Learning Pandas: Mastering Descriptive Statistics with the `describe()` Function

Authored by

Mohammed looti

November 15, 2025

The Importance of Clear Descriptive Statistics in Data Analysis

In the realm of data science and analysis, the initial step often involves gaining a rapid understanding of the dataset's composition and underlying structure. This process relies heavily on Descriptive Statistics--measures that summarize features of a collection of information. The Python ecosystem, championed by the robust **Pandas** library, provides the indispensable tool for this task: the **describe()** function. When applied to a **DataFrame**, this single command instantly computes essential metrics, including count, mean, standard deviation, and quartile ranges, offering immediate, actionable insights into the numerical columns.

While the utility of **describe()** is undeniable, its default output format can sometimes present a challenge. Specifically, when dealing with columns containing very large or very small magnitudes, **Pandas** intelligently switches to **scientific notation** (e.g., 1.23e+06). Although mathematically precise and efficient for computational display, this format introduces an unnecessary cognitive load for human interpretation, especially when presenting results to non-technical stakeholders or when exact dollar amounts or inventory counts are required.

The goal of effective data communication is clarity. This article addresses this friction point by detailing practical, concise methods within the **Pandas** framework to globally or selectively suppress **scientific notation**. Mastering these techniques ensures that your statistical summaries are not only accurate but also immediately accessible and highly professional, thereby significantly enhancing the overall quality and impact of your data reporting.

Deconstructing Scientific Notation and the Need for Fixed-Point Formatting

To understand why we need to format the output, it is important to first define **scientific notation**. This standardized way of writing numbers is used primarily by scientists and engineers to handle numbers that are either too extensive or too minute to be practically written in standard decimal form. It typically involves expressing the value as a mantissa (a number between 1 and 10) multiplied by a power of 10. For instance, a value of 5,400,000 might be rendered as 5.4e+06.

In the context of data analysis, particularly when working with financial records, large population datasets, or high-frequency measurements, the automatic adoption of **scientific notation** by **describe()** can introduce ambiguity. When an analyst sees "1.12e+06," they must mentally calculate or convert this to the more intuitive "1,120,000" to fully appreciate the magnitude of the mean or maximum value. This extra step undermines the efficiency that **Pandas** aims to provide in its summaries.

Therefore, the preferred output format for most business and explanatory reporting is the **fixed-point decimal format**. This format displays numbers exactly as they are, with a specific number of digits after the decimal point, regardless of their magnitude. Achieving this shift requires leveraging

Python's powerful string formatting capabilities, specifically by integrating them into the **.apply()** function chain within **describe()**. The following methods demonstrate precisely how to implement this transformation for both individual data Series and entire **DataFrames**.

Method 1: Fixed-Point Formatting for a Specific Series

When your data analysis focuses on a specific variable--perhaps a column representing annual income or transaction volume--you only need to format the output of that single Series after calling **describe()**. This approach is highly targeted and efficient, requiring only a small, elegant modification to the standard function call. The core of this method involves chaining the **.apply()** method directly onto the statistical results.

The **.apply()** function, in this context, allows us to iterate through the resulting statistical measures (count, mean, std, etc.) and apply a custom formatting rule to each one. We use a **lambda** function to pass a standard Python string **format()** specifier. The key specifier here is `'f'`, which stands for fixed-point notation, compelling Python to display the value as a decimal number rather than using exponents.

The resulting syntax is concise and powerful. By placing the formatting logic immediately after the descriptive calculation, we ensure that the conversion from floating-point scientific notation to readable decimal format happens seamlessly before the results are displayed to the user.

df.describe().apply(lambda x: format(x, 'f'))

This snippet demonstrates the effective use of chaining: the output of `.describe()` (a Series) is passed to `.apply()`, which then uses the **lambda** function to enforce the fixed-point `'f'` format on every statistic computed for the target column.

Method 2: Global Formatting Across Multiple Columns

When the requirement is to generate a comprehensive statistical summary for all numerical columns within a **DataFrame**, Method 1 becomes cumbersome, as it would need to be applied column by column. A far more scalable solution involves applying the formatting logic to the resulting summary **DataFrame** itself. This strategy relies on a nested application of the **.apply()** method.

When `df.describe()` is executed without specifying a column, it returns a new **DataFrame** where the rows are the statistical measures (count, mean, etc.) and the columns are the original numerical features ('sales', 'returns', etc.). To format every cell in this resulting table, we use an outer **.apply()**, which iterates through the rows (the statistical measures). For each row, we then

apply an inner **.apply()**, which iterates through the column values (the numbers themselves) and applies the desired string formatting.

Crucially, Method 2 also allows for precise control over decimal placement, using Python's advanced string **.format()** syntax. For example, the format string `'{0:.5f}'` specifies a fixed-point number that must be displayed with exactly five digits after the decimal point. This level of precision control is vital for analytical consistency and reporting standards.

df.describe().apply(lambda x: x.apply('{0:.5f}'.format))

This nested structure efficiently handles the formatting of potentially hundreds of statistical values across numerous columns, ensuring uniform, non-scientific output throughout the summary table. This is the preferred method for generating large, comprehensive, and presentation-ready descriptive statistics.

Setting Up the Demonstration DataFrame

To effectively showcase the necessity and impact of these formatting methods, we must first establish a representative **DataFrame** where values are sufficiently disparate to trigger the default display of **scientific notation**. Our example will simulate a simplified sales dataset, tracking store performance with columns for total sales and associated returns.

