Use Pandas head() Function (With Examples)

Authored by

Mohammed looti

November 3, 2025

The world of modern data analysis relies heavily on efficient tools for processing and inspecting large datasets. Within the ecosystem of Python, the Pandas library stands out as the fundamental utility for data manipulation. A crucial, yet often underestimated, step in any data science workflow is the initial exploration of the dataset to verify its structure and content. This initial glance is typically performed using the **head() function**, a simple but powerful method that provides immediate insights into the data's integrity.

The **head() function** is specifically designed to grant developers and analysts an efficient window into the beginning of a dataset. It is applicable to both Pandas DataFrame and Series objects, allowing users to efficiently retrieve and display the top portion--specifically, the first *n* rows--of the structure. Utilizing this function is paramount for conducting preliminary sanity checks, confirming that data has been loaded correctly from its source, and grasping the general format and indexing of potentially massive datasets without incurring the performance cost of rendering the entire structure.

Mastering the basic syntax of the head() function is exceptionally straightforward, making it an essential tool for beginners and experts alike. When called directly on a DataFrame object (typically named `df`), the function operates without needing any explicit arguments to return a default number of rows, usually five. Understanding this simple structure is the first step toward effective data exploration.

The foundational syntax for calling this method is demonstrated below:

df.head()

To provide practical, reproducible illustrations throughout this guide, we will employ a basic sample DataFrame. This synthetic dataset is designed to simulate simple sports statistics, tracking key performance indicators such as points scored, assists provided, and rebounds secured across multiple observations. This context allows us to observe how `head()` behaves in real-world scenarios.

import pandas as pd

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

#view DataFrame
df

points assists rebounds
0 25 5 11
1 12 7 8
2 15 7 10
3 14 9 6
4 19 12 6
5 23 9 5
6 25 9 9
7 29 4 12

Example 1: Viewing the Default Number of Rows

One of the most common applications of the **head() function** is calling it without specifying any parameters. This invocation leverages the function's internal default setting, which is hardcoded to display the first five rows of the DataFrame (i.e., where *n* equals 5). This default provides an optimal balance between viewing enough data to confirm structure and keeping the output concise, making it suitable for rapid initial inspection.

This default output is crucial during the initial stages of any data science project. It serves as an immediate confirmation that the data structure is intact: column names are spelled correctly, data types appear as expected in the first few rows, and the index values are correctly established. If any unexpected values, missing data patterns, or corrupted indices appear within these first five rows, the analyst can halt further processing and address the data loading issue immediately, saving significant time downstream.

To execute this essential default view using our sample data, we simply invoke the method directly on our DataFrame object, `df`. Notice that no arguments are passed, relying entirely on the built-in default behavior of the Pandas implementation.

#view first five rows of DataFrame
df.head()

points assists rebounds
0 25 5 11
1 12 7 8
2 15 7 10
3 14 9 6
4 19 12 6

As observed in the output, the resulting structure cleanly presents the first five indexed

observations (indices 0 through 4) across all available features: `points`, `assists`, and `rebounds`. This confirms the successful loading and correct structural alignment of our dataset.

Example 2: Customizing Output using the n Argument

While the five-row default is excellent for a quick structural check, effective data exploration often demands greater flexibility in the number of records displayed. The **head() function** is designed to accommodate this need through the use of the optional **n argument**. This argument accepts an integer value, giving the user precise control over exactly how many rows are returned, starting from the very top of the dataset.

The ability to customize the **n argument** is highly valuable for tailoring the inspection process. For instance, when dealing with extraordinarily large datasets, setting a small *n* (e.g., `n=2`) minimizes output clutter and speeds up display time, providing a minimal yet sufficient snapshot. Conversely, if an analyst suspects a specific data integrity issue or pattern that only emerges across the first eight or ten entries, setting a larger *n* facilitates a more detailed initial investigation without requiring the user to scroll through hundreds of rows.

To retrieve a specific, non-default number of rows, the desired integer value is explicitly passed within the function call. In the subsequent illustration, we override the default setting and instruct the function to return only the first three rows of our sample statistics DataFrame. This demonstrates focused data slicing for inspection purposes.

