How to Use Pandas `describe()` and Suppress Scientific Notation

Authored by

stats writer

November 20, 2025

Mastering Pandas describe() and Display Formatting

The Pandas library is an indispensable tool for data manipulation and analysis in Python, particularly renowned for its efficient handling of structured datasets. One of its most frequently used functions is `describe()`, which offers a rapid summary of the underlying data distribution. This powerful method automatically generates descriptive statistics such as count, mean, standard deviation, and quartiles, providing immediate insight into the central tendency, dispersion, and shape of a dataset. However, when dealing with very large or very small numerical values, Pandas often defaults to using scientific notation (e.g., $1.11e+06$), which, while mathematically precise, can significantly hinder readability for human analysts trying to quickly interpret raw business metrics.

Analysts frequently require that these critical summaries be presented in standard decimal format to ensure clarity and professional reporting standards. This need necessitates specific methods for suppressing the default scientific display format utilized by Pandas, particularly when generating the output of the `describe()` function. While global settings exist to manage display options across an entire session, tailored approaches are sometimes necessary when applying transformations specifically within the context of the descriptive output, especially when mixing data types or focusing on specific columns. Understanding these formatting techniques is essential for any professional working extensively with numerical data summaries in a data analysis workflow.

This guide delves deeply into not only how to effectively utilize the describe() function but, more importantly, provides robust coding solutions for controlling the numerical presentation, ensuring that all statistical outputs are both accurate and easily digestible. We will explore both session-wide formatting techniques using the configuration options and highly targeted methods utilizing the `apply()` function directly on the statistical output, allowing for maximum control and flexibility over the final presentation of results derived from a DataFrame.

Understanding the Power of describe()

The `describe()` function is a cornerstone utility in the Pandas toolkit, designed to provide a quick and comprehensive overview of numerical data within a DataFrame. When called on a DataFrame or a specific Series (column), it computes a set of basic descriptive statistics that summarize the key characteristics of the data distribution. For numerical columns, this typically includes the total count of non-null values, the arithmetic mean, the standard deviation (measuring dispersion), the minimum and maximum values, and the interquartile range defined by the 25th, 50th (median), and 75th percentiles. These statistics are fundamental for initial data exploration (EDA), helping data scientists quickly spot outliers, understand data spread, and check for potential skewness or errors before proceeding to more complex modeling.

The versatility of `describe()` extends beyond mere numerical data; when applied to object (string)

or categorical data types, it provides an entirely different, yet equally useful, set of statistics. For non-numeric data, it reports the count, the number of unique entries, the most frequent value (top), and the frequency of that top value. This adaptability makes `describe()` a mandatory first step in data profiling, allowing users to rapidly assess the quality and structure of all variables within their dataset, regardless of their intrinsic type. Proper interpretation of these metrics can guide decisions on data cleaning, transformation, and feature engineering necessary for subsequent analytical tasks.

However, the output format often presents challenges, particularly when the scale of the data spans several orders of magnitude. When numbers are exceptionally large (e.g., millions or billions) or very small (approaching zero), Pandas defaults to scientific notation. This default behavior is optimized for computational efficiency and ensures that large numbers are displayed compactly in the console. While this is technically correct, it introduces a layer of abstraction that requires mental translation, slowing down direct human reading and comprehension. The goal of using `describe()` is often immediate transparency, which is undermined when the output is dominated by exponents like 'e+06' or 'e-03'.

The Challenge of Scientific Notation in Data Analysis

Scientific notation is a standardized way of writing down numbers that are too large or too small to be conveniently written in standard decimal form. It utilizes powers of 10, significantly simplifying the representation of magnitude. In programming environments like Python, especially within the Pandas ecosystem, this notation is the default method for displaying floating-point numbers when they exceed a certain threshold of digits or precision, thereby preventing excessive console width and ensuring consistency in internal data representation.

Despite its mathematical efficiency, scientific notation often conflicts with the practical needs of business reporting and managerial review. When presenting key performance indicators (KPIs) or summary statistics on sales figures, asset values, or patient counts, stakeholders generally prefer easily recognizable, concrete numbers. A mean value displayed as 1.116748e+06 is less intuitive than 1,116,748.00. The need to suppress this formatting is therefore driven primarily by the requirement for clarity, accessibility, and precision control in client-facing or internal data deliverables.

Addressing this challenge requires targeted control over how Pandas renders floating-point data. While Python itself offers numerous ways to format strings and floats, applying these formatting rules specifically to the output of `describe()`--which returns a Series or a DataFrame--requires using methods capable of iterating over and transforming these results. The common methods involve either setting a global configuration for the entire Pandas session or applying a dedicated formatting function, such as `apply()` combined with a lambda function, directly to the resulting

statistical table. This targeted approach ensures that only the descriptive output is modified, leaving other computational steps unaffected, which is often the most flexible and robust solution for production code.