The data preparation involves importing **Pandas** and structuring the data such that the numerical columns contain both very large numbers (in the millions) and small numbers (in the thousands). This variability is exactly what causes the standard **describe()** output to default to exponential notation for improved compactness, which we aim to override.

import pandas as pd

#create DataFrame
df = pd.DataFrame({'store': ,
'sales': ,
'returns':})

#view DataFrame
print(df)

store sales returns
0 A 8450550 2212200
1 A 406530 145200
2 A 53000 300

3 A 6000 2500
4 B 2000 700
5 B 4000 600
6 B 5400 800
7 B 6500 1200

The resulting `df` provides the perfect testing ground. Notice the 'sales' column, in particular, exhibits a huge range--from 2,000 to over 8 million. When we calculate the mean or standard deviation of such a widely distributed column, the output is highly likely to be presented in exponential form, confirming the need for our formatting solution.

Execution Example 1: Applying Fixed-Point Format to 'sales'

Let us first observe the default behavior of **describe()** when applied solely to the 'sales' column. This initial step confirms the problem we are attempting to solve.

#calculate descriptive statistics for sales column
df.describe()

count 8.000000e+00
mean 1.116748e+06
std 2.966552e+06
min 2.000000e+03
25% 5.050000e+03
50% 6.250000e+03
75% 1.413825e+05
max 8.450550e+06
Name: sales, dtype: float64

As anticipated, the output is cluttered with the 'e+' exponent notation, making it difficult to quickly ascertain, for instance, that the mean sales figure is roughly 1.1 million. This lack of immediate clarity necessitates the application of Method 1.

By integrating the **.apply()** method with the `lambda` function and the fixed-point specifier `'f'`, we force **Pandas** to render these results in a traditional decimal format, vastly improving readability.

#calculate descriptive statistics for sales column and suppress scientific notation
df.describe().apply(lambda x: format(x, 'f'))

count 8.000000

mean 1116747.500000
std 2966551.594104
min 2000.000000
25% 5050.000000
50% 6250.000000
75% 141382.500000
max 8450550.000000
Name: sales, dtype: object

The transformation is immediate and effective. The statistical measures are now presented as clear, unambiguous decimal numbers. This single-column method is perfect for quick inspections and verifying the distribution characteristics of critical variables.

Execution Example 2: Global Formatting for 'sales' and 'returns'

Next, we turn our attention to analyzing the entire set of numerical columns--'sales' and 'returns'--using `df.describe()`. Observing the default output confirms that the descriptive statistics for both columns suffer from the same readability issue.

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

sales returns
count 8.000000e+00 8.000000e+00
mean 1.116748e+06 2.954375e+05
std 2.966552e+06 7.761309e+05
min 2.000000e+03 3.000000e+02
25% 5.050000e+03 6.750000e+02
50% 6.250000e+03 1.000000e+03
75% 1.413825e+05 3.817500e+04
max 8.450550e+06 2.212200e+06

The table above, while statistically complete, presents a visual challenge. The mean of 'returns,' for example, requires translation from 2.954375e+05 to 295,437.5. To rectify this across the entire summary table, we implement Method 2, utilizing the nested **.apply()** structure with precise decimal control.

This method ensures that every value in the resulting summary **DataFrame** adheres to the specified fixed-point format, offering a unified, clean presentation of the descriptive statistics for all relevant columns.

#calculate descriptive statistics for numeric columns and suppress scientific notation
df.describe().apply(lambda x: x.apply('{0:.5f}'.format))

sales returns
count 8.00000 8.00000
mean 1116747.50000 295437.50000
std 2966551.59410 776130.93692
min 2000.00000 300.00000
25% 5050.00000 675.00000
50% 6250.00000 1000.00000
75% 141382.50000 38175.00000
max 8450550.00000 2212200.00000

The final result is a professional-grade summary table where all measures are instantly recognizable and interpretable, eliminating the distraction caused by exponential notation and maintaining the integrity of the calculated figures.

Customizing Precision and Formatting for Reporting

A significant benefit of using Python's string formatting capabilities, particularly the **.format()** method implemented via the **lambda** function in Method 2, is the granular control over decimal precision. In our examples, we used the format specifier `'{0:.5f}'`, which dictates that the output must be a fixed-point number showing exactly five digits after the decimal point.

Analysts can easily tailor this precision based on the nature of the data and the requirements of the report. For financial data, which often requires currency precision, one might use `'{0:.2f}'` to display only two decimal places. Conversely, when working with scientific measurements where high precision is critical, this number can be increased to `.8f` or more. Furthermore, if the data involves integers or whole units (like counts or inventory items), using `'{0:.0f}'` will effectively round the output to the nearest whole number, presenting an even cleaner summary without any decimal points.

This flexibility ensures that the statistical summary remains perfectly aligned with presentation needs, preventing the display of irrelevant decimal noise while still maintaining accuracy and readability. Customizing the format string is a crucial skill for transforming raw computational output into polished, effective communication tools.

Conclusion: Enhancing Data Communication with Pandas Formatting

The ability to effectively present data is just as vital as the accuracy of the underlying analysis. By

incorporating the straightforward formatting techniques outlined here--leveraging the power of **.apply()** and Python's fixed-point format specifiers--you can reliably suppress **scientific notation** in your **describe()** output. Whether analyzing a single data Series or generating a comprehensive summary for an entire **DataFrame**, these methods ensure that your statistical reports are clean, intuitive, and readily understandable by any audience.

Adopting these practices elevates the quality of your data visualizations and reporting, moving beyond default computational outputs to deliver clear, professional analytical summaries. Continuous exploration of **Pandas**' advanced features will further refine your skills in data manipulation and presentation.

Additional Resources for Data Mastery

To delve deeper into advanced data manipulation and analysis with Pandas, consider exploring the following tutorials and documentation:

Official Pandas Documentation
Pandas User Guide: Descriptive Statistics
Python's Built-in format() Function