#view first three rows of DataFrame
df.head(n=3)

points assists rebounds
0 25 5 11
1 12 7 8
2 15 7 10

The resulting output confirms that the view is precisely clipped after the third observation (index 2). This demonstrates how the explicit use of the `n` parameter ensures a highly focused and manageable snapshot of the data's beginning, which is essential for systematic data quality assessment.

Example 3: Applying head() to a Specific Column (Series)

The utility of the **head() function** extends beyond processing entire DataFrames. It is also an integral method available on individual column objects, which Pandas internally represents as a Series. This functionality is immensely helpful when the analytical focus shifts from the overall

dataset structure to the specific distribution, type, or alignment of data within a single feature.

When applying `head()` to a single column, the methodology involves two steps: first, selecting the specific column using standard bracket notation (e.g., `df`), which returns a Series object, and second, chaining the `.head()` method directly onto that resultant Series. This chain operation effectively isolates the inspection to one dimension of the dataset.

This technique is particularly efficient for tasks involving feature engineering or detailed data cleaning. For instance, checking the head of a numerical column can quickly reveal if the values are correctly parsed as integers or floats, or if unexpected strings or null values are present at the beginning of the feature. Let's inspect the first five values exclusively within the `points` column to understand its composition:

#view first five rows of values in 'points' column
df.head()

0 25
1 12
2 15
3 14
4 19
Name: points, dtype: int64

The output clearly confirms the index alignment and the corresponding values for the first five entries in the selected column. Furthermore, the accompanying metadata, including the Series name (`points`) and its data type (`dtype: int64`), is displayed, providing comprehensive information about the structure of that specific feature.

Example 4: Viewing the First n Rows of Multiple Selected Columns

In many analytical situations, the goal is to focus on the relationships between a small subset of features while still maintaining the fundamental row structure and observation indices of the overall dataset. To achieve this selective inspection, analysts must first create a temporary subset of the DataFrame before applying the **head() function**.

This process utilizes double-bracket notation (e.g., `df]`). The outer brackets signify that we are performing an indexing operation on the DataFrame, while the inner list contains the names of the desired columns. Crucially, this operation returns a new, smaller DataFrame object that retains the original indexing but contains only the specified features. We can then apply `.head()` to this resultant DataFrame subset to view the top rows.

This specialized technique is indispensable for streamlining visualization preparation, model building, or correlation analysis by limiting the visible output only to the variables currently under investigation. By maintaining the context of the observation indices, the analyst ensures that relationships between the selected columns remain consistent with the original data structure. Let's simultaneously view the first five rows of the `points` and `assists` columns:

#view first five rows of values in 'points' and 'assists' columns
df].head()

points assists
0 25 5
1 12 7
2 15 7
3 14 9
4 19 12

The resultant output is a focused DataFrame snippet, effectively demonstrating data slicing. It shows the first five observations, retaining the original indices, but only for the two features selected, confirming the power and versatility of combining column selection techniques with the `head()` method.

Conclusion and Related Data Inspection Methods

The **head() function** serves as the analyst's frontline defense against unexpected data structures and errors, providing a rapid, manageable snapshot necessary for effective data quality assessment. Its simplicity, combined with the flexibility offered by the optional `n` parameter, makes it an indispensable component of the initial data exploration phase in any Pandas workflow.

However, a complete mastery of data inspection requires understanding complementary methods. While `head()` focuses on the beginning of the data, the `tail()` function offers a similar capability for viewing the last *n* rows. This is often crucial for checking data integrity issues such as truncated files or erroneous footer rows. Additionally, the `info()` method provides a vital high-level summary of the DataFrame structure, including column names, non-null counts, and memory usage, offering structural insights that `head()` does not cover.

To ensure a holistic understanding of data structure and content, it is highly recommended to integrate all three methods--`head()`, `tail()`, and `info()`--into your standard initial analysis routine.

The following tutorials explain how to perform other common functions in pandas: