How to Find the Max Value by Group in Pandas

Authored by

stats writer

December 16, 2025

The ability to perform sophisticated data analysis hinges on effective data aggregation. In the realm of Python data science, the Pandas library provides highly optimized tools for this purpose. Specifically, determining the maximum value within various subgroups of a dataset is a fundamental operation. This process, often referred to as grouped aggregation, is elegantly handled using a combination of the built-in `groupby()` and `max()` methods.

Understanding how these functions interact is crucial for efficient data manipulation. The `groupby()` function initiates the "split" stage, segmenting a large DataFrame into distinct groups based on the values of one or more specified categorical columns. Following this segmentation, the `max()` function is applied to the resulting groups, performing the "apply" step to calculate the highest numerical value for the remaining columns within each subgroup. This robust framework allows analysts to quickly transform raw data into summarized insights, identifying the absolute peaks across predefined categories.

Mastering Grouped Aggregation in Pandas

Data analysis frequently demands the identification of extreme values conditional on specific categories. Whether you are analyzing sales performance by region, student scores by class, or, as we will demonstrate, athletic metrics by team, finding the maximum value by group in a Pandas DataFrame is a core skill. This powerful technique is central to summarizing data effectively and drawing meaningful comparisons between subsets.

The operation relies on the "split-apply-combine" paradigm, a methodology where data is first split into groups, an analytical function (in this case, `max()`) is applied to each group independently, and the results are finally combined into a single, cohesive output structure. This approach ensures that the calculation of the maximum value is strictly contained within the boundaries of the defined groups, preventing cross-group interference and producing statistically valid aggregation results.

The Core Syntax for Max Grouping

Fortunately, the syntax for executing this critical operation in Pandas is remarkably intuitive, following the standard methodology of chaining methods directly onto the DataFrame object. We first invoke the `groupby()` method, specifying the column(s) that define our groups, and then immediately call the `max()` method to execute the aggregation.

The general structure for finding maximums based on group membership is illustrated below. Note that `column_name` refers to the categorical column used for grouping, while the `max()` function automatically applies the maximum calculation to all suitable numerical columns not included in the grouping key, returning the maximal entry for each metric within that group.

df.groupby('column_name').max()

This concise syntax allows for rapid data transformation. We will explore practical applications of this structure in the subsequent examples, ensuring a complete understanding of how to tailor the output for specific analytical needs.

Preparing the Demonstration DataFrame

To effectively demonstrate these methods, we will utilize a sample dataset tracking team performance metrics. This DataFrame contains three critical columns: `team` (our grouping variable), `points`, and `rebounds`. By analyzing this data, we can identify which team achieved the highest score and the highest rebound count across all their recorded entries.

The setup code below imports the necessary Pandas library and constructs the sample DataFrame. Understanding the structure of the input data is the first step toward successful data aggregation. We must initialize our environment before executing the grouping operations.

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

As shown in the output, the dataset contains seven individual records distributed across three unique teams (A, B, and C). Our objective now is to determine the single highest recorded value for `points` and `rebounds` for each of these three teams independently, using robust `groupby()` techniques.

Example 1: Finding Maximums Across Multiple Columns

When applying the `max()` function immediately after `groupby()` without explicitly selecting a numerical column, Pandas automatically calculates the maximum value for every remaining numerical column in the DataFrame. This method is highly efficient when you need a comprehensive summary of maximum metrics per group, as it aggregates all applicable data fields simultaneously.

The following code snippet demonstrates how to find the maximal scores for both `points` and `rebounds`, grouped entirely by the `team` variable. This provides a complete performance profile for each team based on their highest recorded values in both categories.

#find max values of points and rebounds, grouped by team
df.groupby('team').max().reset_index()

team points rebounds
0 A 24 11
1 B 27 7
2 C 13 12

The Role of reset_index() in Grouped Output

The resulting output is a clean summary that clearly isolates the maximum performance achieved by each team across the measured variables. This aggregated view is significantly more manageable than scanning the original raw data.

