How to Calculate Custom Percentiles with the describe() Function

Authored by

stats writer

November 20, 2025

The describe() function is one of the most powerful initial tools available in the pandas DataFrame toolkit, designed specifically to generate comprehensive summary statistics for numerical data columns. This function quickly provides vital metrics such as count, mean, standard deviation, minimum and maximum values, and, crucially, specific percentiles. While it defaults to calculating the standard quartiles (25th, 50th, and 75th percentiles), its real flexibility lies in the ability to customize these boundary markers using the optional **percentiles** parameter. By supplying this parameter with a sequence--such as a list or array of decimal values representing the desired cutoff points (e.g.,)--data analysts can tailor the statistical output precisely to their domain-specific requirements, moving beyond the standard interquartile range analysis.

Understanding how to leverage the **percentiles** argument is essential for advanced exploratory data analysis (EDA). The default behavior is suitable for general data checks, but when investigating specific distribution characteristics--such as the concentration of values at the extremes or defining custom performance tiers--the ability to specify exact percentiles becomes invaluable. This customization allows for rapid assessment of data dispersion and skewness relative to user-defined thresholds, providing richer context than standard summary metrics alone. Therefore, mastering the manipulation of the **describe()** function's percentile calculation is a key skill for efficient data manipulation and reporting within the Python ecosystem.

Understanding the Role of the describe() Function

The **describe()** function is fundamental for obtaining a quick, quantitative overview of the data contained within a pandas DataFrame. When applied to a DataFrame, it automatically targets all columns with numerical data types and computes a series of descriptive statistics. This provides immediate insights into the central tendency, variability, and shape of the dataset's distributions. Metrics such as the mean and standard deviation are calculated to understand typical values and spread, while the minimum and maximum define the data range.

The inclusion of percentiles in the standard output is crucial, as these values divide the data into specific segments, offering a non-parametric measure of data distribution. Unlike the mean, which can be heavily skewed by outliers, **percentiles** provide robust measures of where observations fall within the dataset. By default, pandas calculates the 25th, 50th, and 75th percentiles, which are also known as the first quartile (Q1), the median (Q2), and the third quartile (Q3), respectively. These three values define the interquartile range (IQR), a common measure of statistical dispersion.

However, the utility of **describe()** extends beyond these default quartiles. By using the **percentiles** argument, data scientists can specify any arbitrary set of percentiles they wish to include in the output. This is particularly useful when analyzing distributions that require finer granularity at specific points, such as identifying the 95th percentile for extreme values or the 10th percentile for

low-end performance metrics. This allows the function to become a tailored tool for deep distribution analysis, offering control over which statistical markers are most relevant to the investigation.

Setting Up the Example Dataset in Pandas

To demonstrate the versatility of the describe() function and its **percentiles** parameter, we will first construct a simple pandas DataFrame. This DataFrame models performance data for eight hypothetical teams, recording their points, assists, and rebounds. This small dataset provides a clear, manageable context to observe how percentile calculations change based on the inputs provided to the function.

The initialization process involves importing the pandas library and then defining the data using a dictionary structure, which is subsequently converted into a DataFrame object. This object, named **df**, serves as the foundation for all subsequent statistical calculations. Observing the raw DataFrame helps in manually verifying the descriptive statistics generated by the **describe()** function later on, especially for small counts where the data points are readily visible.

The following code snippet shows the creation and viewing of the DataFrame:

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

Example 1: Utilizing the Default describe() Output

Our first step is to apply the **describe()** function without providing any optional parameters. This demonstrates the function's default behavior, which is to calculate standard summary statistics, including the default quartiles. This provides a baseline understanding of the data's distribution across the numerical columns (**points**, **assists**, and **rebounds**).

The output will contain eight standard metrics: **count**, **mean**, **std** (standard deviation), **min**, the 25th percentile (Q1), the 50th percentile (Q2 or median), the 75th percentile (Q3), and **max**. The 25% and 75% values are particularly useful as they define the boundaries of the middle 50% of the data, helping to identify potential outliers beyond this range.

The following code shows how to use the **describe()** function to calculate descriptive statistics for each numeric variable in the DataFrame without any percentile customization:

#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

As shown in the output, the **describe()** function successfully calculated the 25th, 50th, and 75th percentiles for each variable by default. For instance, in the **points** column, 75% of the teams scored 20.5 points or less, while 25% scored 14 points or less.

