How to Select Rows Between Two Values in a Pandas DataFrame

Authored by

stats writer

November 20, 2025

One of the most common tasks in data analysis using Python is filtering datasets to select specific subsets of rows. When working with the powerful Pandas DataFrame structure, analysts frequently need to isolate records where a particular column's value falls within a defined numerical range. While there are multiple ways to achieve this, the most efficient and readable method involves the dedicated .between() function. This guide will walk you through the precise steps and syntax required to select all rows whose column values lie inclusively between two specified points.

Understanding the Core Filtering Concept

To select rows between two specific values in a Pandas DataFrame, we employ boolean indexing. This technique involves creating a boolean series (a series of `True`/`False` values) that evaluates whether each row meets the specified condition. This resulting boolean series is then used as an index mask to filter the original DataFrame, retaining only those rows marked as `True`. For range selection, the `Series.between()` method simplifies this process significantly compared to using chained logical operators like `&` (AND).

The Primary Method: Using the .between() Function

The `Series.between()` method is specifically designed for range filtering. When applied to a column (which is a Pandas Series), it checks if each element is greater than or equal to the lower bound and less than or equal to the upper bound. By default, this function is **inclusive**, meaning the start and end values themselves are included in the resulting selection. This function returns a boolean Series which is then passed back to the DataFrame to perform the filtering operation.

Basic Syntax for Filtering Data Using .between()

The basic syntax for filtering rows where a column value falls within a specific range is straightforward and highly optimized for performance in Pandas. You must specify the column name, followed by the call to `.between()`, supplying the lower and upper bounds as arguments. This structure provides immediate readability regarding the intended filtering action.

The following syntax demonstrates how to select rows in a pandas DataFrame where the values in a specified column fall within two boundaries:

df_filtered = df.between(25, 35)]

In this particular example, the resulting `df_filtered` DataFrame will contain only those rows where the value in the **points** column is **between** 25 and 35 (inclusive of both endpoints). This approach is robust and preferred over manual comparison operators for range checks.

Negating the Selection: Filtering Rows Not Between Two Values

Often, data analysis requires identifying data points that fall outside a critical range rather than inside it. To select rows where the column value is **not between** the specified bounds, we utilize the logical NOT operator, represented by the tilde (~) symbol, placed immediately before the filtering condition. This inverts the boolean Series generated by `.between()`.

By applying the tilde (~) operator, every `True` result from the `.between()` call becomes `False`, and every `False` result becomes `True`, thereby selecting all data points outside the designated range.

df_filtered = df.between(25, 35)]

The following detailed examples illustrate how to implement both the inclusion and exclusion methods in a practical data context, using a sample dataset common in sports analytics.

Practical Application: Setting Up the Sample Pandas DataFrame

To demonstrate these filtering techniques clearly, we will create a sample Pandas DataFrame in Python. This DataFrame contains data on various basketball teams and the points scored by their players. We begin by importing the necessary **pandas** library and constructing the data structure.

This setup is crucial for reproducible examples, allowing us to accurately test the `.between()` functionality against known numerical values.

import pandas as pd

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

#view DataFrame
print(df)

team points
0 Mavs 22
1 Mavs 28
2 Nets 35
3 Nets 34
4 Heat 29
5 Heat 28
6 Kings 23

Example 1: Selecting Rows Within the Specified Range (25 and 35)

We now apply the `.between()` method to the `points` column to filter for all rows where the points scored are inclusively between 25 and 35. This operation provides a quick way to isolate players who had moderate-to-high scoring performances within this specific bracket.

Executing this command returns a new DataFrame containing only the subset of data that satisfies the condition, demonstrating the efficiency of Pandas boolean indexing combined with the .between() function.

#select rows where value in points column is between 25 and 35
df_filtered = df.between(25, 35)]

#view filtered DataFrame
print(df_filtered)

team points
1 Mavs 28
2 Nets 35
3 Nets 34
4 Heat 29
5 Heat 28

Observe the output: only the rows where the value in the **points** column is between 25 and 35 have been selected. Rows 0 (22 points) and 6 (23 points) are successfully excluded.

Understanding Inclusivity in .between()

It is important to emphasize that by default, the .between() function is defined to **include** the values provided for the lower and upper bounds. If you look closely at the filtered results from Example 1, the player entry with a points value of 35 (Row 2) is included in the filtered DataFrame. This inclusivity is the standard behavior, although it can be modified using the `inclusive` parameter (e.g., setting `inclusive='left'`, `'right'`, or `'neither'`).

If your requirement is to strictly select values greater than the lower bound and less than the upper bound, you must specify `inclusive='neither'` within the function call, or alternatively, use explicit chained comparison operators.

Example 2: Selecting Rows Outside the Specified Range

To identify the players whose scores fall outside the 25 to 35 range (i.e., those who scored 24 or

less, or 36 or more), we introduce the tilde (~) negation operator before the filtering condition. This selects the complement set of the data found in Example 1.

This technique is vital when looking for outliers or identifying data points that fail to meet a required minimum threshold or exceed a maximum cap.

#select rows where value in points column is not between 25 and 35
df_filtered = df.between(25, 35)]

#view filtered DataFrame
print(df_filtered)

team points
0 Mavs 22
6 Kings 23

As demonstrated, only the rows where the score is not between 25 and 35 (i.e., 22 and 23) have been selected. The negation operator successfully isolated the two lowest-scoring players in our dataset.

Alternative Method: Index-Based Slicing with DataFrame.loc()

While the `.between()` method is used for value-based filtering, if the original question implies selecting rows based on their numerical index positions rather than their contents, we can utilize the DataFrame.loc() method. This method is primarily used for label-based indexing, but when slicing using integer indices, it behaves similarly to standard Python slicing, with a crucial difference: it is **inclusive** of the end index.

The syntax for index slicing involves passing the start row label and the end row label (or index number, if using default integer indexing) separated by a colon (`:`) inside the `.loc` accessor. For instance, if you wanted to select all rows between index 3 and index 8 (inclusive), you would use the command: `df.loc`. This would return a DataFrame containing rows at index 3, 4, 5, 6, 7, and 8.

Summary of Selection Techniques

Choosing the correct method depends entirely on the nature of the data requirement. If you need to filter rows based on the value contained within a column, use the powerful and highly readable `Series.between()` function. If, however, you need to extract a continuous block of rows based purely on their position or index label, the `DataFrame.loc` accessor is the appropriate tool. Mastering these two methods allows for precise and efficient data extraction within Pandas

DataFrames.