How to Easily Plot Value Counts in Pandas

Authored by

stats writer

November 22, 2025

Plotting value counts in Pandas is a simple process, yet it forms the foundation of crucial exploratory data analysis (EDA). The core methodology involves two chained operations: first, leveraging the .value_counts() method on a specific column of a dataset to obtain the frequency distribution of each unique entry, and subsequently, visualizing these numerical results using the built-in .plot() method. This streamlined workflow instantly provides a clear, basic graphical representation of the frequency distribution, allowing analysts to quickly grasp data bias and category prominence. Furthermore, sophisticated users can easily adjust the chart type and other visual parameters to customize the plot extensively. The true efficiency of this approach lies in the ability to chain the .value_counts() and .plot() methods together, enabling the creation of complex visualizations in a single, highly readable line of Python code.

One of the most essential tasks in data preprocessing and exploratory analysis is understanding the distribution of categorical or discrete data points within a column. To facilitate this, the Pandas library provides the powerful **.value_counts()** function. This method is specifically engineered to count the occurrences of every unique value present in a selected column of a DataFrame, returning the results as a new Series object. The resulting Series is typically indexed by the unique values themselves and contains the counts as its data values, making it perfectly structured for immediate visualization. This initial step of aggregation is fundamental before any plotting occurs, transforming raw data into meaningful frequency statistics.

Once the frequency distribution is calculated using **.value_counts()**, the next logical step is visualization. While the raw counts are useful, a graphical representation, such as a bar chart, offers unparalleled insight into the data's shape and skewness. Analysts can utilize one of several methods to plot these resulting frequency distributions. The choice of plotting method often dictates the final visual presentation, particularly concerning the sorting order of the bars. Below, we detail the three primary techniques employed to plot the Series produced by the **.value_counts()** function, each serving a distinct analytical requirement regarding how the categories should be ordered along the x-axis.

Fundamental Approaches to Plotting Value Counts

When generating visualizations from frequency data, controlling the order in which categories appear is paramount for effective communication. The default behavior in Pandas often prioritizes simplicity, but specific analytical tasks require explicit control over sorting. We outline three robust methods to manage the visualization order, ensuring that the resulting bar chart aligns precisely with the underlying analytical question. These methods involve either leveraging the default behavior, implementing an explicit sorting step, or utilizing the original column's unique value

sequence.

Method 1: Plot Value Counts in Descending Order

This is the standard, default behavior when chaining the .value_counts() and .plot() methods. The bars are automatically arranged from the highest frequency to the lowest, providing an immediate overview of the most dominant categories in the dataset.

df.my_column.value_counts().plot(kind='bar')

Method 2: Plot Value Counts in Ascending Order

To reverse the default sorting, we introduce the **.sort_values()** method into the chain. By default, **.sort_values()** sorts in ascending order, arranging the resulting visualization from the least frequent category to the most frequent. This is particularly useful when analyzing outliers or less represented data points.

df.my_column.value_counts().sort_values().plot(kind='bar')

Method 3: Plot Value Counts in Order They Appear in DataFrame

Sometimes, the analyst requires the visualization to follow the chronological or structural order in which the unique values first appear in the original DataFrame, rather than sorting by frequency. This requires a more complex indexing operation using the **.unique()** method to enforce the original sequence onto the frequency counts before plotting.

df.my_column.value_counts().plot(kind='bar')

To illustrate these distinct methodologies practically, we will utilize a small, representative Pandas DataFrame designed to showcase category distributions. The following setup demonstrates the necessary imports, the creation of the sample data, and the preliminary calculation of occurrences using the **.value_counts()** method, which serves as the input data for all subsequent plotting examples. Understanding this foundational output is key to interpreting the visual results produced by each of the three plotting techniques.

Practical Pandas Implementation Setup

Before generating any plots, a robust demonstration requires a concrete dataset. We begin by importing the Pandas library, typically aliased as pd, which is standard practice in data science workflows. Following the import, a small sample DataFrame named `df` is constructed. This sample

dataset contains two columns: `team` (a categorical variable) and `points` (a numerical variable). The core focus of our plotting exercises will be the distribution within the `team` column, which exhibits varying levels of repetition for values A, B, and C.

The initial step after creating and displaying the DataFrame is to calculate the raw frequencies for our target column. We apply `df.team.value_counts()` to the 'team' column. This operation immediately returns a Series indexed by the unique team names, showing how many times each team appears. This resultant Series is the crucial data structure that the **.plot()** method will consume to draw the bar chart. As demonstrated in the output below, Team B appears 5 times, Team A appears twice, and Team C appears only once.

