# Pandas: Select Columns by Data Type

Authored by

**Mohammed looti**

October 27, 2025

## Introduction to Pandas DataFrames and Data Types

In the realm of Python for data analysis, the Pandas library stands out as an indispensable tool. It provides powerful and flexible data structures, most notably the DataFrame, which is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns). Understanding how to efficiently manipulate and filter these DataFrames is crucial for any data professional.

One fundamental aspect of working with data is recognizing and managing its data type. Each column in a Pandas DataFrame holds values of a specific data type, such as integers, floating-point numbers, strings (objects), or booleans. These data types dictate how operations are performed on the data and can significantly impact performance and memory usage.

This guide will delve into a highly practical method for selecting columns in a Pandas DataFrame based on their data types: the **`select_dtypes()`** function. This method offers a streamlined approach to filter your DataFrame, allowing you to include or exclude columns of particular types, which is immensely useful for data cleaning, feature engineering, and focused analysis.

## Understanding Pandas Data Types

Before we dive into the mechanics of selecting columns, it's beneficial to have a clear grasp of the common data types Pandas utilizes. These types are largely inherited from NumPy, the numerical computing library that forms the foundation of Pandas.

Common data types include:

**`int64`**: Represents integer values, such as counts or discrete measurements.

**`float64`**: Denotes floating-point numbers, used for continuous or decimal values.

**`object`**: Typically used for strings or mixed types. In most cases, if you see an `object` dtype, it means the column contains textual data.

**`bool`**: Stores Boolean values (`True` or `False`), often used for flags or logical indicators.

**`datetime64`**: Specifically designed for date and time information, allowing for sophisticated time-series analysis.

**`category`**: A memory-efficient type for columns that contain a limited number of unique values, useful for categorical variables.

Knowing the data types of your columns is paramount. It helps in diagnosing issues like incorrect calculations (e.g., trying to sum strings), ensures proper data visualization, and facilitates efficient

memory management. The **`select_dtypes()`** method empowers you to leverage this knowledge for precise column selection.

## The `select_dtypes()` Method: An Overview

The `select_dtypes()` method is a powerful feature within Pandas DataFrames designed specifically for filtering columns based on their data types. Instead of manually listing column names, which can be cumbersome for large DataFrames, `select_dtypes()` allows you to specify the types of columns you wish to include or exclude programmatically.

This method is particularly useful in scenarios such as:

Extracting only numerical columns for statistical analysis or machine learning models.

Isolating categorical or textual columns for specific preprocessing steps.

Cleaning data by identifying and handling non-numeric entries in columns expected to be numerical.

The primary parameters for `select_dtypes()` are **`include`** and **`exclude`**. You can pass a single data type or a list of data types to either of these parameters, providing immense flexibility in your selection criteria. We will explore both methods in detail with practical examples.

## Setting Up Our Example DataFrame

To illustrate the functionality of the **`select_dtypes()`** method, let's begin by creating a sample Pandas DataFrame. This DataFrame will contain various data types, allowing us to demonstrate how different selection criteria affect the output.

Consider the following Python code to construct our example DataFrame:

**import pandas as pd**

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

#view DataFrame
print(df)

team points assists minutes all_star
0 A 18 5 10.1 True
1 B 22 7 12.0 False
2 C 19 7 9.0 False
3 D 14 9 8.0 True
4 E 14 12 8.4 True
5 F 11 9 7.5 True

Upon creation, this DataFrame `df` comprises several columns, each with a distinct data type:

'**team**': This column contains character strings ('A', 'B', 'C', etc.), which Pandas typically infers as an `object` data type.

'**points**' and '**assists**': These columns consist of whole numbers, making them `int64` (integer) data types.

'**minutes**': This column holds decimal numbers, hence it is a `float64` (floating-point) data type.

'**all_star**': This column stores Boolean values (`True` or `False`), corresponding to the `bool` data type.

To confirm these data types, we can use `df.dtypes`:

**#display data type of all columns**
**df.dtypes**

team object
points int64
assists int64
minutes float64
all_star bool
dtype: object

This output clearly shows the inferred data type for each column, which will be the basis for our subsequent column selection operations.

