Sum Multiple Columns in PySpark (With Example)

Authored by

stats writer

November 16, 2025

Introduction: The Strategy for Row-Wise Summation in PySpark

Working with large datasets often requires calculating metrics across various columns for every single record--a process known as a row-wise operation. When using PySpark, summing the values of multiple columns to create a new derived column is a core skill for feature engineering and aggregation. While there are several methods, leveraging built-in SQL expressions via the `F.expr()` function offers the best combination of clarity, performance, and scalability across distributed clusters.

This guide details the precise syntax required to aggregate columns dynamically within a DataFrame. We will focus on the highly optimized technique involving the construction of a SQL-like expression string, which PySpark's Catalyst Optimizer processes efficiently.

The PySpark Methodology: Using F.expr() with withColumn

To sum columns efficiently, we combine three essential elements: the `withColumn` transformation, the `pyspark.sql.functions` module (aliased as `F`), and Python's string `.join()` method. The `.join()` method dynamically generates the sum expression by placing the addition operator (`+`) between the names of all target columns.

The resulting expression string is passed to `F.expr()`, which instructs Spark to execute this custom SQL expression for every row. The entire transformation is wrapped within `withColumn` to append the result to the DataFrame as a new column.

from pyspark.sql import functions as F

#define columns to sum
cols_to_sum =

#create new DataFrame that contains sum of specific columns
df_new = df.withColumn('sum', F.expr('+'.join(cols_to_sum)))

This powerful snippet creates a new column called `sum` that contains the total of values across the designated columns--in this case, **game1**, **game2**, and **game3**. Using `F.expr()` ensures that this calculation is executed optimally by the Spark engine.

Example: Setting Up the PySpark DataFrame

To demonstrate this syntax in a practical context, let us define a sample DataFrame containing scores for various basketball teams across three separate games. Our goal is to calculate the total points scored by each team.

First, we must initialize the Spark context and define the data structure. This preparatory step ensures we have a valid PySpark DataFrame object (`df`) on which to apply our aggregation logic.

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|game1|game2|game3|
+-------+-----+-----+-----+
Mavs	25	11	10
Nets	22	8	14
Hawks	14	22	10
Kings	30	22	35
Bulls	15	14	12
Blazers	10	14	18
+-------+-----+-----+-----+

The resulting DataFrame `df` clearly outlines the points scored per game for six different teams. Our objective now is to introduce a new column, `sum`, that reflects the total points scored across the three game columns (`game1`, `game2`, and `game3`).

Executing the Column Summation in Practice

We now apply the dynamic expression method discussed previously. We first define the list of columns to be summed. By passing this list to `'+'.join()`, we programmatically construct the SQL expression `'game1+game2+game3'`, which is then utilized by `F.expr()` inside the `withColumn` call.

This approach is highly recommended for its stability and performance, as it leverages the core capabilities of pyspark.sql.functions and avoids the overhead of less efficient Python-based loops or User Defined Functions (UDFs).

from pyspark.sql import functions as F

#define columns to sum
cols_to_sum =

#create new DataFrame that contains sum of specific columns
df_new = df.withColumn('sum', F.expr('+'.join(cols_to_sum)))

#view new DataFrame
df_new.show()

```
+-------+-----+-----+-----+---+
| team|game1|game2|game3|sum|
+-------+-----+-----+-----+---+
| Mavs| 25| 11| 10| 46|
| Nets| 22| 8| 14| 44|
| Hawks| 14| 22| 10| 46|
| Kings| 30| 22| 35| 87|
| Bulls| 15| 14| 12| 41|
|Blazers| 10| 14| 18| 42|
+-------+-----+-----+-----+---+
```

The resulting DataFrame `df_new` now includes the `sum` column, successfully calculated across the rows. This column accurately reflects the total points achieved by each respective team over the

three games.

Validating the Row-Wise Aggregation

A crucial step in any data transformation process is validation. By examining the output above, we can confirm that the row-wise operation functioned as intended, correctly adding the values from `game1`, `game2`, and `game3` for each team.

For instance, observe the calculation for the initial entries:

The sum of points for the **Mavs** team is 25 + 11 + 10, totaling **46**.
The sum of points for the **Nets** team is 22 + 8 + 14, totaling **44**.
The sum of points for the **Hawks** team is 14 + 22 + 10, totaling **46**.

The consistency between the source columns and the derived `sum` column confirms the successful application of the dynamic expression methodology using `withColumn` and `F.expr()`. This technique ensures that complex calculations are performed natively by the Spark engine, achieving high throughput necessary for big data processing.