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

#calculate occurrences of each value in 'team' column
df.team.value_counts()

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

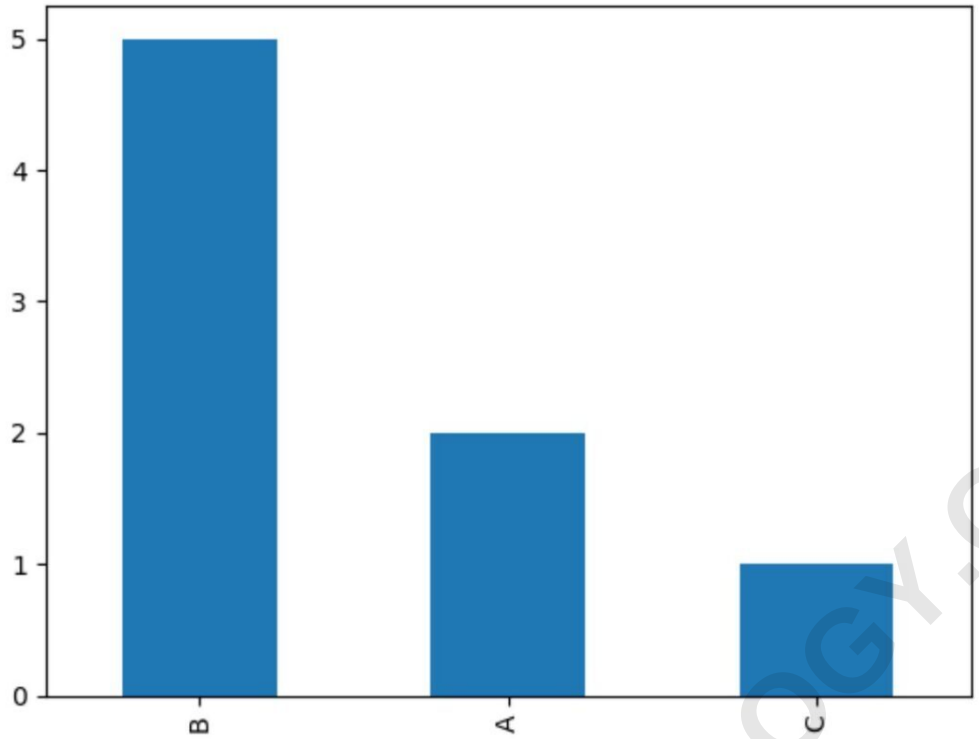
Example 1: Plot Value Counts in Descending Order

Visualizing Default Frequency Sorting

The most common requirement in frequency analysis is identifying the categories with the highest occurrence rates. Pandas simplifies this by implementing a default descending sort order when the **.value_counts()** output is directly piped into the **.plot()** method. This sequence ensures that the resulting bar chart immediately highlights the most dominant values. This approach is highly efficient for quick, high-level data summaries. The following code executes this fundamental plotting operation, producing a visualization where the bars are ordered based on height, from tallest to shortest, corresponding to the category counts.

The syntax for this operation is concise and demonstrates the powerful chaining capabilities within the Pandas ecosystem. We select the target column (`df.team`), calculate the frequencies (`.value_counts()`), and immediately call the plotting function (`.plot()`), specifying the visualization type as a bar chart (`kind='bar'`). Note that no explicit sorting method, such as `.sort_values()`, is necessary here because the frequency counting function inherently returns results sorted by frequency in descending order.

#plot value counts of team in descending order
df.team.value_counts().plot(kind='bar')



Upon inspecting the resulting bar chart, the structure clearly confirms the sorting mechanism. The x-axis accurately displays the unique team names (B, A, C), while the y-axis represents the absolute frequency or count of each team's appearance in the DataFrame. Specifically, Team B (count 5) is placed first, followed by Team A (count 2), and finally Team C (count 1). This confirms that, by default, the bars are rigorously sorted in descending order of their associated frequency count, offering immediate visual confirmation of the data's dominant categories.

Note: Should the analytical need arise for a vertical orientation instead of the standard vertical bars, the transformation is straightforward. To create a horizontal bar chart--useful when category names are long--simply replace the **bar** value with **barh** (for 'bar horizontal') within the **kind** argument of the **.plot()** method. This flexibility allows for rapid adaptation of visualizations to optimize clarity and readability based on the specific dataset characteristics.

