Pandas: How to Find the Maximum Value Across Multiple Columns in a DataFrame

Authored by

Mohammed looti

October 28, 2025

When analyzing complex datasets stored within the pandas DataFrame structure, a frequent requirement is determining the maximum value horizontally, or row-wise, across a specified subset of columns. This operation is fundamental in tasks such as feature engineering, identifying peak performance indicators, or flagging outlier data points within a record. Fortunately, the **pandas** library offers robust and highly efficient methods to achieve this calculation using the built-in aggregation functions combined with precise axis specification. We will explore two primary methods for calculating and utilizing these row-wise maximums, ensuring clarity and performance in your data analysis workflow.

The core of performing any calculation across multiple columns row-wise relies on two essential components: first, correctly selecting the columns of interest, and second, applying the aggregation function with the correct dimensional argument. Understanding the concept of the **axis parameter** is critical for mastering this operation. By leveraging the .max() method, you can quickly generate a new data structure containing only the highest value found across the chosen fields for every single row in your dataset.

The following methods illustrate the standard, idiomatic approach in **pandas** for performing this calculation. These techniques are highly scalable and maintainable, making them the preferred choice for data scientists and analysts working with large volumes of tabular data. We will first demonstrate how to retrieve the maximum values as a standalone Series, and subsequently, how to integrate this derived information back into the original DataFrame as a new, insightful column.

Understanding Row-Wise Operations and the Axis Parameter

Before diving into the code, it is essential to grasp the role of the **axis parameter** within pandas DataFrame aggregation functions. In **pandas**, data operations typically occur along an axis. By default, most aggregation functions operate along `axis=0`, which means they calculate results column-wise (i.e., finding the maximum of all values *within* a single column). To calculate the maximum value across multiple columns, comparing values horizontally within a single record or row, we must explicitly specify the axis=1 parameter.

Setting axis=1 fundamentally changes the direction of the calculation. Instead of collapsing the rows to produce a single result per column, we collapse the columns (the selected subset) to produce a single result per row. This row-wise application is key to feature creation and comparison across different metrics that describe the same entity (the row). For instance, if you have columns representing 'Score A', 'Score B', and 'Score C', using `axis=1` tells **pandas** to look at the three scores for the first entry, find the highest one, then repeat this process for the second entry, and so on.

While the concept of the axis might seem simple, correctly specifying it is the single most common point of error when performing multidimensional data analysis in Python. Always remember:

axis=0 refers to the index (rows), and operations along this axis are vertical; **axis=1** refers to the columns, and operations along this axis are horizontal. Leveraging this parameter correctly ensures that the powerful vectorized operations provided by **pandas** are applied efficiently to yield the desired row-level results.

Method 1: Retrieving the Maximum Value as a Standalone Series

The first method focuses on isolating the calculation of the maximum value for specific columns, returning the result as a new Series object. This approach is highly useful when you only need to inspect or use the maximum values temporarily without necessarily integrating them immediately into the main pandas DataFrame. The operation involves two steps: subsetting the DataFrame to include only the relevant columns, and then applying the .max() function with the row-wise axis defined.

The syntax below represents the clean, concise method for executing this task. We pass a list of column names (e.g., `col1`, `col2`, `col3`) to the indexing operator (`df`), which extracts a temporary DataFrame subset. The .max() function is then chained onto this temporary subset, forcing the calculation to proceed horizontally across the specified columns for every row:

df].max(axis=1)

Understanding the output structure is paramount. Since we are aggregating the column dimension (`axis=1`), the result is a one-dimensional structure--a Series. This resulting Series maintains the index of the original DataFrame, ensuring that the maximum value calculated for Row 0 remains correctly associated with the index label 0, and so forth. This index preservation is what allows us to seamlessly map these maximum values back to the original records if required, which is precisely what Method 2 accomplishes.

Practical Demonstration: Setting up the DataFrame

To effectively demonstrate these methods, we will initialize a sample pandas DataFrame representing statistics for several fictional players. This dataset includes columns for player identification, as well as three quantitative metrics: `points`, `rebounds`, and `assists`. This structure provides a realistic scenario where we might want to find a player's single highest achievement across two or more categories within a given game or season.

We begin by importing the **pandas** library, which is standard practice in any data manipulation script involving DataFrames. We use the conventional alias `pd` for brevity. Following the import pandas statement, the DataFrame is created using a Python dictionary structure, mapping column names to lists of corresponding data values. This setup ensures that we have a clean, reproducible

dataset for the subsequent examples.

