How to Easily Remove the First Row in a Pandas DataFrame: 2 Methods

Authored by

stats writer

December 3, 2025

Working with data often requires precise manipulation, and in the world of Python data science, the Pandas DataFrame is the fundamental building block. A common requirement when cleaning, transforming, or preparing data is the need to remove specific rows, especially the very first row. This might be necessary if the first row contains metadata, descriptive headers that were incorrectly parsed, or simply an unwanted entry that should not be included in subsequent analysis. Understanding how to efficiently and cleanly drop the initial row is a core skill for any data analyst utilizing the Pandas library, ensuring data integrity and correct processing.

We will explore two robust and widely used techniques for achieving this specific task within Pandas. The primary methods leverage two powerful functionalities: the specialized drop() method, which allows for deletion based on explicit labels or indices, and the powerful integer-location based indexing method, iloc, which facilitates precise slicing of the underlying Data Structure. Both approaches are efficient and reliable, capable of producing identical end results, but they offer distinct syntactical and philosophical pathways for manipulating data, which makes the choice between them highly situational.

The core challenge lies in targeting the zero-based position of the first row accurately. In Pandas, data manipulation functions often rely heavily on the concept of the index, which acts as the unique identifier for each row. The first row invariably corresponds to the index label at position zero. Whether you choose to explicitly remove the row by its label using drop(), or construct a new DataFrame that simply excludes the first position using iloc, achieving the desired outcome requires a clear understanding of index alignment and slicing mechanics.

Overview of Row Deletion Techniques

When tasked with removing the initial row of a Pandas DataFrame, developers usually turn to one of the following two standard and highly recommended techniques. The first method, using the drop() function, is explicit about the operation--you are telling Pandas to drop a specific row label. The second method, employing iloc slicing, achieves the same result by returning a view or copy of the original data that starts from the second row, thereby implicitly excluding the first. Understanding the subtle differences in memory usage and syntax between these two approaches is crucial for writing optimized and clear code.

You can use one of the following methods to drop the first row in a pandas DataFrame:

Method 1: Use drop

df.drop(index=df.index, axis=0, inplace=True)

Method 2: Use iloc

df = df.iloc

Each method produces the same result, successfully eliminating the row corresponding to the zero-position index.

These two code snippets represent the most idiomatic ways to perform this operation in Pandas. Note that Method 1 modifies the DataFrame in place due to the `inplace=True` argument, permanently altering the original object. In contrast, Method 2 uses assignment (`df = ...`) to overwrite the original DataFrame reference with the sliced result, which is technically a new object derived from the original data, ensuring non-destructive slicing followed by reassignment. Choosing between in-place modification and generating a new DataFrame object is often a matter of performance consideration and adherence to functional programming paradigms.

Preparing the Sample DataFrame

Before demonstrating the technical implementation of these methods, we must establish a consistent DataFrame to work with. This sample dataset allows us to observe the results of the row removal clearly and verify that both the drop() and iloc methods achieve the intended goal of eliminating the first entry. The dataset below represents simple sports statistics, where the default integer index starts at 0.

The following examples show how to use each method in practice with the following pandas DataFrame. We first import the Pandas library and then construct the structured data object using a dictionary input, defining columns for 'team', 'position', 'assists', and 'rebounds'. It is essential to visualize the original DataFrame structure to appreciate how the index labels shift or disappear after the operation.

import pandas as pd

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

#view DataFrame
df

team position assists rebounds

0 A G 5 11
1 A G 7 8
2 A F 7 10
3 A F 9 6
4 B G 12 6
5 B G 9 5
6 B F 9 9
7 B F 4 12

As observed in the output, the row we intend to eliminate is the one labeled '0', which contains the data for Team 'A' with 5 assists and 11 rebounds. This specific row must be targeted either by its label or its integer position, which coincide in this default scenario. The following sections will demonstrate how both the index-based approach and the positional-based approach handle this task effectively.

Method 1: Utilizing the DataFrame.drop() Function

The drop() method is fundamentally designed for removing entries from a DataFrame based on their label--either row labels (index) or column labels. When removing a row, we must explicitly specify the label we wish to delete and indicate that the operation should be performed along the row axis. Since the first row in a newly created DataFrame defaults to the index label 0, we can use the DataFrame's index attribute to dynamically select this label, making the code robust even if the index were not a simple range.

The core of this method involves three crucial parameters. First, the index parameter is assigned the value of the label we want to drop, which we retrieve using df.index, ensuring we capture the label of the first element regardless of what that label actually is (it could be a date, a string, or an integer). Second, the axis parameter must be set to 0. This signifies that the operation should be applied across the rows (the index) rather than the columns (axis=1). Failing to specify axis=0 will result in an error if Pandas attempts to find the label in the column names.

Finally, the argument inplace=True is critically important. By default, Pandas operations that modify data, like drop(), return a new DataFrame object, leaving the original DataFrame untouched. Setting inplace to True instructs Pandas to modify the DataFrame object directly, saving memory by avoiding the creation of a redundant copy. If inplace were set to False (the default), the result of the drop operation would need to be explicitly assigned back to the original variable, such as df = df.drop(...).

Practical Implementation of drop()

The following code demonstrates the application of the drop() function to remove the first row (index 0) of our previously defined DataFrame. Notice how the index labels remaining after the operation remain unchanged (1, 2, 3, etc.), but the row corresponding to label 0 is completely absent, proving the success of the deletion operation.

