

Learning to Remove the First Row in Pandas DataFrames: A Step-by-Step Guide

Authored by
Mohammed loot

November 1, 2025

RECOMMENDED CITATION

Mohammed loot (2025). *Learning to Remove the First Row in Pandas DataFrames: A Step-by-Step Guide*. PSYCHOLOGICAL STATISTICS. Retrieved from <https://statistics.arabpsychology.com/?p=8063>

Introduction: Mastering Row Deletion in Pandas

In the realm of modern [data analysis](#) and preprocessing, the ability to efficiently manipulate and clean datasets is paramount. One of the most common tasks faced by data scientists and developers using [Python](#) is the targeted removal of rows. This necessity often arises when dealing with header information mistakenly included as data, corrupted records, or extraneous initial entries that skew analysis.

When working specifically with the powerful, two-dimensional structure known as the [Pandas DataFrame](#), developers frequently need to eliminate the very first row. Fortunately, the Pandas library offers highly optimized, readable methods to achieve this goal, providing flexibility depending on whether you need to target a specific label or a positional index.

This guide will explore two primary and robust techniques for dropping the initial row: leveraging the explicit, label-based [drop\(\) function](#), and employing the fast, positional slicing afforded by the [iloc](#) indexer. Understanding both methods is essential, as they operate on fundamentally different principles--label vs. position--which is critical knowledge for advanced data manipulation workflows.

Establishing the Test Environment: Creating the DataFrame

To provide a concrete foundation for demonstrating these row-deletion techniques, we must first initialize a representative [Pandas DataFrame](#). This baseline structure will serve as our test subject, allowing us to clearly observe the results and side effects of both the **drop()** and **iloc** methods.

We will create a simple DataFrame containing simulated sports statistics. By default, Pandas assigns a zero-based integer index, meaning our first row, which is the target for removal, corresponds to the index label `0` and the positional index `0`. The subsequent code block illustrates the creation and display of this initial dataset.

```
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
```

Method 1: Utilizing the `drop()` Function for Label-Based Deletion

The [`drop\(\)` function](#) is the standard Pandas method for removing entries based on their explicit labels, whether those labels are index names or column names. To remove the first row using this function, we must supply the precise label corresponding to that row. While this is often `0`, it is best practice to dynamically retrieve the label to ensure code robustness, particularly if the DataFrame index has been reset or modified.

We achieve this dynamic retrieval using `df.index`, which returns the label of the element located at the positional index `0`. By passing this label to the `drop()` function, we explicitly tell Pandas which row label to target. Furthermore, we must specify key parameters to define the operation: we set `axis=0` to clearly indicate that we are operating on rows (the index axis), as opposed to columns (`axis=1`).

A crucial consideration when using `drop()` is the `inplace` parameter. Setting `inplace=True` ensures that the operation modifies the original DataFrame directly, without requiring manual assignment back to the variable (e.g., `df = df.drop(...)`). This makes the code concise, although it is important to remember that this modification is permanent within the current session.

#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
```

As the output confirms, the row associated with the original index label `0` has been successfully removed, leaving a DataFrame that begins with index `1`.

Method 2: Leveraging `iloc` for Efficient Positional Slicing

The second, and often preferred, technique for removing the first row relies on the `iloc` indexer. The name `iloc` stands for integer-location based indexing, which means it treats the [Pandas DataFrame](#) strictly as a zero-indexed array, ignoring any custom index labels that may have been assigned. This method is exceptionally fast because it leverages optimized array slicing inherent to [Python](#)'s data structures.

To drop the first row using `iloc`, we employ standard Python slicing notation to select everything *except* the element at positional index `0`. The syntax is `df.iloc`. We utilize the slice to achieve the desired result of omitting the first row.

This powerful slicing notation instructs Pandas precisely how to construct the new DataFrame. The components of are interpreted as follows:

The row selection `1:` specifies that the new DataFrame should start at positional index `1` (the second row) and continue until the end. This effectively skips the first row (index `0`).

The column selection `:` indicates that all columns from the original DataFrame should be included in the result.

Unlike the `drop(inplace=True)` operation, `iloc` always returns a new DataFrame copy. Therefore, to make the change permanent, we must explicitly assign the sliced result back to our original variable: `df = df.iloc`. This assignment overwrites the original DataFrame with the modified version.

#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
```

Upon viewing the updated DataFrame, we observe the identical result achieved by the **drop()** method: the initial row of data has been successfully removed via positional indexing.

Comparing drop() vs. iloc Slicing for Performance and Context

While both the [drop\(\) function](#) and [iloc](#) slicing successfully remove the first row, the choice between them should be guided by performance needs, index type, and code clarity. Understanding these differences is crucial for writing efficient and maintainable Pandas code.

The `iloc` slicing approach (`df = df.iloc`) is generally regarded as the most performant method for this specific task. Pandas slicing operations are highly optimized, relying on zero-copy views and low-level numerical array manipulation (thanks to NumPy integration). When dealing with large datasets where execution speed is a priority, `iloc` provides a significant advantage due to its direct manipulation of positional indices.

Conversely, the `drop()` method is intrinsically designed for label management. While it is more explicit--you are telling Pandas to remove the item with label 'X'--it incurs a slight overhead because it must first look up the label, verify its existence, and then proceed with deletion and DataFrame reconstruction. If your DataFrame has a non-standard index (like a `DateTimeIndex` or custom string labels), `drop()` becomes the safer and more readable choice for removing specific entries based on those labels. However, strictly for removing the positional first row (index 0), `iloc` provides superior speed and brevity.

Summary and Best Practices for Row Deletion

Effective data preparation requires fluency in both label-based and positional indexing methods. When the objective is strictly to remove the first row of a dataset, regardless of what the index label happens to be, the [iloc](#) slicing technique is the recommended best practice in [Python](#) for its efficiency and concise syntax.

Here is a concise summary of the two methods and their optimal use cases:

Positional Slicing (iloc): Syntax `df = df.iloc`. This is the fastest technique, ideal when you are certain you want to remove the row located at positional index 0. It always returns a new DataFrame copy.

Label-based Deletion (drop): Syntax `df.drop(index=df.index, axis=0, inplace=True)`. This method is more explicit and is preferred when dealing with non-integer index labels (e.g., specific dates or IDs) or when you require the `inplace` modification feature.

Mastery of both these techniques ensures flexibility and efficiency during the crucial data cleaning and preprocessing stages when working with the [Pandas DataFrame](#) structure.

Additional Resources

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