How to Extract the First Row of a Pandas DataFrame in 2 Simple Steps

Authored by

stats writer

December 4, 2025

To get the first row of a pandas DataFrame, the most efficient and common methods involve utilizing positional indexing via the .iloc accessor or using the intuitive .head(1) method. While both achieve the immediate goal of isolating the initial record, understanding their subtle differences--particularly regarding indexing methodology and the resulting object type--is crucial for writing performant and robust data analysis code. The resulting object, in most single-row selection cases, is a pandas Series, which represents a one-dimensional array where the original column names become the index labels.

Introduction to Row Selection in Pandas DataFrames

The pandas library is the cornerstone of data manipulation in Python, largely due to its foundational structure: the DataFrame. A DataFrame is essentially a two-dimensional labeled data structure with columns of potentially different types, analogous to a spreadsheet or SQL table. When performing data exploration, sampling, or quality checks, accessing the very first record often provides necessary context about the dataset's structure and content.

Selecting the first row can be accomplished through several robust mechanisms provided by the pandas API. The choice between these methods often depends on whether you prefer explicit integer location (positional indexing) or methods designed for sampling (like retrieving the top N records). For accessing the absolute first row, which always corresponds to position zero (index 0), two methods stand out due to their clarity and efficiency: using the integer location accessor, .iloc, or employing the dedicated sampling function, .head. While their outcomes may appear identical for simple retrieval, understanding their internal mechanisms is key when dealing with complex or very large datasets.

Throughout this comprehensive guide, we will explore the precise syntax for these methods, demonstrate how to combine row selection with column subsetting, and clarify the important distinction regarding the data type returned by these operations. We will use practical code examples to illustrate these concepts, ensuring you can confidently extract the first record for any data analysis task.

Method 1: Positional Indexing Using .iloc

The most precise way to retrieve the first row is by using the .iloc accessor. The term iloc stands for "integer location," meaning it relies strictly on the zero-based integer position of the rows and columns, irrespective of the actual labels assigned in the DataFrame's index. Since the first row of any DataFrame is always at position zero, passing to the .iloc accessor immediately yields the desired result.

This approach is highly recommended when dealing with DataFrames where the index labels

might be non-sequential, non-unique, or even custom strings. Since .iloc ignores the label index entirely and focuses only on physical location, it guarantees that you retrieve the physically first record loaded into memory. When only a single integer is provided (e.g., .iloc), pandas interprets this as selecting all columns for the specified row position.

You can use the following methods to get the first row of a pandas DataFrame:

Method 1: Get First Row of DataFrame (Using .iloc)

df.iloc

When executing this command, the .iloc accessor returns a one-dimensional object, specifically a pandas Series. In this Series object, the original column names of the DataFrame are preserved as the index labels, and the corresponding values from the first row become the data elements of the Series. This return type is efficient for accessing individual elements of the row quickly, but it's an important distinction if your downstream analysis requires a two-dimensional DataFrame format (in which case, slicing .iloc would be necessary).

Method 2: Selecting the First Row Using the .head() Method

A conceptually simpler method for retrieving the top records in a DataFrame is using the .head() method. While .head() is typically used to display the first five rows (the default behavior), passing the integer 1 as an argument ensures that only the very first row is returned. This method is often favored during initial data inspection because of its self-descriptive name and ease of use.

Crucially, the .head(1) method differs from .iloc primarily in its return type. Since .head() is designed to return a subset of the original DataFrame structure (N rows), .head(1) returns a new DataFrame containing only that single row. This distinction is vital: if you need a two-dimensional object for subsequent DataFrame operations (like merging or joining), .head(1) is the preferred choice, whereas if you intend to access values directly by column name, the pandas Series returned by .iloc might be more convenient.

Both .iloc and .head(1) are highly optimized operations. However, in contexts where micro-optimizations matter, .iloc is sometimes marginally faster as it directly accesses the underlying NumPy array structure to extract the specific position, whereas .head(1) involves the overhead of constructing and returning a new DataFrame object, even if it only contains one row.

Selecting the First Row with Specific Columns

Often, data analysis requires not just the first row, but only specific attributes or columns from that

record. Pandas allows for highly flexible indexing by chaining column selection (using double square brackets]) with row selection methods like .iloc. This technique is extremely powerful for creating focused subsets of data.

When selecting specific columns first, the result is a smaller DataFrame containing only those columns. Applying .iloc to this reduced DataFrame then extracts the first row, based on the original structure. For example, if a DataFrame has ten columns but you are only interested in columns A, B, and C, you would first filter for these columns and then apply the positional indexer. This improves readability and ensures that only the necessary data is processed, which can be beneficial for memory management when dealing with wide datasets.

Method 2: Get First Row of DataFrame for Specific Columns

df].iloc

When implementing this combined selection, it is crucial to remember the order of operations. Column selection df] is performed first, which generates a temporary, smaller DataFrame. The .iloc operation is then applied to this temporary object. The final output, consistent with single-row selection, will be a pandas Series where the index consists of the specified column names (column1 and column2).