The following code shows how to use the **drop()** function to drop the first row of the pandas DataFrame:

#drop first row of DataFrame
df.drop(index=df.index, axis=0, inplace=True)

#view updated DataFrame
df

team position assists rebounds
1 A G 7 8
2 A F 7 10
3 A F 9 6
4 B G 12 6
5 B G 9 5
6 B F 9 9
7 B F 4 12

Notice that the first row has been removed from the DataFrame, and the subsequent rows retain their original index labels, starting from 1.

The output confirms that the row indexed '0' is successfully removed. Importantly, the remaining rows retain their original index labels (1 through 7). This behavior is characteristic of the drop() method, which removes the label specified without automatically resetting the remaining index. If a contiguous, zero-based index is required after the deletion, an additional step using df.reset_index(drop=True, inplace=True) would be necessary.

Also note that we must use **inplace=True** for the row to be removed in the original DataFrame, otherwise the modification would only exist in the returned object, which we would then have to assign back to **df**.

Method 2: Slicing with DataFrame.iloc

The second primary method for removing the first row leverages the iloc indexer. Unlike drop(), which works based on index labels, iloc works purely based on the integer position of the data, similar to standard Python list or array slicing. Since the first row is always at integer position 0, we can use slicing notation to select everything from position 1 onwards, effectively excluding the initial entry.

The syntax df.iloc requires careful decomposition. The square brackets contain two dimensions separated by a comma: rows and columns. For the rows dimension (1:), the number 1 signifies the starting integer position--meaning we start at the second row. The colon immediately following the 1 indicates that we continue until the end of the DataFrame. For the columns dimension (:), a single colon means "select all columns." The result is a selection of all data starting from the second row and including all columns.

A crucial distinction here is that iloc slicing always returns a new DataFrame object, never modifying the original in place (the concept of inplace=True does not apply to slicing operations). Therefore, to update our working DataFrame, we must explicitly reassign the result back to the variable: df = df.iloc. This reassignment overwrites the original DataFrame reference with the new, truncated Data Structure, making this approach particularly popular in scenarios where immutability is preferred or required for safe data pipelines.

Practical Implementation of iloc

The following code illustrates how the iloc function, combined with Pythonic slicing notation, achieves the desired row removal. We utilize the reassignment technique to ensure the original DataFrame is updated to reflect the exclusion of the first row (position 0).

The following code shows how to use the **iloc** function to drop the first row of the pandas DataFrame:

#drop first row of DataFrame
df = df.iloc

#view updated DataFrame
df

team position assists rebounds
1 A G 7 8
2 A F 7 10
3 A F 9 6
4 B G 12 6
5 B G 9 5

6 B F 9 9
7 B F 4 12

Notice that the first row has been removed from the DataFrame, and the index labels remain identical to the previous method's result.

Just like with the drop() method, the iloc approach removes the positional data but preserves the existing index labels (1 through 7). If you were starting a new analytical process, retaining the original index might be desirable for traceability. However, if you need a clean, zero-based index for further manipulation (e.g., using numpy array conversion or simple iterative loops), you would again need to follow up with a reset_index call.

Comparing drop() vs. iloc for Row Removal

Although both drop() and iloc successfully remove the first row, they embody different strategies. The drop() function is verbose and explicit about deletion, requiring the user to specify the index label and the axis. This clarity can be beneficial when dealing with DataFrames that have non-sequential or custom indices, such as date-time indices, where positional slicing is less intuitive or reliable. It explicitly signals the intent to delete data based on its label identifier.

Conversely, the iloc slicing method is highly concise and performant, leveraging native Python slicing mechanics. It operates on the integer position, making it straightforward for removing the first N rows, but potentially less clear if the DataFrame has undergone significant prior index manipulation. Since it always returns a new object, it aligns well with functional programming principles, ensuring that the original data is preserved until the explicit reassignment occurs. This separation of concerns can prevent unintended side effects in complex codebases.

For the specific task of removing the very first row (position 0) in a standard DataFrame, iloc is often preferred by experienced Pandas users due to its brevity and speed. However, if you are working with a large dataset and are highly constrained by memory, using drop() with inplace=True might offer marginal performance benefits by avoiding the memory overhead of creating an entirely new DataFrame copy, though modern Pandas implementations often optimize memory usage even during slicing.

Conclusion: Choosing the Right Method

The decision between using drop() or iloc ultimately hinges on context and coding style. If your primary goal is readability and you want to explicitly state that you are deleting the row corresponding to the index label 0, drop(index=df.index, axis=0, inplace=True) is the optimal choice. It provides explicit control over the deletion process and modifies the DataFrame in place,

which can simplify code flow when sequential transformations are needed.

If, however, your priority is conciseness, adherence to slicing conventions, and creating a new DataFrame object without modifying the original (until explicit reassignment), the iloc slicing method is superior. The syntax df = df.iloc is a classic and very powerful approach in data selection and manipulation, especially for handling edge cases involving data boundaries.

Regardless of the method chosen, both techniques showcased provide reliable and efficient ways to clean your data by removing the unwanted initial row. Mastery of these fundamental Pandas operations is key to moving beyond basic data loading and into advanced data cleaning and preparation workflows.