## Method 1: Selecting Columns by Including Specific Data Types

The first approach to selecting columns based on their data types involves using the **`include`** parameter within the `select_dtypes()` method. This parameter allows you to specify a single data type or a list of data types that you wish to retain in your resulting DataFrame. All columns

matching these specified types will be included, while others will be excluded.

For instance, if your analytical task requires working exclusively with numerical data, you might want to select all columns that are either `int` or `float`. This is a common requirement in statistical modeling or when preparing data for algorithms that only accept numerical inputs.

Let's apply this to our example DataFrame. We will select all columns that have an **`int`** or **`float`** data type:

**#select all columns that have an int or float data type**
**df.select_dtypes(include=)**

points assists minutes
0 18 5 10.1
1 22 7 12.0
2 19 7 9.0
3 14 9 8.0
4 14 12 8.4
5 11 9 7.5

As observed in the output, only the '**points**', '**assists**', and '**minutes**' columns are returned. This is precisely because these columns are of type `int64` and `float64`, respectively, matching our inclusion criteria. The '**team**' (`object`) and '**all_star**' (`bool`) columns have been successfully filtered out. This method provides a clean and efficient way to isolate subsets of your data based on their fundamental nature.

## Method 2: Selecting Columns by Excluding Specific Data Types

Conversely, the `select_dtypes()` method also allows you to specify data types that you wish to **exclude** from your selection using the **`exclude`** parameter. This approach is particularly useful when you want to keep most columns but remove a few specific types, or when you want to work with all columns except those that are non-numeric or of a particular format.

For instance, you might want to remove all textual columns (typically `object`) and Boolean flags (`bool`) to focus solely on quantitative variables that are not binary. This can simplify your DataFrame for tasks that require only continuous or discrete numerical data.

Let's demonstrate this by selecting all columns in our DataFrame that do not have a data type equal to either **`bool`** or **`object`**:

**#select all columns that don't have a bool or object data type**

**df.select_dtypes(exclude=)**

points assists minutes
0 18 5 10.1
1 22 7 12.0
2 19 7 9.0
3 14 9 8.0
4 14 12 8.4
5 11 9 7.5

As you can see from the output, the columns '**team**' (`object`) and '**all_star**' (`bool`) have been successfully excluded, leaving only the numerical columns: '**points**', '**assists**', and '**minutes**'. This method offers a complementary way to filter your data, allowing you to define what you *don't* want rather than what you *do* want. It's an equally effective strategy depending on the specific filtering requirement.

## Advanced Considerations and Best Practices

While the **`include`** and **`exclude`** parameters are straightforward, there are a few advanced considerations and best practices to keep in mind when using `select_dtypes()`. These tips can further enhance your data manipulation workflow and address more complex filtering scenarios.

Firstly, it's important to note that you cannot use both **`include`** and **`exclude`** parameters simultaneously in a single call to `select_dtypes()`. If you specify both, Pandas will raise a `ValueError`. If your filtering logic requires both inclusion and exclusion criteria, you would typically apply one operation and then chain another or perform subsequent filtering steps. For example, you might first include all numerical types and then exclude specific numerical types in a second step if necessary.

Secondly, Pandas also allows for the use of NumPy dtypes or dtype categories. For instance, you can use `include='number'` to select all numerical columns (`int`, `float`), or `include=` for time-related columns. This categorical approach can simplify your code when dealing with broad groups of data types. Similarly, `exclude='object'` is a common shortcut to remove all string columns.

Finally, always remember to inspect the data types of your DataFrame using `df.dtypes` before and after applying `select_dtypes()`. This practice helps confirm that the filtering has worked as expected and that your DataFrame now contains the desired column subset with the correct data types. This simple check can prevent downstream errors and ensure the integrity of your data analysis.

## Conclusion

The `select_dtypes()` method in Pandas is an incredibly versatile and powerful tool for data professionals. It provides an elegant and efficient way to filter DataFrame columns based on their underlying data type, whether you choose to **`include`** specific types or **`exclude`** others.

By leveraging this method, you can streamline your data preparation workflows, making your code cleaner, more readable, and less prone to errors associated with manual column selection. Whether you are preparing data for machine learning models, performing statistical analysis, or simply cleaning your dataset, `select_dtypes()` is an essential function to have in your Pandas toolkit.

## Additional Resources

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