after generating the GroupBy object.

Crucially, this list must be contained within **double brackets** (`df.groupby('team')]`). The outer brackets signify indexing or column selection on the GroupBy object, while the inner brackets define the Python list of columns to be aggregated. If single brackets were used, Pandas would attempt to return a Series, leading to an error since multiple columns were specified.

When multiple columns are aggregated, the output is no longer a simple Series but a structured DataFrame. This resulting DataFrame maintains the grouping column (`team`) as its index and includes separate columns for each aggregated metric (`points` and `rebounds`), making side-by-side comparison of group minimums straightforward.

The following code shows how to find the minimum value of the **points** and **rebounds** columns, grouped by the **team** column:

#find minimum value of points and rebounds, grouped by team
df.groupby('team')].min()

points rebounds
team
A 23 8
B 11 6
C 8 5

Analyzing the resulting DataFrame provides comprehensive minimum statistics for each team:

Team A:

Minimum points recorded: **23**
Minimum rebounds recorded: **8**

Team B:

Minimum points recorded: **11**
Minimum rebounds recorded: **6**

Team C:

Minimum points recorded: **8**
Minimum rebounds recorded: **5**

Note: It is critically important that you use double brackets when specifying multiple value columns, otherwise Pandas interprets the selection as an attempt to select a single column using a list, resulting in a KeyError.

Leveraging the .agg() Function for Complex Grouped Minimums

While chaining groupby() with min() is fast and direct, the agg() function offers superior flexibility, particularly when needing to apply several different aggregation functions simultaneously or applying different functions to different columns. For finding minimums, agg() can accept `'min'` as a string argument.

For instance, one might want to find the minimum points, but the maximum rebounds, grouped by team. The standard chained method would require two separate operations, but agg() allows this to be handled in a single, dictionary-based input. The dictionary maps column names to the list of desired functions.

If only the minimum is required, using `df.groupby('team').agg('min')` on all numerical columns yields the same result as the chained method shown in Example 2, but provides an entry point for future scaling. This approach is highly recommended for projects where the required statistical summary may evolve over time, as it centralizes the aggregation logic.

Understanding the Index and Output Structure

A key concept to grasp when performing groupby() operations is the transformation of the output structure. By default, the column(s) used for grouping are moved from being regular data columns to becoming the index of the resulting object. This is typically desirable, as it clearly organizes the aggregated results by the category used for segmentation.

As demonstrated in the examples, when aggregating a single column (Example 1), the output is a Pandas Series indexed by `team`. When aggregating multiple columns (Example 2), the output is a DataFrame, also indexed by `team`. If a user needs the grouping column (`team`) to remain a standard column rather than the index, the `as_index=False` parameter must be passed into the groupby() call.

This index handling behavior reinforces the conceptual link between the group identifier and the calculated summary statistic, making the results immediately accessible and intuitive for subsequent data merging or analysis tasks.

Summary and Best Practices

Finding the minimum value by group in Pandas is an essential skill for any data professional utilizing Python for data analysis. The core methodology relies on the powerful "Split-Apply-Combine" mechanism implemented via the groupby() method.

For simple, single-column minimum calculations, the direct chaining of groupby(), column selection, and min() is the cleanest and fastest approach. When working with multiple columns or requiring a mix of aggregation functions (e.g., min and max), the agg() function provides the necessary versatility and control.

Always remember the critical syntax difference between selecting a single column (single brackets) which returns a Series, versus selecting multiple columns (double brackets) which returns a DataFrame. Adhering to these syntactical conventions ensures that your Pandas aggregation operations are clean, efficient, and yield the expected summarized output structure.

How to Calculate the Sum of Columns in Pandas
How to Calculate the Mean of Columns in Pandas
How to Find the Max Value of Columns in Pandas