Global Suppression using set_option()

Before diving into method-specific formatting, it is beneficial to understand how to apply global display settings using the set_option() function. This approach is highly effective if you wish to suppress scientific notation for all floating-point numbers displayed throughout your current Pandas session, not just those generated by `describe()`. By setting the appropriate option, you instruct Pandas to use a specific string format for all floating-point representations, thereby eliminating exponents across the board.

To achieve this, you modify the `display.float_format` option within `pd.options`. This option accepts a formatting string, usually utilizing Python's format specification mini-language. For instance, setting it to `'{:.2f}'` will display all floats with exactly two decimal places, effectively suppressing scientific notation unless the number is astronomically large or small and the specified precision is insufficient. This global change is quick, requires minimal code, and impacts every float output, which is ideal for interactive analysis where readability is paramount.

While globally suppressing scientific notation using set_option() provides broad coverage and simplicity, it also comes with potential downsides. Firstly, it changes the display behavior for the entire session; if you are running code that relies on default scientific notation or requires variable precision levels for different types of data, this universal setting might introduce inconsistencies. Secondly, setting a fixed precision (e.g., two decimal places) might obscure subtle but important differences in data where higher precision is actually required. Therefore, for production code or notebooks where precise control is needed, often the column-specific `apply()` method is preferred over the global option, offering a surgical approach to formatting the output of `describe()`.

Method 1: Suppressing Notation for a Single DataFrame Column

When dealing with a statistical summary for a single column (a Pandas Series), a highly effective and targeted method for suppressing scientific notation involves chaining the `describe()` function with the `apply()` method. This allows us to apply a custom formatting function to every statistical value generated by `describe()`. Since `describe()` returns a Series in this case, the `apply()` method iterates through each statistic (count, mean, std, etc.) and transforms it into the desired string format.

The core of this method lies in using a `lambda` function combined with Python's built-in `format()` function. We pass a format string, typically `'f'`, which forces the number to be displayed in fixed-point decimal notation, overriding the exponential notation. This technique ensures that the output

is instantly readable, maintaining the statistical context while prioritizing human comprehension over compact display. This approach is particularly useful when focusing on specific metrics derived from a single variable within a larger DataFrame.

The following syntax illustrates how to apply this method, explicitly defining the output format for a column named 'my_column'. Notice how the use of `format(x, 'f')` is embedded within the lambda function passed to the `apply()` method, ensuring that every resulting statistic is converted into a standard float string representation before being returned.

You can use the **describe()** function to generate descriptive statistics for variables in a pandas DataFrame.

To suppress scientific notation in the output of the **describe()** function, you can use the following methods:

Method 1: Suppress Scientific Notation When Using describe() with One Column

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

Method 2: Handling Multiple Columns with Precision Control

When applying `describe()` to an entire DataFrame (or multiple numerical columns), the function returns a resulting DataFrame where rows represent the statistics (count, mean, std, etc.) and columns represent the variables. Suppressing scientific notation in this multidimensional output requires a slightly more complex application of the apply() method, leveraging its ability to operate across either rows or columns.

To format all numerical columns simultaneously, we apply a formatting function over the columns axis (default, or `axis=0`) of the statistical DataFrame generated by `describe()`. The outer `apply()` iterates through each column of the resulting statistics table (e.g., the 'sales' column, the 'returns' column). For each column (which is a Series of statistics), an inner `apply()` is used to format the individual floating-point numbers. This nested structure ensures that every single statistical value within the summary table is correctly rendered in fixed-point decimal notation.

A key advantage of Method 2 is the ability to easily specify the required decimal precision using Python's format strings, such as `'{0:.5f}'`. In this example, '5f' dictates that the output should show exactly five digits after the decimal point. This precision control is vital for maintaining data integrity and satisfying reporting requirements where exact decimal representation is mandatory. This method provides the maximum flexibility, allowing precise formatting control without resorting to global session changes, ensuring that descriptive summaries across multiple variables are clean, readable, and highly accurate.

Method 2: Suppress Scientific Notation When Using describe() with Multiple Columns

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

Setting Up the Demonstration Data

To effectively demonstrate both techniques for suppressing scientific notation, we first establish a concrete sample DataFrame. This DataFrame simulates typical business data, containing a mix of categorical ('store') and large numerical data ('sales' and 'returns'), which are prone to being displayed in scientific notation due to the large magnitudes involved. By creating this structured dataset, we can clearly illustrate the problem caused by the default display settings and then verify the success of our formatting solutions.

