How to Calculate Mean and Standard Deviation of a DataFrame Without `describe()`

Authored by

stats writer

November 20, 2025

One of the most frequent tasks in Pandas data analysis involves calculating fundamental statistical measures of central tendency and dispersion, such as the arithmetic mean and the standard deviation. While the powerful `describe()` function offers a comprehensive suite of descriptive statistics in a single call, analysts often require only these two specific metrics. Fortunately, the Pandas library provides highly efficient, dedicated methods--`mean()` and `std()`--to achieve this goal directly, or alternatively, advanced indexing can be employed to selectively extract these metrics from the `describe()` output itself.

Understanding which method to choose depends entirely on the context of your workflow. If the primary objective is pure efficiency and minimizing computation time by focusing solely on these two measures, utilizing the dedicated `mean()` and `std()` functions is the recommended path. Conversely, if you are working within a structure where the complete output of `describe()` is necessary for other downstream tasks, filtering that existing statistical summary via sophisticated indexing techniques, such as the loc accessor, provides a convenient way to isolate the desired results without redundant calculations.

The Comprehensive Power of the describe() Function

The `describe()` function is a cornerstone utility in the Pandas ecosystem, designed to quickly generate a summary of descriptive statistics for numerical columns within a DataFrame. When executed without any arguments, it automatically processes all columns containing numeric data types and computes several key measurements that summarize the distribution of the data. This high-level overview is invaluable during the initial phases of exploratory data analysis (EDA), allowing data scientists to rapidly assess the shape, central tendency, and spread of their variables, helping to detect outliers or unusual distributions immediately.

By default, the `describe()` function meticulously calculates and returns the following eight essential metrics for each numeric variable in a DataFrame. This comprehensive output is structured as a new DataFrame, where the calculated metrics occupy the index rows and the variables form the columns. This standardized output format facilitates easy integration into reports or further statistical modeling, although it often provides more information than necessary when only the mean and standard deviation are required.

count (number of non-null values present in the column)
mean (the arithmetic average value)
std (the standard deviation, measuring dispersion)
min (the minimum observed value)
25% (the 25th percentile, or first quartile)
50% (the 50th percentile, or median)
75% (the 75th percentile, or third quartile)

max (the maximum observed value)

While the utility of the full descriptive summary is clear, generating all eight metrics can sometimes be computationally inefficient, particularly when dealing with massive datasets where the calculation of percentiles might introduce unnecessary overhead. This is precisely why specific, targeted functions like `mean()` and `std()` exist, offering a streamlined approach to calculating only the required statistical properties, bypassing the need for generating the full `describe()` output and subsequent filtering.

Isolating Mean and Standard Deviation by Filtering describe()

Despite the existence of direct methods, a common technique in Pandas involves using `describe()` and subsequently filtering its results to obtain only the mean and <a href="https://en.wikipedia.org/wiki/Standard_deviation. This approach is highly effective because the output of `describe()` is a standard DataFrame, which can be easily queried using the powerful loc accessor. The key insight here is that the statistical measures (like 'mean' and 'std') are represented as labels in the index of the resultant DataFrame.

To specifically retrieve only the mean and standard deviation rows, we chain the loc accessor onto the result of the `describe()` call. The `loc` method allows selection based on labels, and by passing a Python list containing the labels `'mean'` and `'std'`, we instruct Pandas to return a new DataFrame containing only those selected rows. This provides a clean, two-row summary containing exactly the metrics requested across all numeric columns.

The syntax for this selective extraction is remarkably concise, demonstrating the flexibility and power of chained operations in Pandas. This method is particularly useful when the analyst prefers the highly readable, structured, and transposed format that `describe()` naturally produces, which groups the statistics as row headers for easy comparison across variables. This approach is often utilized when transitioning from a full EDA summary to a focused final report visualization.

df.describe().loc]

The following detailed example demonstrates how to use this syntax effectively in a practical scenario involving athlete performance data, illustrating how to move from a comprehensive statistical summary to a highly focused output containing only the mean and standard deviation.

Example: Using describe() and loc to Filter Results

Let us begin by establishing a sample DataFrame containing simulated performance statistics for several basketball players. This step ensures we have structured data upon which to perform our descriptive statistics calculations. The DataFrame includes both categorical data (`team`) and

several numeric variables (`points`, `assists`, `rebounds`) that will be analyzed.

import pandas as pd

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

#view DataFrame
print(df)

team points assists rebounds
0 A 18 5 11
1 B 22 7 8
2 C 19 7 10
3 D 14 9 6
4 E 14 12 6
5 F 11 9 5
6 G 20 9 9
7 H 28 4 12

If we apply the `describe()` function directly to this DataFrame, Pandas automatically identifies the numeric columns (`points`, `assists`, `rebounds`) and computes the complete set of eight descriptive metrics for each. Notice how the output is generated, including the count, minimum, maximum, and all quartiles, alongside the mean and standard deviation.