Example 2: Plot Value Counts in Ascending Order

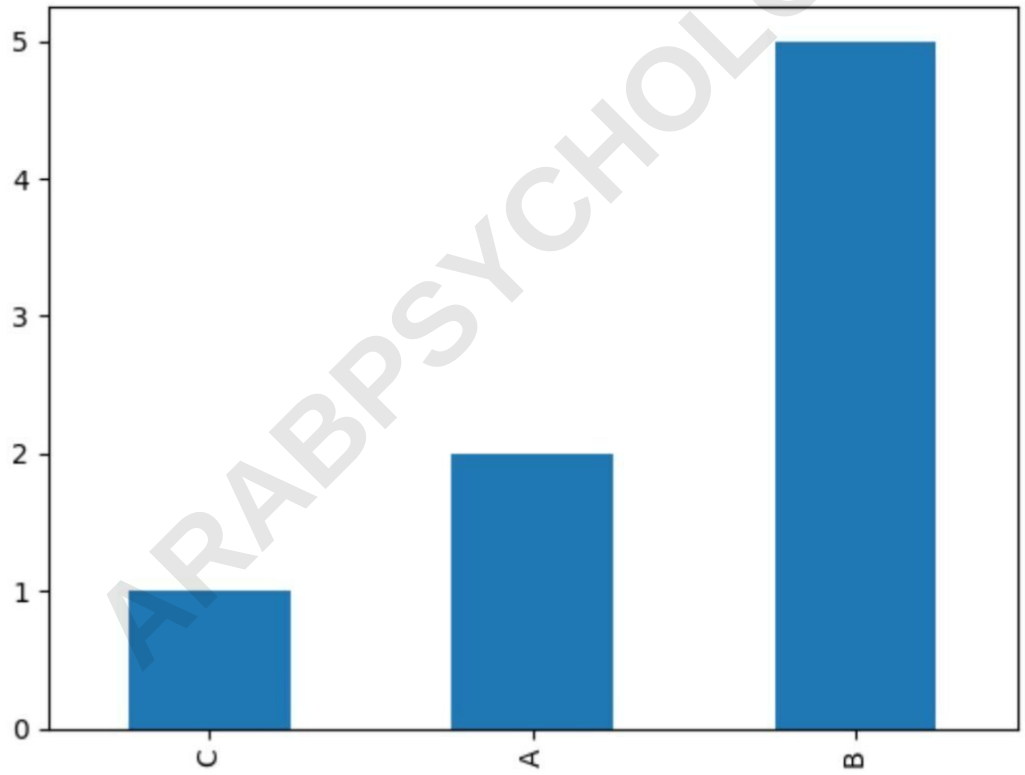
Inverting the Sort Order for Focused Analysis

While descending order is the default and often preferred for dominance analysis, situations

frequently arise where the focus shifts to the least frequent categories, potentially identifying outliers or underrepresented segments. To achieve a visualization sorted in ascending order--from the smallest count to the largest count--we must explicitly intervene in the data chain by introducing the **.sort_values()** method. This method, when applied to the output of .value_counts(), sorts the results by the count values themselves.

The integration of **.sort_values()** is key to this method. Since **.sort_values()** defaults to ascending order, placing it immediately after .value_counts() overrides the inherent descending sort of the frequency method. This modified Series is then passed to the .plot() function, guaranteeing that the resultant visual representation displays the categories sorted by their frequency count from lowest to highest.

#plot value counts of team in descending order
df.team.value_counts().sort_values().plot(kind='bar')



As clearly illustrated by the resulting bar chart, the order of the categories along the x-axis has been inverted compared to the default plot. Team C, which had the lowest frequency (count 1), is now displayed first, followed by Team A (count 2), and finally Team B (count 5). This explicit

manipulation of the sorting order is essential for visualizations intended to emphasize scarcity or minor categories within the data distribution, providing a targeted perspective that the default descending sort cannot offer.

