How to Easily Understand axis=0 and axis=1 in Pandas

Authored by

stats writer

December 3, 2025

The **axis** parameter in Pandas is fundamental for controlling the direction of data manipulation operations. It specifies whether an operation should be applied vertically across rows or horizontally across columns of a DataFrame. Misunderstanding this simple parameter is a common hurdle for new data scientists, but grasping its logic unlocks powerful data manipulation capabilities.

When you execute a function that supports the **axis** argument, you are essentially telling the function which dimension of the data structure should be collapsed or iterated over. This is critical for aggregation methods like summing, calculating means, or dropping null values, where the result is a reduction of the original dataset along one dimension.

In simple terms, `axis=0` and `axis=1` define the directional flow of the calculation. For instance, when using the `.sum()` method, specifying `axis=0` sums values down the columns, yielding a result for each column. Conversely, `axis=1` sums values across the rows, producing a result for each row entry.

The Core Conceptual Difference: axis=0 vs axis=1

Many essential functions in Pandas require specifying an axis along which to apply a calculation or modification. While the terminology can sometimes be confusing (rows vs. columns), understanding the dimensionality helps clarify the operation.

The following widely accepted rule of thumb governs the use of the **axis** parameter in most DataFrame operations:

axis=0: This represents the index (rows). Operations are applied **column-wise**, meaning the calculation proceeds vertically down each column. This is the default behavior for most aggregation methods.
axis=1: This represents the columns. Operations are applied **row-wise**, meaning the calculation proceeds horizontally across each row.

Think of the axis number as the dimension being **acted upon** or the dimension that is **collapsed**. When you use `axis=0`, the row index (0) is collapsed, leaving the column index (1) in the output. When you use `axis=1`, the column index (1) is collapsed, leaving the row index (0) in the output.

Practical Demonstration: Setting up the Pandas DataFrame

To illustrate how the **axis** argument works in practice, we will use the following example Pandas DataFrame, which contains statistics for several fictional sports teams (A, B, and C). We first import the necessary library and create the data structure.

import pandas as pd

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

#view DataFrame
df

team points assists rebounds
0 A 25 5 11
1 A 12 7 8
2 B 15 7 10
3 B 14 9 6
4 B 19 12 6
5 B 23 9 5
6 C 25 9 9
7 C 29 4 12

This DataFrame, named `df`, provides the foundation for our examples demonstrating how the **axis** parameter influences aggregation calculations like mean, sum, and max.

Example 1: Calculating Central Tendency (Mean) along Different Axes

The calculation of the mean (average) is a common analytical task. By using the `.mean()` method, we can quickly derive insights, but the axis setting dramatically changes the result's context.

We use **axis=0** to find the mean of each column in the DataFrame, effectively calculating the average down the rows:

#find mean of each column
df.mean(axis=0)

points 20.250
assists 7.750
rebounds 8.375
dtype: float64

The output shows the mean value of each numeric column. Notice that Pandas automatically

avoids calculating the mean of the 'team' column because it contains character (string) data.

We can also use **axis=1** to find the mean of each row in the DataFrame, calculating the average horizontally across the numerical features:

#find mean of each row
df.mean(axis=1)

0 13.666667
1 9.000000
2 10.666667
3 9.666667
4 12.333333
5 12.333333
6 14.333333
7 15.000000
dtype: float64

From the output, we see a new Series where each value is the average of the numerical scores for that row:

The mean value in the first row is **13.667**.
The mean value in the second row is **9.000**.
The mean value in the third row is **10.667**.

Example 2: Aggregation using Summation (.sum())

The `.sum()` method helps us calculate totals across the data. Setting the **axis** controls whether we calculate subtotals per column or per row.

We can use **axis=0** to find the sum of specific columns in the DataFrame, aggregating vertically:

#find sum of 'points' and 'assists' columns
df].sum(axis=0)

points 162
assists 62
dtype: int64

We can also use **axis=1** to find the sum of each row in the DataFrame, aggregating horizontally across the features:

#find sum of each row
df.sum(axis=1)

0 41
1 27
2 32
3 29
4 37
5 37
6 43
7 45
dtype: int64

Example 3: Finding Extreme Values (Max)

The `.max()` method determines the highest value along the designated dimension. This is useful for identifying outliers or performance peaks, and the interpretation depends heavily on whether `axis=0` or `axis=1` is applied.

We use **axis=0** to find the max value of specific columns in the DataFrame, identifying the highest score recorded for each category:

#find max of 'points', 'assists', and 'rebounds' columns
df].max(axis=0)

points 29
assists 12
rebounds 12
dtype: int64

We can also use **axis=1** to find the max value of each row in the DataFrame, identifying the highest metric achieved within that particular observation:

#find max of each row
df.max(axis=1)

0 25
1 12
2 15
3 14

```
4 19
5 23
6 25
7 29
dtype: int64
```

From the output we can see the peak score achieved in each observation:

The max value in the first row (index 0) is **25**.
The max value in the second row (index 1) is **12**.
The max value in the third row (index 2) is **15**.

Other Crucial Applications of the Axis Parameter

Beyond simple aggregation, the **axis** parameter is fundamental in data preparation and cleansing tasks, particularly when manipulating the structure of the DataFrame.

When using methods like `.drop()` to remove data, the axis designation is critical for determining whether rows or columns are eliminated. `axis=0` means the operation seeks the specified label in the index (row labels), while `axis=1` means it seeks the specified label in the columns (column headers).

For example, `df.drop(columns=, axis=1)` removes the entire 'team' column, acting horizontally across the columns. Conversely, using `axis=0` allows for the removal of observations based on their row index.

Conclusion: Summarizing Directionality

Mastering the distinction between `axis=0` and `axis=1` is crucial for efficient data manipulation in Pandas. While `axis=0` (the default) operates vertically down the index (rows), often yielding column results, `axis=1` operates horizontally across the columns, yielding row results.

The conceptual key is that the axis number refers to the dimension that is being reduced or collapsed by the operation. By consistently applying this concept, whether you are calculating the mean, finding the maximum, or simply dropping data, you can control precisely how your data transformations are executed.