The creation and initial display of the DataFrame are shown in the code block below. This provides a clear baseline for verifying the results of our row-wise maximum calculations later on. We will focus our subsequent examples on the `points` and `rebounds` columns to determine the highest contribution metric for each player in those two categories.

import pandas as pd

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

#view DataFrame
print(df)

player points rebounds assists
0 A 28 5 10
1 B 17 6 13
2 C 19 4 7
3 D 14 7 8
4 E 23 14 4
5 F 26 12 5
6 G 5 9 8

As demonstrated in the output, Player A achieved 28 points and 5 rebounds, while Player G achieved 5 points and 9 rebounds. Our goal is to programmatically identify the higher of these two values for every player, leveraging the efficiency of **pandas** vectorized operations rather than iterating through rows manually.

Example 1: Isolating the Maximum Value

Utilizing Method 1, we will now execute the command to find the maximum value in each row, specifically across the `points` and `rebounds` columns. The code below first selects these two columns and then applies the .max() function, ensuring that the critical axis=1 parameter is set to perform the row-wise comparison.

The resulting output is a Series structure indexed identically to the original DataFrame, where each value represents the larger of the two metrics for that specific player. This output is immediate and

highly efficient, especially when dealing with millions of rows, as **pandas** is optimized for these types of columnar operations.

#find max value across points and rebounds columns
df].max(axis=1)

0 28
1 17
2 19
3 14
4 23
5 26
6 9
dtype: int64

The interpretation of the output is straightforward and confirms the successful application of the row-wise calculation. For instance, examining the first three rows clarifies the comparison logic:

The maximum value across the `points` (28) and `rebounds` (5) columns for the first row (Player A) was **28**.
The maximum value across the `points` (17) and `rebounds` (6) columns for the second row (Player B) was **17**.
The maximum value across the `points` (19) and `rebounds` (4) columns for the third row (Player C) was **19**.

Note that for Player G (index 6), the points were 5 and rebounds were 9. The resulting maximum value is correctly reported as **9**, demonstrating that the function is indeed comparing values horizontally across the columns and not simply selecting the maximum value from the entire dataset. This intermediate Series is a powerful result on its own, but often, analysts need to integrate this value directly into the DataFrame for further analysis or reporting.

Example 2: Integrating the Maximum Value into the DataFrame

The second and often more practical method involves creating a new column in the existing DataFrame to store the calculated row-wise maximum values. This process, known as **feature creation** or **feature engineering**, adds analytical value by providing a derived metric alongside the original raw data. By assigning the result of the `df.max(axis=1)` operation directly to a new column name using standard **pandas** assignment syntax, we seamlessly integrate the Series of maximums.

This technique ensures that the new column, which we title `max_points_rebs`, is perfectly aligned

with the existing data due to the preservation of the index during the .max() calculation. This alignment is automatic and robust, provided the original DataFrame has not been modified between the calculation and the assignment steps. This is a common pattern in data analysis: calculate a metric, and then append it to the dataset for subsequent filtering, grouping, or visualization.

The following code block demonstrates this feature creation process. We define the new column name and use the identical row-wise maximum calculation from Example 1 as the source for the values. We then print the updated DataFrame to verify the successful addition and population of the new column.

#add new column that contains max value across points and rebounds columns
df = df].max(axis=1)

#view updated DataFrame
print(df)

player points rebounds assists max_points_rebs
0 A 28 5 10 28
1 B 17 6 13 17
2 C 19 4 7 19
3 D 14 7 8 14
4 E 23 14 4 23
5 F 26 12 5 26
6 G 5 9 8 9

As is evident from the output, the new column titled `max_points_rebs` now serves as a derived feature, storing the highest value across the `points` and `rebounds` metrics for each respective player. This feature is immediately ready for use in advanced statistical modeling or reporting dashboards, allowing analysts to quickly identify which of the two tracked performance indicators was dominant for each observation.

Advanced Considerations: Alternative Functions and Performance

While the .max() function provides the value itself, there are scenarios where analysts might need to know which column contributed that maximum value. For this, **pandas** offers the `.idxmax()` method. When applied with axis=1, `.idxmax()` returns the column label (name) that holds the maximum value for that row. This is incredibly useful for dynamic analysis, such as determining which specific metric was the peak performer for a given entity.

Furthermore, for extremely large datasets where performance is paramount, data scientists often

look to underlying libraries like **NumPy**. The `numpy.max()` function can be applied directly to the array representation of the DataFrame subset (`df].values`). However, for the vast majority of standard data analysis tasks, the native **pandas** approach using `df.max(axis=1)` is sufficiently fast due to **pandas**' optimized C-based backend, and it maintains higher readability and integration with the DataFrame structure.

In summary, choosing between the two demonstrated methods depends on the downstream task: use Method 1 (standalone Series) for temporary inspection or calculation, and use Method 2 (column assignment) for permanent feature creation within the pandas DataFrame. Mastery of the axis=1 parameter ensures that these powerful row-wise aggregations are performed accurately and efficiently.

Additional Resources

The following tutorials explain how to perform other common tasks in pandas, providing a broader context for data manipulation and analysis:

Pandas Tutorial: Subsetting Data
Pandas Documentation: idxmax()
Pandas Documentation: apply() method