Setup: Creating the Sample DataFrame

To fully illustrate the behavior of these row selection methods, we will define a sample DataFrame representing player statistics. This DataFrame contains three columns: points, assists, and rebounds, and has eight records, utilizing the default zero-based integer index.

Defining a consistent sample dataset allows us to verify the output of each selection technique accurately. We initialize the DataFrame using the standard pd.DataFrame() constructor, passing a dictionary where keys are the column labels and values are lists of corresponding data points. This foundational step ensures our subsequent examples are reproducible and easy to follow.

The following examples show how to use each method in practice with the following pandas DataFrame:

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

As displayed above, the first row of our DataFrame, corresponding to index 0, contains the values 25 for points, 5 for assists, and 11 for rebounds. Our objective in the following examples is to accurately extract this specific set of values using the methods discussed previously, demonstrating their practical application within a coding environment.

Example 1: Retrieving the Entire First Row of the DataFrame

In this initial example, we focus on obtaining all data contained within the first row (index 0) of the DataFrame. We use the .iloc method, which is the most direct way to access data by its integer position. This operation effectively slices the DataFrame along the row axis, returning a projection of all column values for that specific position.

Executing df.iloc provides a swift and efficient mechanism to confirm the details of the first record. The result confirms the expected values (25, 5, and 11), correctly labeled by their respective column names. This output format--the pandas Series--is ideal if the next step is performing calculations or accessing individual data points via dictionary-like key lookup (e.g., first_row).

The following code shows how to get the first row of a pandas DataFrame:

#get first row of DataFrame
df.iloc

points 25
assists 5
rebounds 11
Name: 0, dtype: int64

Notice that the values in the first row for each column of the DataFrame are returned. The output clearly indicates the name of the row (0, derived from the positional index), the data type of the underlying elements (int64), and the column names acting as the series index. Understanding that this output is a Series is critical for future operations; if a DataFrame object is required, using df.iloc or df.head(1) would be necessary to maintain the two-dimensional structure.

Example 2: Subset Selection of Columns from the First Row

In more complex analyses, efficiency dictates that we retrieve only the necessary subset of information. This example demonstrates how to combine column selection with row selection to isolate the points and rebounds values exclusively from the first record. This technique involves placing the column selection prior to the positional row extraction.

By first defining the columns of interest, df], we reduce the computational scope immediately. The subsequent application of .iloc ensures that we extract the first positional record from this reduced dataset. This is highly effective for tasks where memory efficiency is paramount, especially when working with wide DataFrames containing hundreds of columns.

The following code shows how to get the values in the first row of the pandas DataFrame for only the **points** and **rebounds** columns:

#get first row of values for points and rebounds columns
df].iloc

points 25
rebounds 11
Name: 0, dtype: int64

Notice that the values in the first row for the **points** and **rebounds** columns are returned. The resulting pandas Series now only contains two elements, corresponding to the specific column indices requested. This method underscores the flexibility of pandas indexing, allowing users to precisely target the desired data subset both vertically (columns) and horizontally (rows) in a single, chained operation. This approach significantly streamlines the data preparation phase of any statistical modeling or reporting task.

Differentiating Between Series and DataFrame Outputs

A crucial consideration when selecting the first row is the resulting data structure. As demonstrated, using .iloc returns a one-dimensional pandas Series, while using .head(1) returns a single-row DataFrame. Understanding this distinction is vital for ensuring compatibility with subsequent functions in your data pipeline.

The Series object is efficient for extracting scalar values or performing vectorized operations across the row's elements. For instance, if you need to calculate the sum of points and rebounds for the first player, working directly with the Series allows for immediate attribute access. Conversely, if the subsequent step involves functions that strictly require a DataFrame input--such as .merge(), .join(), or complex filtering operations--the DataFrame format provided by .head(1) is mandatory.

If you prefer the integer indexing approach (.iloc) but require a DataFrame output, you must use slicing notation, even for a single row: df.iloc. The slice 0:1 tells pandas to start at index 0 and stop before index 1, thus encompassing only the row at position 0, and the slicing operation inherently preserves the two-dimensional DataFrame structure. Choosing the correct structure early on prevents unexpected errors during complex data transformation steps.

Summary of Best Practices for First Row Selection

When determining the best strategy for retrieving the first row of a DataFrame, the decision should be guided by the requirements of the downstream process. Both .iloc and .head(1) are excellent tools, but they serve slightly different purposes in practice.

For scenarios demanding maximum performance and direct access to data attributes, especially when the required output is a pandas Series, the use of .iloc is generally the preferred method. It is lightweight, ignores potential complications arising from a custom index, and provides the raw data efficiently. Conversely, if maintaining the DataFrame structure is non-negotiable, or if you are simply using the command for a quick inspection, .head(1) offers a cleaner, more readable solution that guarantees a two-dimensional output.

Finally, when combining row selection with column filtering, always structure the operation to select columns first, followed by row positioning, such as df].iloc. This practice ensures that the indexing operation is performed on the smallest possible subset of data, optimizing memory utilization and execution speed, making your pandas code both elegant and highly efficient for any scale of data analysis.