How to Extract the First Column from a Pandas DataFrame

Authored by

stats writer

December 4, 2025

When working with data analysis in Python, the Pandas DataFrame is the cornerstone structure, providing a robust way to manage tabular data. A common task involves accessing specific subsets of this data, particularly the initial column. Retrieving the first column is fundamental for many operations, such as feature selection or initial data inspection. The most precise and recommended method for accessing a column based on its numerical position, regardless of its label, is through the use of the powerful .iloc property. This method utilizes integer-based indexing, allowing developers to pinpoint the exact location of the desired data.

To retrieve the first column using integer indexing, the standard syntax is `df.iloc`. Here, the colon (`:`) signifies selecting all rows, while `0` refers specifically to the column at index zero--the very first column. It is critical to understand that when using a single integer index like `0` in the column selection axis, a Pandas Series object is returned. This distinction between a Series (one-dimensional data structure) and a DataFrame (two-dimensional data structure) output is vital for subsequent data manipulation tasks. We will explore how to force a DataFrame output shortly.

While `.iloc` is the preferred method for positional selection, Pandas offers flexibility. If the column name is known, you can use the standard column indexing notation, such as `df`. Alternatively, for cleaner syntax in many cases, especially when column names are valid Python variable identifiers, the dot notation (`df.column_name`) provides an accessible shortcut, returning the same result as bracket notation. Furthermore, if you are focusing on label-based selection rather than position, the `.loc` accessor can be employed, using the syntax `df.loc`. Understanding these varying methods allows for flexibility depending on whether you are working purely positionally or relying on explicit labels.

The subsequent sections detail the exact methods and implications of retrieving the first column of a Pandas DataFrame, ensuring that the resulting data structure--be it a Series or a DataFrame--aligns precisely with your analytical requirements. We will focus primarily on the robust and unambiguous positional indexing provided by the `.iloc` property.

Understanding the Importance of Column Indexing in Pandas

Effective indexing is the bedrock of high-performance data manipulation within the Pandas ecosystem. The ability to quickly and accurately select data subsets is paramount for tasks ranging from filtering and aggregation to model preparation. In the context of column selection, understanding the difference between positional indexing and label indexing is essential for writing resilient and readable code.

Positional indexing, implemented via the `.iloc` accessor, relies exclusively on the zero-based integer location of the rows and columns. This means that regardless of what the column is named (e.g., 'A', 'Name', 'Date'), the first column is always accessed using the index `0`. This method is

particularly crucial in workflows where the column names might change dynamically, or when iterating through columns programmatically based on their order of appearance in the DataFrame structure.

Conversely, label indexing, primarily handled by the `.loc` accessor or direct bracket notation, relies on the explicit column names defined in the DataFrame's index. While often more readable when dealing with fixed schemas, it can lead to errors if the column names are misspelled or change unexpectedly. Since our goal here is to consistently retrieve the **first** column, `.iloc` offers the greatest guarantee of success, as the position zero is immutable, whereas the label associated with that position is not.

The Primary Method: Using the .iloc Indexer

The `.iloc` property (short for **integer location**) is specifically designed for selection by position. It requires numerical indices for both the row and column axes, separated by a comma within the selection brackets. To target the entire first column, we must specify that all rows should be included while targeting the column index 0.

The standard syntax used to reliably extract the first column based on its position is achieved by employing Python slicing for the row axis and a specific integer for the column axis. This approach ensures that every record is captured while isolating only the necessary feature set. The colon `:` acts as a wildcard, indicating the selection of all available row indices, followed by the specific column index.

You can use the following syntax to get the first column of a pandas DataFrame:

df.iloc

This fundamental command is central to positional data extraction in Pandas. It leverages the concept of zero-based indexing common in Python programming. However, it is essential to be aware of the type conversion that occurs upon execution, as detailed in the next section, which dictates how the resulting data can be used in subsequent analysis.

Differentiating Output: Pandas Series vs. DataFrame

A critical consideration when retrieving a single column is understanding the data type of the result. Pandas is designed to automatically simplify the data structure when only a single dimension (a single column or a single row) is selected. When using `.iloc` with a single integer index (e.g., `0`), the output is automatically coerced into a one-dimensional Pandas Series.

A Series is suitable for operations where a vector of data is needed--for example, when calculating

descriptive statistics or passing data to a NumPy function. However, if your subsequent processing steps require the data to maintain the two-dimensional structure typical of a DataFrame (e.g., chaining indexing operations, using methods exclusive to DataFrames, or consistency in function input), you must adjust your selection syntax.

To ensure the output remains a DataFrame, even when selecting only a single column, we must use column slicing notation for the column axis instead of a single integer. By defining a range that starts at the desired index and ends one position after it, we force Pandas to retain the dimensional integrity of the result. For the first column (index 0), we use the slice `:1`.

If you'd like the result to be a Pandas DataFrame instead, you can use the following syntax:

df.iloc

The slice `:1` means "start at index 0 and stop before index 1." This range-based selection signals to Pandas that the output should maintain its DataFrame structure. Understanding this subtle difference between indexing with an integer (Series output) and indexing with a slice (DataFrame output) is essential for sophisticated Pandas usage.

Alternative Indexing Methods: Label-Based Selection

While positional indexing with `.iloc` is ideal for retrieving the technical "first column," there are robust alternatives for selection when the column's label is known. These methods often enhance code readability, particularly in static data pipelines where column names are fixed and meaningful.