#calculate descriptive statistics for each numeric variable
df.describe()

points assists rebounds
count 8.000000 8.00000 8.000000
mean 18.250000 7.75000 8.375000
std 5.365232 2.54951 2.559994
min 11.000000 4.00000 5.000000
25% 14.000000 6.50000 6.000000
50% 18.500000 8.00000 8.500000
75% 20.500000 9.00000 10.250000
max 28.000000 12.00000 12.000000

To narrow down this extensive output to only the central tendency and dispersion measures, we utilize the elegant filtering mechanism provided by the loc accessor. We specifically target the index labels corresponding to the mean and standard deviation, effectively slicing the resulting statistical DataFrame to include only the necessary rows. The resulting output is significantly cleaner and focuses immediately on the required data metrics for interpretation.

#only calculate mean and standard deviation of each numeric variable
df.describe().loc]

points assists rebounds
mean 18.250000 7.75000 8.375000
std 5.365232 2.54951 2.559994

Observe carefully that the output successfully isolates the mean and <a href="https://en.wikipedia.org/wiki/Standard_deviation for each numeric variable. This process confirms that while the `describe()` function internally computed all eight descriptive statistics, the subsequent use of the loc accessor allowed us to select only the rows labeled `mean` and `std`, efficiently presenting the focused summary required for the analysis.

The Direct Approach: Using mean() and std()

For users who wish to strictly adhere to the prompt of calculating mean and standard deviation without relying on the intermediate generation of the full `describe()` output, Pandas offers dedicated, optimized methods: `df.mean()` and `df.std()`. These functions are designed for calculating a single statistic across specified axes, making them much more efficient when only these two metrics are needed, as they bypass the computation overhead associated with generating percentiles and other statistics.

When applied directly to a DataFrame (e.g., `df.mean()`), these methods calculate the statistic column-wise by default (axis=0). The output is not a two-row DataFrame like the filtered `describe()` result, but rather a Pandas Series, where the index consists of the column names and the values are the calculated statistics. This Series format is often preferred for subsequent calculations or quick lookups, and it aligns perfectly with standard Pandas aggregation behavior.

To obtain both the mean and <a href="https://en.wikipedia.org/wiki/Standard_deviation using this direct approach, one simply calls the two methods separately. The results can then be easily combined into a single, cohesive DataFrame or presented side-by-side. This approach guarantees that only the requested calculations are performed, leading to performance improvements, especially when processing enormous volumes of data where optimization is paramount.

Practical Implementation of Direct Methods

Using the same basketball player DataFrame (`df`) established earlier, we can demonstrate the simplicity and clarity of using the direct statistical methods. We will first calculate the mean across all numeric columns and then calculate the <a href="https://en.wikipedia.org/wiki/Standard_deviation separately.

Calling `df.mean()` calculates the arithmetic average of each column. Since the `team` column is categorical, Pandas intelligently excludes it from the calculation by default, focusing only on the `points`, `assists`, and `rebounds` columns. The result is a Series object, mapping the column name to its calculated mean value.

 # Calculate the mean of each numeric column
df.mean()

points 18.250
assists 7.750
rebounds 8.375
dtype: float64

Similarly, `df.std()` computes the standard deviation, which quantifies the amount of variation or dispersion of a set of data values. A high standard deviation indicates that the data points are spread out over a wider range of values, while a low standard deviation suggests they are clustered closely around the mean. This metric is crucial for assessing data volatility and reliability.

 # Calculate the standard deviation of each numeric column
df.std()

points 5.365232
assists 2.549510
rebounds 2.559994
dtype: float64

By comparing the outputs of `df.mean()` and `df.std()` to the filtered `describe()` output, we observe that the numerical results are identical, confirming the accuracy of both methods. The choice between the direct functions and the filtered `describe()` approach, therefore, primarily rests on formatting preference and computational necessity, where the direct functions offer superior performance optimization.

Combining Results and Specifying Axis

While the direct methods return two separate Series objects, it is often desirable to combine them into a single, easily readable structure, similar to the output produced by the filtered `describe()` method. This can be achieved using Pandas' concatenation functions or by creating a new DataFrame from a dictionary of the results.

Furthermore, it is important to note the versatility of these statistical methods regarding the axis of operation. By default, Pandas aggregates along `axis=0` (column-wise), meaning the function is applied down the rows. However, if the requirement is to calculate the mean or standard deviation across the columns (row-wise statistics), one simply needs to pass the argument `axis=1` to the function call (e.g., `df.mean(axis=1)`). This flexibility allows analysts to quickly compute summary statistics for individual records or rows, which is invaluable in specific time-series or observational analyses.

 # Combining mean and std results into a single DataFrame
summary_df = pd.DataFrame({'mean': df.mean(),
'std': df.std()})

# Displaying the combined summary, transposed for clarity
print(summary_df.T)

points assists rebounds
mean 18.25000 7.75000 8.375000
std 5.36523 2.54951 2.559994

This combined output provides the exact structure and content as the filtered `describe()` method, but achieves it using only the minimal necessary calculations. Understanding the nuances between the direct methods and the filtered approach allows expert users to write more readable, maintainable, and highly optimized data analysis code, ultimately improving the overall efficiency of large-scale data processing workflows.