How to Remove Spaces from PySpark Dataframe Column Names Easily

Authored by

stats writer

January 1, 2026

The simplest and most scalable method to cleanse column names within a PySpark DataFrame involves leveraging a combination of List Comprehension and the `.alias()` function. While the `.withColumnRenamed()` method is effective for singular changes, it quickly becomes cumbersome when dealing with dozens or hundreds of columns requiring transformation. The optimized approach allows you to iterate through all column names programmatically and apply a string manipulation function--such as `.replace()`--to standardize the naming convention efficiently, often replacing spaces with underscores.

The Challenge of Whitespace in PySpark Column Names

Although modern data systems are increasingly flexible, having whitespace or special characters in column identifiers remains a significant obstacle in data processing. When working with Apache Spark, especially when integrating with other tools like SQL queries, Parquet files, or other programming languages (such as Scala or Java), column names must adhere to strict conventions. Spaces often cause parsing errors, require tedious quoting in SQL statements, and violate best practices for maintainable code bases.

When ingesting data from external sources, such as legacy databases or CSV files where human-readable headers were prioritized over technical naming conventions, it is common to encounter column names like "Customer Name" or "Total Revenue (USD)." Standardizing these names, typically by converting them to snake_case (e.g., `customer_name`), is a critical step in the ETL (Extract, Transform, Load) pipeline. This preparation ensures downstream analytical and machine learning processes run smoothly without namespace conflicts or unexpected errors.

Why Bulk Renaming is Necessary

The standard `.withColumnRenamed()` transformation is ideal if you only need to change one or two columns. It requires explicitly naming both the old and the new column, which is straightforward but not scalable. If your PySpark DataFrame has fifty columns, manually writing fifty lines of `df = df.withColumnRenamed("Old Name", "New_Name")` is inefficient and prone to typographical errors. Furthermore, this manual method fails to address the inherent dynamism of data schema--if a new column with a space is added to the source data tomorrow, the existing transformation logic will break or miss the new column.

To overcome the limitations of manual renaming, we utilize PySpark's functional programming capabilities combined with Python's powerful List Comprehension structure. This allows us to inspect the list of current column names (`df.columns`), apply a transformation function to every element in that list, and then use the `.select()` method to reconstruct the DataFrame with the new, corrected column names in a single, highly readable operation.

The Efficient Solution: Using List Comprehension and Aliasing

The most robust technique involves iterating over the existing column names and using the `.alias()` function to assign a transformed name to the column reference. This approach is encapsulated within a list comprehension, which generates a list of column expressions ready for the `.select()` transformation. This methodology guarantees that every column is processed, regardless of the DataFrame's width, ensuring uniformity across the entire dataset structure.

The core structure involves importing the necessary functions from pyspark.sql.functions and then executing a selection operation. The logic is designed to take the original column name `x`, apply the Python `.replace()` method (e.g., `x.replace(' ', '_')`), and use this modified string as the alias for the column represented by `F.col(x)`. This selection process effectively recreates the DataFrame with the desired schema changes applied.

You can use the following syntax to remove spaces from each column name in a PySpark DataFrame, replacing them with underscores:

from pyspark.sql import functions as F

#replace all spaces in column names with underscores
df_new = df.select()

This snippet is highly efficient because the column transformation occurs entirely within the Apache Spark execution plan, utilizing optimized functions rather than relying on less efficient user-defined functions (UDFs) or iterative Python loops outside the distributed context. It is the recommended best practice for schema standardization in large-scale data environments.

Practical Example Setup: Creating the PySpark DataFrame

To demonstrate this functionality, we will first create a sample PySpark DataFrame containing intentionally poorly named columns. This DataFrame models basketball statistics, where the column headers contain spaces, which would complicate subsequent filtering or join operations.

We start by initializing the `SparkSession` and defining our data array and the problematic column list. Notice how the column names `'team name'`, `'points scored'`, and `'total assists'` all include whitespace, necessitating cleanup before analysis.

Example: How to Remove Spaces from Column Names in PySpark

Suppose we have the following PySpark DataFrame that contains information about various

basketball players:

```
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()
```

```
+---------+-------------+-------------+
|team name|points scored|total assists|
+---------+-------------+-------------+
| Mavs| 18| 4|
| Nets| 33| 9|
| Hawks| 12| 7|
| Thunder| 15| 3|
| Lakers| 19| 2|
| Cavs| 24| 5|
| Magic| 28| 7|
+---------+-------------+-------------+
```

As observed in the output above, the column headers are not conventionally machine-readable due to the embedded spaces. This structure necessitates a renaming step to ensure smooth operation when integrating with external systems or applying complex Spark SQL logic. Our goal is to convert these column names into a format that uses underscores as separators, aligning with standard Python and Spark conventions (e.g., `team_name`).