We can derive the following key insights from this aggregated table:

Team A recorded a maximum **points** value of 24 and achieved a peak **rebounds** count of 11 across its records.
Team B demonstrated the highest individual scoring performance with a maximum **points** value of 27, though their maximum **rebounds** value remained lower at 7.
Team C, while having the lowest max **points** value (13), recorded the highest maximum **rebounds** value in the entire dataset at 12.

The crucial final step is the invocation of the `reset_index()` function. When `groupby()` is executed, the grouping column (`team`) is automatically promoted to become the index of the resulting Series or DataFrame. Using `reset_index()` explicitly converts this index back into a standard column, resulting in the clean, sequentially indexed DataFrame seen in the output, which is generally preferred for further analysis, merging, or reporting, as it treats the group identifier like

any other piece of data.

Example 2: Targeting a Specific Column for Maximum Calculation

In many analytical scenarios, the goal is not to calculate maximums for all numerical columns, but rather to focus the aggregation specifically on one target metric. This refinement is achieved by using standard column selection bracket notation immediately following the `groupby()` call, but before the `max()` method is applied. This approach is highly recommended when dealing with wide DataFrames containing numerous unnecessary columns, as it streamlines the aggregation process and minimizes memory usage.

This targeted approach returns a Pandas Series containing the maximum values for the chosen column, indexed by the group key. By chaining `reset_index()`, we convert this Series back into a two-column DataFrame (Group Key and Max Value). The following example shows how to calculate only the maximum `points` achieved by each team.

#find max value of points, grouped by team
df.groupby('team').max().reset_index()

team points
0 A 24
1 B 27
2 C 13

This focused operation confirms that Team B holds the highest single score. This methodology is critical when performing column-specific analyses where multi-column aggregation would obscure the target metric or introduce complexity due to differing scales and units across metrics.

Example 3: Ordering Results Using sort_values()

Once the maximum values per group have been calculated, it is often necessary to rank or sort the groups based on these new aggregated metrics. Pandas facilitates this through the `sort_values()` function, which can be seamlessly chained onto the resulting DataFrame generated by the `groupby().max().reset_index()` sequence. This ability to chain multiple operations is a hallmark of efficient data manipulation in Python.

To rank the teams based on their maximum `points` achieved, from the highest score down to the lowest, we must specify the sorting column (`points`) and set the `ascending` parameter to `False`. This descending sort order instantly highlights the top-performing group, providing immediate insight into relative group success based on the maximal metric.

#find max value by team, sort descending
df.groupby('team').max().reset_index().sort_values(, ascending=False)

team points
1 B 27
0 A 24
2 C 13

The output clearly shows Team B as the group with the highest individual points record (27), followed by Team A (24), and then Team C (13). This composite operation--grouping, aggregation, un-indexing, and sorting--demonstrates the fluidity and power of method chaining in Pandas for producing clean, ranked summaries.

Sorting Results in Ascending Order

Conversely, if the analytical goal requires identifying the lowest maximum achieved by any group, perhaps to identify groups with consistently lower peak performances, we can set the `ascending` parameter to `True` within the `sort_values()` function. This modification sorts the results from smallest to largest maximum value.

Sorting in ascending order can be valuable for identifying groups that performed poorly relative to their peers or for verifying data integrity where unexpected low maximums might indicate issues in the underlying data collection.

#find max value by team, sort ascending
df.groupby('team').max().reset_index().sort_values(, ascending=True)

team points
2 C 13
0 A 24
1 B 27

Summary of Pandas Grouped Maximums

The combination of `groupby()` and `max()` provides a foundational statistical tool for any Pandas user. By utilizing these methods, we can transform granular, row-level data into concise, group-level summaries. Key to generating clean output are two steps: selecting only the necessary columns before aggregation (as shown in Example 2) and correctly resetting the index after the aggregation step using `reset_index()` to ensure the grouping key is treated as standard data.

Mastering this technique allows for rapid identification of outliers, peak performance, and comparative metrics across distinct categories within complex datasets, making it an indispensable part of the data science workflow. This methodology can be easily adapted to calculate other statistics, such as means (`mean()`) or sums (`sum()`), simply by swapping out the aggregation function.

If you are interested in exploring other fundamental data aggregation techniques in Pandas, consider reviewing the following related tutorials:

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