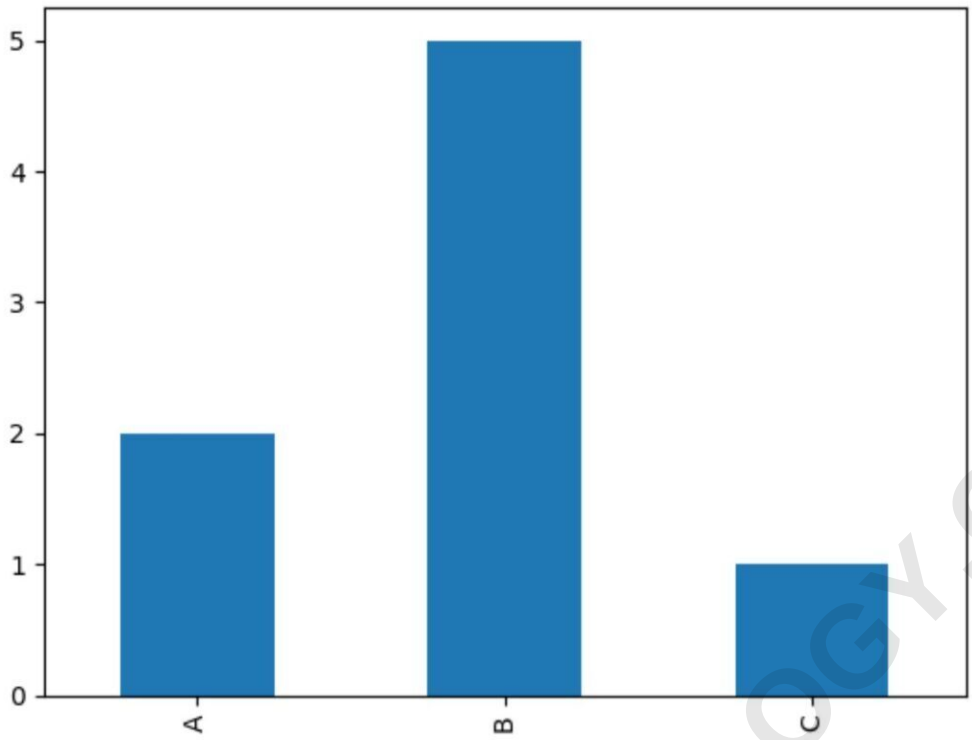
Example 3: Plot Value Counts in Order They Appear in DataFrame

Maintaining Contextual Order: Non-Frequency Based Sorting

In certain analytical contexts, the absolute frequency of values is less important than the temporal, sequential, or geographical order in which those unique values first appear within the original dataset. For instance, if the DataFrame is inherently ordered chronologically, sorting the categories by frequency (descending or ascending) would destroy this contextual relationship. To plot the value counts based solely on the physical appearance order of the unique values in the column, a more sophisticated indexing technique is required.

This method involves two distinct steps. First, we calculate the frequencies using **.value_counts()**. Second, we generate an ordered list of unique categories using the column's **.unique()** method. The key is then using this ordered list derived from **.unique()** to index and select the counts calculated by **.value_counts()**. This forces the resulting Series, which contains the correct frequency counts, to adopt the indexing order defined by the appearance sequence in the original column. This indexed Series is then passed to the .plot() method for visualization.

#plot value counts of team in order they appear in DataFrame
df.team.value_counts().plot(kind='bar')



As demonstrated by the visualization above, the categories are now sorted neither by count nor alphabetically, but strictly based on the order of their first appearance in the source column. When examining the original `df` column, the value 'A' appears first (row 0), followed by 'B' (row 2), and then 'C' (row 7). Thus, this is the precise sequence in which the bars are placed in the bar chart: A, B, C. This technique is invaluable for analyses where maintaining the inherent structure of the data source is more critical than emphasizing frequency magnitude.

Conclusion: Customizing and Extending Value Count Visualizations

The ability to quickly and accurately visualize frequency distributions using the chained **.value_counts()** and **.plot()** methods is a cornerstone of effective data analysis in the Pandas ecosystem. We have explored three crucial sorting techniques--descending by default, ascending via **.sort_values()**, and contextual order using **.unique()** indexing--each providing a different analytical lens through which to view the data. Mastering these sorting controls is essential for producing visualizations that not only inform but also adhere to the specific narrative requirements of the analysis.

Beyond sorting, the built-in .plot() wrapper offers extensive customization inherited from Matplotlib. Users can adjust colors, add titles, label axes, change plot types (e.g., from 'bar' to 'pie' or 'line'),

and refine aesthetics to meet publication quality standards. For instance, using the `figsize` argument allows control over the plot dimensions, and parameters like `rot` (rotation) can manage the orientation of long x-axis labels. By understanding the core structure demonstrated here, analysts are well-equipped to transition from basic frequency plots to highly detailed and informative data visualizations.