Applying the Transformation and Verifying Results

We will now apply the powerful list comprehension technique using the F.col() and `.alias()` methods. This operation generates a new DataFrame (`df_new`) where the structure is identical to the original, but the schema has been updated to reflect the desired names. The use of the standard Python `.replace()` function makes this transformation exceptionally straightforward and readable.

We import the `functions` module from `pyspark.sql`, aliasing it as `F` for brevity. The list comprehension iterates through `df.columns`, takes each column name `x`, replaces the space with an underscore (`'_'`), and instructs Spark to select the original column represented by `F.col(x)` but assign it the new, clean name using `.alias(...)`.

We can use the following syntax to replace the spaces in each column name with an underscore instead:

from pyspark.sql import functions as F

#replace all spaces in column names with underscores
df_new = df.select()

#view new DataFrame
df_new.show()

+---------+-------------+-------------+
|team_name|points_scored|total_assists|
+---------+-------------+-------------+
Mavs	18	4
Nets	33	9
Hawks	12	7
Thunder	15	3
Lakers	19	2
Cavs	24	5
Magic	28	7
+---------+-------------+-------------+

Observe the resulting DataFrame `df_new`. The column names have been successfully standardized: `team name` became `team_name`, `points scored` became `points_scored`, and `total assists` became `total_assists`. This transformation is idempotent and guarantees a clean, machine-readable schema suitable for all subsequent analytical tasks within the Apache Spark ecosystem.

Alternative Approach: Complete Removal of Whitespace

While replacing spaces with underscores (creating snake_case) is the industry standard for clarity and compatibility, some use cases may require the complete removal of all whitespace, resulting in concatenated names (e.g., `pointsscored`). Although less readable, this can be achieved just as easily by modifying the second argument of the `.replace()` function. Instead of passing an underscore (`'_'`), we simply pass an empty string (`''`).

This method maintains the core efficiency of the list comprehension and aliasing technique but changes the resulting string manipulation logic. It is important to consider the potential readability impact when choosing this approach, especially for column names that are complex or very long.

You could simply remove the spaces from each column name without replacing them with another character by using the following syntax:

from pyspark.sql import functions as F

#remove all spaces in column names
df_new = df.select()

#view new DataFrame
df_new.show()

+--------+------------+------------+
|teamname|pointsscored|totalassists|
+--------+------------+------------+
Mavs	18	4
Nets	33	9
Hawks	12	7
Thunder	15	3
Lakers	19	2
Cavs	24	5
Magic	28	7
+--------+------------+------------+

Notice that the spaces in each column name have simply been removed, resulting in concatenated names like `pointsscored`. This demonstrates the flexibility of using standard Python string methods within the List Comprehension applied to the DataFrame schema.

Advanced Considerations for Column Naming Conventions

The method detailed above is highly effective for simple whitespace replacement. However, real-world data often presents challenges beyond simple spaces, such as leading/trailing whitespace, dashes, special characters (like `$, %, #`), or inconsistent capitalization. To handle these complexities, the standard Python `.replace()` function can be supplemented or replaced by more sophisticated regular expression tools.

For instance, to handle any non-standard character, you might use Python's `re` module within the list comprehension, although this requires careful design to ensure serialization efficiency in Spark. A simpler and often sufficient approach is to chain string methods: use `x.strip()` to remove leading/trailing whitespace, and then use `x.replace('-', '_').replace('.', '_')` to handle other common separators before finalizing the column name transformation.

It is crucial to adopt and enforce a consistent naming convention, such as **snake_case** for all columns. Consistency minimizes debugging time and improves code collaboration across teams, regardless of whether the data is being processed via Python, R, or SQL on the Spark cluster.

Summary of PySpark Column Manipulation Techniques

The optimal technique for renaming multiple columns in a PySpark DataFrame involves using list comprehension coupled with the `.select()` and `.alias()` functions. This approach is superior to manual iteration using `.withColumnRenamed()` for bulk operations due to its concise nature and guaranteed application across all columns.

The transformation relies on the standard Python `replace` function to modify the strings representing the column headers. For developers seeking further detail on the functionality used, the documentation for `F.col()` and other functions within pyspark.sql.functions provides comprehensive information on column manipulation within the Spark context.

The following tutorials explain how to perform other common tasks in PySpark: