How to Calculate Summary Statistics in PySpark: A Step-by-Step Guide

Authored by

stats writer

February 2, 2026

RECOMMENDED CITATION

stats writer (2026). *How to Calculate Summary Statistics in PySpark: A Step-by-Step Guide*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=129242

PySpark is globally recognized as a powerful and scalable tool essential for performing extensive data analysis and manipulation, particularly when dealing with large datasets (Big Data). A fundamental requirement in any initial exploration of data is the calculation of summary statistics, which provide crucial, early insights into the distribution, central tendency, and variability of the underlying data. Understanding these basic metrics allows data scientists to quickly identify patterns, anomalies, and potential data quality issues before moving on to complex modeling.

The efficiency of PySpark stems from its ability to distribute computational tasks across a cluster, making the calculation of statistical summaries on petabytes of data feasible and fast. PySpark provides streamlined, built-in functions designed specifically for statistical aggregation. These functions include essential measures like the mean, median (50th percentile), standard deviation (stddev), and various quartiles (25th and 75th percentiles). These statistical operators can be applied either to specific columns of a PySpark DataFrame or across the entire dataset, offering a comprehensive and customizable overview.

Furthermore, PySpark is not limited to its predefined set of statistical metrics; it offers advanced flexibility allowing users to define and implement custom summary statistics calculations using user-defined functions (UDFs). Given its robust efficiency in parallel processing and its extensive library of statistical capabilities, PySpark represents the ideal platform for efficiently computing summary statistics, making it an indispensable tool in the modern data ecosystem.

Calculating Comprehensive Summary Statistics in PySpark

To effectively analyze the characteristics of your data stored within a PySpark DataFrame, the framework offers highly optimized methods for computing descriptive statistics. The primary tool for this purpose is the built-in **.summary()** transformation, designed for high performance on distributed clusters. We will explore three primary ways to utilize the .summary() function, ranging from broad, default calculations to highly specific, filtered analyses.

The flexibility of the .summary() function allows data professionals to quickly generate the statistical views necessary for exploratory data analysis (EDA). Understanding these methods is key to unlocking the full potential of PySpark for large-scale statistical reporting. Below, we detail the core syntax patterns used to calculate summary statistics for columns in a PySpark DataFrame:

Method 1: Calculating Comprehensive Default Statistics for All Columns

This is the simplest and most commonly used method. When called without any arguments, the **.summary()** function calculates a predefined set of comprehensive statistics (count, mean,

standard deviation, min, max, and the three quartiles) for all columns present in the DataFrame. While it provides a quick overview, caution is advised when interpreting numerical statistics (like mean or standard deviation) for non-numeric (string or categorical) columns, as the resulting values will typically be null or nonsensical.

df.summary().show()

Method 2: Calculating Specific Statistical Metrics

For scenarios where only a subset of statistical measures is required--perhaps only the range and the central tendency--PySpark allows the user to pass specific strings as arguments to the **.summary()** function. This approach reduces unnecessary computation and produces a cleaner, more focused output table, ideal for targeted reporting. The standard arguments accepted include 'min', '25%', '50%', '75%', and 'max'.

df.summary('min', '25%', '50%', '75%', 'max').show()

Method 3: Focusing Analysis on Only Numeric Columns

In many real-world analysis tasks, it is crucial to restrict statistical calculations to columns that contain quantitative data. Attempting to calculate the mean or standard deviation of string columns (like names or categories) results in null values or errors. This method involves a pre-filtering step where we identify and select only the numeric columns based on their data types before applying the .summary() function.

numeric_cols =

df.select(*numeric_cols).summary().show()

PySpark Setup and Sample Data Preparation

The following examples show how to use each method in practice with a concrete PySpark DataFrame that contains information about various basketball players. This setup is required to initialize the Spark environment and define the structured data.