Example 2: Specifying Custom Percentiles for Detailed Analysis

When the standard quartiles are insufficient for a specific analysis, we can utilize the **percentiles** argument. This argument accepts a list or array of floating-point numbers between 0 and 1, where 0.3 represents the 30th percentile, 0.6 the 60th percentile, and so forth. This allows for highly targeted analysis of data distribution. For instance, in performance-based datasets, we might be interested in the 90th percentile to identify top performers accurately, or the 30th percentile to establish a low-performance benchmark.

For this example, we will calculate the 30th, 60th, and 90th percentiles for the numerical variables. This demonstrates how to override the default percentile calculations while preserving the other key summary statistics provided by the describe() function. Note that the function expects the percentile values as fractions (decimals).

The following code shows how to use the **describe()** function with the **percentiles** argument set to , calculating the 30th, 60th, and 90th percentiles for each numeric variable in the DataFrame:

#calculate custom percentiles for each numeric variable
df.describe(percentiles=)

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
30% 14.400000 7.00000 6.200000
50% 18.500000 8.00000 8.500000
60% 19.200000 9.00000 9.200000
90% 23.800000 9.90000 11.300000
max 28.000000 12.00000 12.000000

Interpretation of Custom Percentile Results

The output clearly demonstrates that the **describe()** function successfully returns the custom 30th, 60th, and 90th percentiles alongside the standard metrics. Examining the **points** column, we can now state that 30% of the teams scored 14.4 points or fewer, while the top 10% (above the 90th percentile) scored more than 23.8 points. This level of detail offers a much sharper perspective on the team distribution compared to relying solely on the 25th and 75th percentiles.

It is important to notice a key feature in the output: regardless of the custom percentiles requested, the 50th percentile is always included. The **describe()** function treats the 50th percentile as the median value, which is considered a core measure of central tendency and is therefore calculated as a default metric separate from the list specified in the **percentiles** argument. This ensures that the robust measure of the middle data point is always present, maintaining the completeness of the statistical summary.

Furthermore, the ability to specify custom percentiles is critical when dealing with non-normally distributed data or when focusing on specialized benchmarks. For instance, in financial datasets, one might use the 5th and 95th percentiles to define Value at Risk (VaR), which measures

potential losses at extreme ends of the distribution. By customizing the output of the describe() function, analysts save time by avoiding manual calculation of these specialized statistics.

Example 3: Suppressing Percentile Calculations

There may be instances where an analyst requires only the basic statistical measures (count, mean, standard deviation, min, max) and wishes to exclude all percentile calculations, including the default 25th and 75th quartiles. While the 50th percentile (the median) is typically retained, suppressing other percentiles can streamline the summary statistics output, making it cleaner for reports focused purely on standard deviation and average performance.

To achieve this suppression, we pass an empty list to the **percentiles** argument: **percentiles=**. This informs the **describe()** function to calculate the mandatory descriptive measures but ignore any requests for quartile or custom percentile calculations beyond the hardcoded inclusion of the median. This technique is useful when performing preliminary data checks where detailed distribution markers are not yet needed, offering a simplified statistical snapshot of the data.

The following code shows how to use the **describe()** function with the argument **percentiles=** to calculate no custom or default quartiles for each numeric variable in the DataFrame:

#calculate no percentiles for each numeric variable
df.describe(percentiles=)

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
50% 18.500000 8.00000 8.500000
max 28.000000 12.00000 12.000000

The Significance of the 50th Percentile (Median)

As evident in the output of Example 3, even when the **percentiles** argument is set to an empty list, the 50th percentile remains present in the output. This consistent inclusion highlights the unique status of the 50th percentile within descriptive statistics. It represents the median, which is the value separating the higher half from the lower half of a data sample.

The median is a robust measure of central tendency because it is not unduly influenced by extreme outliers, unlike the mean. For instance, if the **points** column contained an outlier score of 100, the mean would shift dramatically, but the median would remain relatively stable. The

describe() function's hardcoded calculation of the 50th percentile ensures that this critical measure of center is always available, providing immediate insight into the true middle point of the data distribution, regardless of skewness.

Therefore, while the **percentiles** argument offers exceptional control over boundary definitions (Q1, Q3, custom deciles, etc.), the 50th percentile is treated as an independent core metric, fundamental to the concept of comprehensive summary statistics provided by the **describe()** function. This design choice guarantees that analysts always have access to a reliable, outlier-resistant measure of central tendency when exploring their data.