How to Display Full Column Content in PySpark DataFrames

Authored by

stats writer

January 19, 2026

The Necessity of Full Column Display in PySpark

When working with large datasets and complex transformations within PySpark, viewing the data accurately is fundamental for verification and validation. The primary method for inspecting data contained within a DataFrame is using the native show() function. While incredibly useful for quickly sampling results, this function defaults to a behavior that often frustrates developers and data scientists: column content truncation.

By default, PySpark imposes a width limit on the columns displayed, meaning that lengthy strings or complex identifiers are cut off, typically replaced by ellipses (`...`). This truncation, while designed to keep the output readable and formatted neatly on standard terminals, severely hinders detailed debugging and thorough data quality checks. If you cannot see the entire content of a cell, you cannot reliably confirm the integrity of your data processing pipelines or the results of string manipulation operations.

Therefore, mastering the technique to display full column content is not merely a convenience, but a critical skill for anyone performing serious work in the Spark ecosystem. This process involves explicitly overriding the default display behavior of the show() function, ensuring that every character within the specified column is rendered completely, irrespective of its width. The following sections detail the standard and recommended methods for achieving this full visibility in PySpark.

Understanding PySpark's Default Truncation Behavior

The default behavior of the show() function in PySpark is designed for efficiency and presentation within a standard terminal window. Specifically, it limits the visible width of string columns to 20 characters. Any string exceeding this limit is truncated, often masking vital information within the dataset. This default limit is imposed to prevent excessively wide output that is difficult to scroll through and interpret quickly.

However, when dealing with rich textual data, long URLs, detailed error messages, or columns containing complex identifiers, this 20-character limit is often insufficient. Identifying subtle differences between records, such as trailing spaces or minor spelling variations, becomes impossible when the content is summarized by an ellipsis. This can lead to misdiagnosed data issues, incorrect ETL logic, or flawed results during final data analysis stages.

To combat this inherent limitation, PySpark provides an optional parameter within the `show` method that allows the user to disable this truncation mechanism entirely, thereby forcing the display of the complete string values for all columns. We will explore the two primary ways to invoke this non-truncation behavior: using a boolean flag or a numerical zero value.

Method 1: Utilizing the truncate=False Argument

The most explicit and commonly recommended way to force a DataFrame to show the full content of each column is by passing the argument `truncate=False` to the show() function. This boolean flag directly instructs the function's rendering logic to bypass the standard width restrictions.

By setting `truncate` to `False`, the function calculates the required column width based on the longest string entry in that column, ensuring that the entire content is visible without being cut off. This modification only affects the output visualization; it does not alter the underlying data or the DataFrame structure itself.

This method is highly readable and immediately conveys the intent of the programmer--to view the untruncated data. It is the preferred approach for clarity in code, especially during interactive sessions or when writing reproducible examples for colleagues.

You can use the following methods to force a PySpark DataFrame to show the full content of each column, regardless of width:

Method 1: Use truncate=False

df.show(truncate=False)

Method 2: Leveraging the truncate=0 Argument

An alternative, yet functionally identical, approach to disabling truncation involves setting the `truncate` parameter to zero (`truncate=0`). While `truncate` primarily expects a boolean value (True or False) to enable or disable the feature, it can also accept an integer value. If an integer `N` is provided, it dictates the exact number of characters to display before truncation occurs.

When `truncate=0` is specified, it effectively tells the show() function to set the truncation width to zero characters. Since displaying zero characters is logically equivalent to displaying the full, untruncated content (as there is no width limit imposed), this also results in the full display of all column values.

In practice, both `truncate=False` and `truncate=0` achieve the exact same output. The choice between the two is generally a matter of personal or team style preference, although `truncate=False` is often considered more semantically clear regarding the intention to disable the feature entirely.

Method 2: Use truncate=0

df.show(truncate=0)

Setting Up the Demonstration DataFrame

To illustrate these methods effectively, we first need to define a simple DataFrame containing a column with string values that exceed the default 20-character limit. This setup requires initializing a SparkSession, which is the entry point for all Spark functionality, followed by defining the raw data and schema.

The example below defines a list of data where the 'employees' column entries are deliberately long. By viewing the default output, we can clearly observe how PySpark handles these extensive strings when no truncation parameters are provided. This baseline view confirms the problem we are attempting to solve.

The following examples show how to use each method in practice with the following PySpark DataFrame:

from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

#define data
data = ,
,
,
,
]

#define column names
columns =

#create dataframe using data and column names
df = spark.createDataFrame(data, columns)

#view dataframe
df.show()

+-----+--------------------+-----+
|store| employees|sales|
+-----+--------------------+-----+

A	Andy Bob Chad Dou...	136
B	Frank Henry	223
C	Ian John Ken Liam...	450
D	Oscar Prim	290
E	Quentin Ross Sarah	189
+-----+--------------------+-----+

Analyzing the Truncated Default Output

Upon viewing the output of the standard `df.show()` command above, it is immediately apparent that the 'employees' column is truncated for several rows. Rows A and C, which contain names longer than the 20-character default width, are cut short, ending with `...`. This prevents the user from determining the full list of employees assigned to stores A and C.