The most standard label-based retrieval uses direct bracket notation, where the column name is passed as a string indexer (e.g., `df`). This method is highly flexible and works regardless of whether the column name adheres to Python variable naming conventions. It consistently returns a Series, equivalent to using `.iloc` if 'points' happened to be the first column.

For scenarios demanding strict label alignment, the `.loc` accessor provides an explicit mechanism. Similar to `.iloc`, `.loc` requires comma-separated indices, but it uses row and column labels instead of integers. If the first column is named 'points', the `.loc` equivalent would be `df.loc`. This method is crucial when the column order might shift, but you absolutely must retrieve the column identified by a specific name.

Finally, for rapid interactive work, the dot notation (e.g., `df.points`) offers the most concise syntax. However, this method is constrained: it only works if the column name does not contain spaces, special characters, or conflict with existing DataFrame attributes or methods. While convenient, reliance on dot notation in production code is often discouraged due to potential ambiguities and lack of flexibility compared to bracket or `.loc` notation. If 'points' is the first

column, `df.points` will retrieve the same Series as `df.iloc`.

The following examples show how to use this syntax in practice.

Practical Application: Retrieving the First Column (Series Output)

This example demonstrates the creation of a sample DataFrame and the subsequent extraction of its first column using the single integer index method, which results in a Pandas Series. This approach is standard when you need a one-dimensional array of values for vectorized calculations or feature extraction.

We begin by importing the Pandas library and constructing a simple DataFrame containing three statistical categories: 'points', 'assists', and 'rebounds'. Once the DataFrame is initialized, we use the powerful `.iloc` indexer to select all rows (`:`) and the column at position zero (`0`). This selection is assigned to the variable `first_col`, representing the 'points' column.

The resulting output clearly shows the index, the values from the 'points' column, and the metadata indicating its name and data type. Crucially, the final step involves explicitly checking the type of the resulting object, confirming that `df.iloc` returns the expected one-dimensional Series structure, which is vital for maintaining data consistency in downstream operations.

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

import pandas as pd

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

#view DataFrame
print(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

```
#get first column (Series output)
first_col = df.iloc

#view first column
print(first_col)
```

```
0 25
1 12
2 15
3 14
4 19
5 23
6 25
7 29
Name: points, dtype: int64
```

The result is confirmed to be a pandas Series:

```
#check type of first_col
print(type(first_col))
```

```
<class 'pandas.core.series.Series'>
```

Practical Application: Retrieving the First Column (DataFrame Output)

When the workflow dictates that the output must maintain a two-dimensional structure, even when selecting only one column, we must utilize column slicing. This example utilizes the same underlying DataFrame from the previous section but modifies the `.iloc` syntax to enforce a DataFrame return type.

Instead of passing the integer `0`, we pass the slice `:1` to the column indexer of `.iloc` (i.e., `df.iloc`). This tells Pandas to select all positions from the start (index 0) up to, but not including, index 1. This range selection preserves the dimensionality of the object, ensuring that the result is technically a DataFrame containing a single column.

Observing the output reveals that the column header ('points') and the DataFrame structure (including the alignment and internal representation) are maintained, unlike the clean vector output of the Series. The type check at the conclusion verifies that the object returned is indeed of the `pandas.core.frame.DataFrame` class, allowing for seamless integration into operations that require DataFrame inputs.

The following code shows how to get the first column of a pandas DataFrame and return a DataFrame as a result:

#get first column (and return a DataFrame)
first_col = df.iloc

#view first column
print(first_col)

points
0 25
1 12
2 15
3 14
4 19
5 23
6 25
7 29

#check type of *first_col*
print(type(first_col))

<class 'pandas.core.frame.DataFrame'>

Best Practices and Performance Considerations

Choosing the correct indexing method goes beyond simple retrieval; it significantly impacts code clarity, robustness, and performance. For retrieving the first column specifically, `df.iloc` is generally the fastest and most explicit way to specify the task positionally, particularly when dealing with massive datasets where performance optimization is key.

When developing production-level code, it is usually recommended to rely on label-based selection (e.g., `df` or `df.loc`) whenever possible, provided the column names are stable. This practice prevents silent errors if the underlying data schema changes its column order, ensuring that you are always selecting the intended feature regardless of its position.

However, if the task is strictly defined as retrieving the column at index 0--perhaps for an iterative process or if the data structure is guaranteed to be consistent--then `.iloc` remains the superior choice for positional access. Always document whether your code expects a Series or a DataFrame output and use the appropriate indexing method (integer vs. slicing) to ensure type consistency throughout your data pipeline, minimizing unexpected behavioral issues during

execution.

Summary of First Column Retrieval Methods

To summarize the options available for retrieving the first column of a Pandas DataFrame, understanding the desired output type (Series or DataFrame) is the determining factor in selecting the appropriate syntax. Below is a concise breakdown of the methods covered:

Positional Selection (Series Output): Use `df.iloc`. This is the simplest and most common method when a one-dimensional array of values is needed.
Positional Selection (DataFrame Output): Use `df.iloc`. This uses slicing to maintain the two-dimensional structure, useful for chained DataFrame operations.
Label Selection (Series Output): Use `df`. This relies on the explicit name of the column, highly recommended for readable and stable code.
Label Selection (DataFrame Output): Use `df]`. Note the double brackets here; Pandas interprets double brackets as selecting a list of column names, even if the list contains only one item, thus returning a DataFrame.

By mastering these various methods, data practitioners can navigate the complexities of Pandas indexing efficiently and ensure that data extraction aligns perfectly with the requirements of subsequent analytical steps.