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|conference|points|assists|
+----+----------+------+-------+
| A| East| 11| 4|
| A| East| 8| 9|
| A| East| 10| 3|
| B| West| 6| 12|
| B| West| 6| 4|
| C| East| 5| 2|
+----+----------+------+-------+
```

Example 1: Comprehensive Summary Statistics (Default Method in Action)

We use the following syntax to calculate the full set of default summary statistics for all columns in the DataFrame. This provides the most complete initial overview of the dataset.

#calculate summary statistics for each column in DataFrame
df.summary().show()

```
+-------+----+----------+-----------------+------------------+
|summary|team|conference| points| assists|
+-------+----+----------+-----------------+------------------+
| count| 6| 6| 6| 6|
| mean|null| null|7.666666666666667| 5.666666666666667|
| stddev|null| null|2.422120283277993|3.9327683210007005|
| min| A| East| 5| 2|
```

```
| 25%|null| null| 6| 3|
| 50%|null| null| 6| 4|
| 75%|null| null| 10| 9|
| max| C| West| 11| 12|
+-------+----+----------+-----------------+------------------+
```

The output displays the following key summary statistics for each column in the DataFrame. Note that for string variables like 'team' and 'conference', quantitative measures such as mean and stddev are naturally displayed as null:

count: The number of non-null values in the column.
mean: The arithmetic average value (for numeric columns only).
stddev: The standard deviation, indicating data dispersion.
min: The minimum value (or lexicographically smallest for strings).
25%: The first quartile (Q1).
50%: The 50th percentile, which is the median (Q2).
75%: The third quartile (Q3).
max: The maximum value (or lexicographically largest for strings).

Example 2: Calculating Specific Subsets of Summary Statistics (Targeted Metrics)

We utilize the following syntax to calculate only specific descriptive statistics--namely, the range and the interquartile measures--for all columns in the DataFrame. This is useful for focused analysis where the full default output is not required.

```
#calculate specific summary statistics for each column in DataFrame
df.summary('min', '25%', '50%', '75%', 'max').show()
```

```
+-------+----+----------+------+-------+
|summary|team|conference|points|assists|
+-------+----+----------+------+-------+
| min| A| East| 5| 2|
| 25%|null| null| 6| 3|
| 50%|null| null| 6| 4|
| 75%|null| null| 10| 9|
| max| C| West| 11| 12|
+-------+----+----------+------+-------+
```

Example 3: Summary Statistics Restricted to Numeric Columns (Best Practice)

To achieve the cleanest statistical report, we first identify the numeric columns by iterating through the DataFrame schema and then restrict the summary calculation using the DataFrame's **.select()** method. This approach ensures that all returned metrics are mathematically relevant.

#identify numeric columns in DataFrame
numeric_cols =

#calculate summary statistics for only the numeric columns
df.select(*numeric_cols).summary().show()

```
+-------+-----------------+------------------+
|summary| points| assists|
+-------+-----------------+------------------+
| count| 6| 6|
| mean|7.666666666666667| 5.666666666666667|
| stddev|2.422120283277993|3.9327683210007005|
| min| 5| 2|
| 25%| 6| 3|
| 50%| 6| 4|
| 75%| 10| 9|
| max| 11| 12|
+-------+-----------------+------------------+
```

Notice that summary statistics are displayed exclusively for the two numeric columns in the DataFrame - the **points** and **assists** columns. This exemplifies the best practice for accurate and clean statistical reporting within the PySpark environment.

Further Exploration and Resources

The .summary() function is a foundational tool, but PySpark offers many other ways to calculate granular statistics, such as using specific aggregate functions like `avg()`, `count()`, and `skewness()` directly within the .agg() method for customized group-by analysis.

It is highly recommended that users refer to the complete documentation for the PySpark **summary** function, which outlines advanced usage scenarios, parameter descriptions, and performance considerations for large-scale cluster computing. Mastering these basic techniques forms the bedrock for advanced data analysis and machine learning preparation using the Apache

Spark framework.

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