How to Easily Remove the First Column in a Pandas DataFrame

Authored by

stats writer

December 3, 2025

In the world of Data Analysis and manipulation using Pandas DataFrame objects, cleaning and preparing data is a foundational step. Often, datasets imported from external sources contain superfluous columns, such as internal indices or auto-generated keys, which are not needed for subsequent analysis. Removing these unwanted columns is a frequent necessity when preparing data for modeling or reporting.

Dropping the first column in a Pandas DataFrame can be efficiently achieved using several distinct Python methods. While the .drop() method is the most conventional approach, leveraging techniques like integer location indexing (.iloc) or the built-in Python del function provides flexible alternatives. Understanding these techniques is vital for writing clean, performant data processing scripts, especially when dealing with large datasets where efficiency matters.

Identifying the column to drop can be done either by its explicit name or, as required here, by its positional index. Since the goal is to remove the absolute first column, we utilize the positional index 0. This article will thoroughly explore three primary methods for isolating and removing the initial column from any Pandas DataFrame structure, analyzing the syntax and practical implementation of each.

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

Method 1: Use drop

df.drop(columns=df.columns, axis=1, inplace=True)

Method 2: Use iloc

df = df.iloc

Method 3: Use del

del df]

Each of these methods is functionally equivalent and produces the exact same result--a DataFrame missing the column at the zero index. The choice between them often comes down to personal preference, readability, and whether you require an in-place modification or the assignment of a new object.

Setting Up the Example DataFrame

To provide clear and reproducible examples, we will first create a sample Pandas DataFrame. This dataset simulates typical structured data, featuring columns that we might encounter during a routine Data Analysis task. The DataFrame includes columns for 'team', 'position', 'assists', and 'rebounds'.

The following code snippet demonstrates the necessary imports and the creation of our initial DataFrame object. Pay close attention to the structure, as the column 'team' is currently located at index 0, making it the target for removal in all subsequent examples. This setup is crucial for visualizing the outcome of the three column-dropping techniques.

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 displayed, the 'team' column is unequivocally the first column, holding the index position 0. We will now proceed with applying each method to confirm that this column is successfully eliminated from the DataFrame structure.

Method 1: Utilizing the Powerful .drop() Method

The .drop() method is the most standard and flexible way to remove rows or columns in Pandas.

When removing the first column using this technique, we do not reference the column by its specific name directly, but rather by dynamically retrieving the name of the column located at index 0 using the syntax df.columns.

This approach offers superior flexibility because it removes the column irrespective of its actual name, relying only on its positional index. The method requires two critical parameters for column removal: the axis=1 argument, which explicitly tells Pandas to operate on columns rather than rows (where axis=0 is the default), and the use of inplace=True, which modifies the original DataFrame object directly without requiring re-assignment.

The following code shows how to use the .drop() method function to drop the first column of the pandas DataFrame in place:

#drop first column of DataFrame
df.drop(columns=df.columns, axis=1, inplace=True)

#view updated DataFrame
df

position assists rebounds
0 G 5 11
1 G 7 8
2 F 7 10
3 F 9 6
4 G 12 6
5 G 9 5
6 F 9 9
7 F 4 12

Notice immediately that the first column called 'team' has been successfully removed from the DataFrame. The remaining columns shift to the left, and the DataFrame retains its row indices. The use of inplace=True is important here; without it, the function would return a new DataFrame copy, and the original df would remain unchanged unless explicitly reassigned (e.g., df = df.drop(...)).

Method 2: Slicing Columns with .iloc

The .iloc indexer is primarily used for integer-location based indexing and selection. While it is often employed to select specific rows and columns for extraction, it is highly effective for removal tasks through clever slicing. By selecting all rows but starting the column selection from index 1 (thereby skipping index 0), we effectively create a new DataFrame that excludes the initial column.

The slicing syntax is key to this operation. The first part, :, indicates that we select all rows. The second part, 1:, specifies that we select all columns starting from index 1 up to the end. This inherently excludes the column at index 0. Because .iloc always returns a new DataFrame slice, reassignment is necessary to update the original variable.

The following code shows how to use the .iloc function to drop the first column of the pandas DataFrame by slicing it away:

#drop first column of DataFrame
df = df.iloc

#view updated DataFrame
df

position assists rebounds
0 G 5 11
1 G 7 8
2 F 7 10
3 F 9 6
4 G 12 6
5 G 9 5
6 F 9 9
7 F 4 12

The result confirms that the 'team' column has been successfully eliminated. This method is often favored by Python users who are already comfortable with standard array slicing, as it feels idiomatic and concise. However, it requires reassigning the resulting slice back to the original DataFrame variable df.

Method 3: Direct Deletion using the del function

The del function (or statement) is a standard Python mechanism used to delete objects or items from collections. When applied to a Pandas DataFrame, which behaves much like a dictionary of Series objects where keys are column names, del provides the fastest and most direct way to remove a column in place.

Similar to the .drop() method approach, we must first identify the name of the column at the zero index using df.columns. Once the name is retrieved, it is passed as the key to the DataFrame object using square bracket notation. The del function then permanently removes that column from the underlying DataFrame structure.

The following code shows how to use the del function to drop the first column of the pandas DataFrame. This is an extremely memory-efficient operation as it modifies the object directly without creating copies.

#drop first column of DataFrame
del df]

#view updated DataFrame
df

position assists rebounds
0 G 5 11
1 G 7 8
2 F 7 10
3 F 9 6
4 G 12 6
5 G 9 5
6 F 9 9
7 F 4 12

Once again, the output confirms the successful removal of the 'team' column. The del statement is often preferred in performance-critical scenarios where removing a column requires minimal overhead, as it avoids function call overhead associated with methods like .drop() method.

Comparing Performance and Usage

While all three methods achieve the desired result of dropping the first column, they differ significantly in terms of execution speed, memory management, and code readability. Understanding these nuances helps in selecting the optimal method for different contexts within a Data Analysis workflow.

The del statement is generally the fastest mechanism for removing a column, particularly for large DataFrames, as it is a low-level operation that modifies the object in place directly. It avoids creating intermediate copies, making it highly memory-efficient. However, it is less descriptive than the .drop() method, which clearly indicates an intent to 'drop' an element.

Conversely, the .drop() method, when used with inplace=True, offers a good balance between readability and performance. Without inplace=True, or when using .iloc slicing, a copy of the DataFrame is created, which can be computationally expensive and memory-intensive when dealing with massive datasets. Therefore, for general-purpose scripting where clarity is paramount, .drop(inplace=True) is often the recommended standard.

Conclusion

We have successfully demonstrated three effective techniques for dropping the initial column in a Pandas DataFrame: utilizing the versatile .drop() method, leveraging the power of positional slicing with .iloc, and employing the efficient del function. Each method requires dynamically accessing the name of the column at index 0 using df.columns, a technique that ensures portability across different datasets.

For most data manipulation tasks, the .drop() method is the preferred tool due to its excellent readability and explicit handling of axes. However, when optimizing for speed and memory footprint, particularly in iterative processes or large-scale data cleansing, the del statement provides a marginal performance advantage. The .iloc method remains a reliable choice for those accustomed to numpy-style array indexing.

By mastering these three approaches, developers and data scientists gain flexibility in their data preparation workflow, ensuring that their Pandas operations are not only accurate but also optimized for the specific demands of their projects.