This default truncation behavior, while visually neat, masks crucial data points. For instance, in the first row, we see "Andy Bob Chad Dou...", which doesn't give us visibility into whether the last name listed is "Eric" or something else entirely. In a real-world scenario where data accuracy is paramount, this lack of visibility necessitates immediate corrective action using the available parameters of the show() function.

The issue confirms that some of the rows in the **employees** column are cut off because they exceed the default width in PySpark, which is 20 characters. This limitation must be explicitly bypassed when performing detailed inspection or when using the output for validation purposes during complex transformations.

Example 1: Showing Full Column Content Using truncate=False

We can now apply the first, highly readable method to resolve the truncation issue: setting the `truncate` argument to `False`. This simple modification forces the display engine to adapt the column width dynamically based on the longest element present in the 'employees' column.

The execution of `df.show(truncate=False)` yields a table where the 'employees' column expands horizontally to accommodate the longest string ("Ian John Ken Liam Mike Noah"). This ensures complete fidelity of the output data representation, making it suitable for thorough quality checks.

Example 1: Show Full Column Content Using truncate=False

We use the **truncate=False** argument to show the full content of each column in the PySpark DataFrame:

#view dataframe with full column content
df.show(truncate=False)

```
+-----+-------------------------+-----+
|store|employees |sales|
+-----+-------------------------+-----+
|A |Andy Bob Chad Doug Eric |136 |
|B |Frank Henry |223 |
|C |Ian John Ken Liam Mike Noah|450 |
|D |Oscar Prim |290 |
|E |Quentin Ross Sarah |189 |
+-----+-------------------------+-----+
```

Notice that we can now see the full content of the **employees** column, confirming that the names previously cut off are now fully visible.

Example 2: Showing Full Column Content Using truncate=0

As discussed, setting the truncation limit to zero provides an identical functional result to using `truncate=False`. This method is often used by programmers who prefer numerical inputs for parameters, or who are familiar with how truncation parameters are handled in other programming environments.

Executing `df.show(truncate=0)` verifies that the output structure is precisely the same as the previous example. The column width adjusts dynamically, presenting all characters from the longest string entry in the 'employees' column.

The ability to use both a boolean flag and a zero integer provides flexibility, though consistency within a project is often advised. Regardless of the syntax chosen, the core goal--achieving full data visibility--is accomplished efficiently.

Example 2: Show Full Column Content Using truncate=0

We can also use the **truncate=0** argument to show the full content of each column in the PySpark DataFrame:

#view dataframe with full column content
df.show(truncate=0)

```
+-----+-------------------------+-----+
|store|employees |sales|
```

```
+-----+-------------------------+-----+
|A |Andy Bob Chad Doug Eric |136 |
|B |Frank Henry |223 |
|C |Ian John Ken Liam Mike Noah|450 |
|D |Oscar Prim |290 |
|E |Quentin Ross Sarah |189 |
+-----+-------------------------+-----+
```

Once again, we can now see the full content of the **employees** column, demonstrating that both arguments successfully disable the default truncation limit.

Summary and Best Practices for Data Inspection

Displaying untruncated column content in PySpark is essential for tasks requiring absolute data fidelity, such as specialized data analysis, detailed data profiling, or rigorous debugging of ETL scripts. The simple inclusion of `truncate=False` or `truncate=0` in the show() function call resolves the issue caused by the default 20-character limit.

While using `truncate=False` is generally preferred for its clarity, it is important to remember that displaying untruncated data can result in very wide output tables, potentially scrolling far off the screen. For extremely wide datasets, consider selecting only the specific columns you need to inspect before calling `show(truncate=False)` to maintain terminal readability. For example, use `df.select('col_with_long_strings', 'other_key_col').show(truncate=False)`.

For developers seeking deeper understanding of these display parameters and other functions available for the DataFrame API, consulting the official documentation is always recommended. This ensures you are utilizing the most current and optimized methods for data manipulation and visualization within the Apache Spark ecosystem.

Note: You can find the complete documentation for the PySpark **show** function here.

Further PySpark Tutorial Resources

The following tutorials explain how to perform other common tasks in PySpark, covering everything from complex joins to optimization strategies:

Handling Null Values and Data Imputation Strategies in Spark.
Using Window Functions for Advanced Analytical Queries on DataFrames.
Optimizing Shuffle Operations Using Configuration Settings in PySpark.
Implementing User Defined Functions (UDFs) for Custom Logic.

Mastering these techniques will significantly enhance your ability to manipulate and analyze vast quantities of data efficiently using the power of distributed computing provided by Apache Spark.

</p