Pandas: Sort Results of value_counts()

Authored by

Mohammed looti

April 12, 2026

The Pandas library is an indispensable tool for data analysis in Python, offering powerful and flexible data structures like the DataFrame. One of its frequently used functions is **value_counts()**, which efficiently calculates the frequency of unique values within a Series or a DataFrame column. This function is particularly useful for understanding the distribution of categorical or discrete data points, providing a quick summary of how often each unique item appears.

While **value_counts()** inherently sorts its results, you might encounter scenarios where the default sorting mechanism doesn't align with your analytical needs. For instance, you might want to see the least frequent items first, or perhaps maintain the original order in which unique values first appeared in your dataset. Understanding how to manipulate the sorting of these counts is crucial for tailoring your data summaries to specific requirements and enhancing the clarity of your data analysis.

This article will explore three distinct methods for sorting the output of the **value_counts()** function. We will delve into how to sort by descending order (the default behavior), by ascending order, and by the sequential appearance of unique values in the original DataFrame. Each method will be illustrated with practical examples, providing you with the knowledge to control the presentation of your frequency distributions effectively.

Understanding the Sample DataFrame

Before we dive into the various sorting techniques, let's establish a common dataset to ensure all our examples are consistent and easy to follow. We will create a simple Pandas DataFrame that simulates a small dataset, perfect for demonstrating the effects of different sorting approaches on the results of **value_counts()**. This dataset will contain two columns: 'team' (a categorical variable) and 'points' (a numerical variable).

The 'team' column, with its repeated values, will be our primary focus for applying **value_counts()**. By analyzing this column, we can see how many times each team appears, and subsequently, how to sort these counts according to our specific needs. The 'points' column serves to create a more realistic DataFrame structure, though it won't be directly involved in the frequency counting examples.

Below is the Python code to generate our sample DataFrame. This foundational step will allow us to execute and understand the sorting methods in a clear, reproducible manner.

import pandas as pd

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

```
#view DataFrame
print(df)

team points
0 A 15
1 A 12
2 B 18
3 B 20
4 B 22
5 B 28
6 B 35
7 C 40
```

As you can see from the output, our DataFrame contains eight rows, with three distinct teams: 'A', 'B', and 'C'. Now, let's proceed to apply and sort the results of the **value_counts()** function on the 'team' column using the different methods.

Method 1: Sorting Counts in Descending Order (Default Behavior)

The most common way to view frequency distributions is with the highest counts first. This provides an immediate understanding of the most prevalent categories within your dataset. The **value_counts()** function in Pandas offers this as its default behavior, simplifying the initial exploration of data distributions.

When you call **value_counts()** on a Series without any additional arguments for sorting, it automatically arranges the resulting frequencies from the largest count to the smallest. This default descending order sorting is incredibly intuitive for tasks such as identifying top performers, most common items, or dominant categories in a dataset. It's often the first step in tasks like data cleaning or exploratory data analysis.

Let's apply this method to our sample DataFrame's 'team' column. The following code snippet demonstrates how to achieve this, showing the concise nature of Pandas for such operations:

```
df.my_column.value_counts()
```

In our specific example, to count the occurrences of each unique value in the 'team' column and sort them in descending order, we execute the following:

```
#count occurrences of each value in team column and sort in descending order
df.team.value_counts()
```

```
B 5
A 2
C 1
Name: team, dtype: int64
```

As the output clearly indicates, 'B' is the most frequent team with 5 occurrences, followed by 'A' with 2, and 'C' with 1. This immediate presentation of the most frequent items first makes it easy to grasp the primary characteristics of the 'team' distribution at a glance.

Method 2: Sorting Counts in Ascending Order

While viewing counts in descending order is often the default and most intuitive, there are many situations where sorting in ascending order becomes more valuable. For instance, when you're looking for anomalies, rare events, or categories with the lowest participation, presenting the least frequent items first can highlight crucial insights that might otherwise be overlooked. This approach is particularly useful in quality control, fraud detection, or identifying niche segments.

To sort the results of **value_counts()** in ascending order, we can chain the **sort_values()** method directly after the **value_counts()** call. The **sort_values()** method, when called without any arguments, sorts the Series (which is the output of **value_counts()**) in ascending order by its values. This provides a straightforward way to flip the default sorting behavior.

The general syntax for this operation is as follows:

```
df.my_column.value_counts().sort_values()
```

Applying this to our 'team' column in the sample DataFrame, we can count the occurrences and then arrange them from the least frequent to the most frequent. This example showcases how easily you can adapt the output to highlight different aspects of your data:

```
#count occurrences of each value in team column and sort in ascending order
df.team.value_counts().sort_values()
```

```
C 1
A 2
B 5
Name: team, dtype: int64
```

Here, the output now presents team 'C' first, with its single occurrence, followed by 'A' and then 'B'. This ordering immediately draws attention to the less common categories, which can be invaluable

for specific analytical objectives. The flexibility offered by chaining **sort_values()** allows for quick adjustments to your data presentation.

Method 3: Sorting Counts by Their Appearance Order in the DataFrame

Sometimes, the intrinsic order of values within your original DataFrame holds significant meaning. For instance, if your categorical data represents stages in a process, items introduced chronologically, or predefined levels that don't necessarily correspond to alphabetical or numerical order, you might want your frequency counts to reflect this original sequence. Standard sorting methods (alphabetical, numerical, ascending, descending order) would disrupt this inherent order, potentially obscuring important context.

To sort the results of **value_counts()** based on the order in which unique values first appear in the DataFrame column, a slightly more advanced technique is required. This involves first identifying the unique values in their order of appearance using the **unique()** method, and then using this ordered array to reindex the result of **value_counts()**. This ensures that the output preserves the observational sequence of your data.

The general approach involves calling **value_counts()** and then indexing the resulting Series with the ordered unique values obtained from the column using **unique()**. The syntax looks like this:

df.my_column.value_counts()

Let's apply this method to our 'team' column. Observe how the output reflects the initial appearance order of teams 'A', 'B', and 'C' in our original DataFrame:

#count occurrences of each value in team column and sort in order they appear
df.team.value_counts()

A 2
B 5
C 1
Name: team, dtype: int64

In this output, the counts are now sorted based on the order in which the unique values (teams 'A', 'B', then 'C') first appeared in the 'team' column of our DataFrame. This method is particularly powerful when the sequence of categories itself conveys important information, allowing your frequency distribution to align with the narrative or structure of your raw data.

Conclusion and Further Exploration

The **value_counts()** function in Pandas is a versatile tool for summarizing the frequency of unique items within a Series. As demonstrated, while it defaults to sorting counts in descending order, Pandas provides elegant solutions for customizing this output. Whether you need to highlight the least frequent items using ascending order or preserve the contextual order of appearance from your original DataFrame, these methods offer the flexibility required for nuanced data analysis.

Mastering these sorting techniques not only enhances the readability and interpretability of your frequency distributions but also empowers you to extract more specific insights from your datasets. By choosing the appropriate sorting method, you can tailor your data summaries to precisely match the analytical questions you are trying to answer, making your Pandas workflows more efficient and effective.

For those eager to deepen their Pandas expertise, consider exploring the official Pandas documentation for more advanced functionalities of **value_counts()**, such as the `normalize` parameter for relative frequencies or the `dropna` parameter for handling missing values. Continuously refining your skills with such fundamental functions will significantly improve your data analysis capabilities.