The chosen dataset includes sales figures reaching into the millions and return figures reaching the hundreds of thousands, deliberately setting the stage for Pandas' internal display logic to activate scientific notation when summarizing these variables. Observing the statistical output before and after applying the custom formatting provides the clearest possible understanding of the value added by these suppression techniques. We start by importing the Pandas library and then constructing the DataFrame structure.

The following code block initializes the DataFrame, which will serve as the basis for all subsequent examples. Note the immediate printing of the DataFrame to confirm the data structure and values before statistical computation begins. This ensures transparency in the process and verifies that the initial data setup is correct, a standard practice in robust data science workflows.

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

Example 1: Applying Precision to a Single Column

In this first practical example, we focus exclusively on the 'sales' column, illustrating Method 1 for targeted formatting. Initially, we run the standard `describe()` function on the 'sales' Series. As expected, given the magnitude of the sales figures (up to 8 million), the resulting descriptive statistics are displayed using scientific notation. This default view, while numerically correct, makes metrics like the mean or standard deviation difficult to digest immediately, requiring the reader to convert the exponents back into conventional numbers.

Observe the output below; the mean is displayed as `1.116748e+06`, and the maximum value is `8.450550e+06`. This is the problem we aim 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

The key to transforming this output is applying the `lambda x: format(x, 'f')` function directly to the descriptive statistics Series. By using the fixed-point format specifier 'f', we force Pandas to render the numbers in standard decimal format. This simple chaining drastically improves the readability of the summary.

After applying this transformation using the apply() method, the output below clearly shows the mean value as `1116747.500000` and the maximum as `8450550.000000`. The immediate benefit is the elimination of the mental effort required to decode the exponential representation, yielding an instantly understandable summary suitable for direct reporting. Note that because we used the generic 'f' format, Pandas defaults to a high level of decimal precision, which can be adjusted if necessary (though the subsequent example demonstrates better precision control).

#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

Example 2: Batch Formatting Multiple Numeric Columns

Building on the previous concept, Example 2 addresses the common scenario where a data analyst needs to summarize the entire numerical portion of a DataFrame, which, in our case, includes both 'sales' and 'returns'. Running `df.describe()` without any formatting immediately reveals the same issue of obfuscation through scientific notation, now applied across both columns of the resulting summary DataFrame. This simultaneous presentation makes the entire table difficult to interpret quickly.

The initial output confirms this challenge, showing values like `2.954375e+05` for the mean of 'returns'. This standard default output is often unsuitable for final reporting and necessitates the more advanced application of the nested `apply()` structure introduced in Method 2.

#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

To suppress the notation effectively across all columns, we use the chained apply() structure.

Crucially, we utilize the format string `'{0:.5f}'`. The `.5f` component specifies that all output values should be displayed in fixed-point decimal format, enforced to show exactly five decimal places. This provides a balance between suppressing scientific notation and ensuring a controlled, standardized level of precision across all reported descriptive statistics.

The result below demonstrates the success of this batch formatting. Both the 'sales' and 'returns' summaries are now completely free of exponential notation, and all values adhere to the specified five-decimal-place precision, yielding a professional and easily interpretable summary table. This technique is highly recommended for production reporting when summarizing multiple numerical variables within a single comprehensive table.

#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

Note that in this example we used **0:.5f** to display **5** decimal places in the output.

Feel free to change the **5** to a different number to display a different number of decimal places, allowing for flexible precision tailored to specific data requirements.

Conclusion: Enhancing Data Presentation

The ability to generate clean, readable summaries is paramount in data science, bridging the gap between raw computational output and actionable business intelligence. While the Pandas `describe()` function efficiently delivers fundamental statistical insights, its default reliance on scientific notation for large numbers often compromises data clarity. By mastering the techniques outlined--specifically, the use of `apply()` with custom formatting strings--analysts gain granular control over numerical presentation.

We have demonstrated two reliable methods: a single-column approach using `format(x, 'f')` and a robust multi-column approach utilizing nested `apply()` calls with precise format strings like `'{0:.5f}'`. These methods are superior to relying solely on global `set_option()` configurations, as

they offer context-specific control and maintain session neutrality, ensuring that only the desired output is formatted according to specific reporting standards.

Integrating these formatting techniques into a standard data analysis workflow ensures that statistical summaries are not only mathematically accurate but also instantly interpretable by all stakeholders. Prioritizing readability through the suppression of scientific notation elevates the quality of data products and facilitates smoother decision-making processes based on clear, transparent descriptive statistics.