How to Calculate the Median of a PySpark Column

Authored by

stats writer

February 8, 2026

Calculating the median--the middle value of a dataset--is a fundamental task in data analysis. When working with large-scale datasets, tools like PySpark offer specialized functions for efficient computation across distributed environments. Traditionally, finding the median involves sorting the data and identifying the central element(s). If the total number of records is odd, the median is the single middle value. If the total number is even, the median is the average of the two middle values.

In PySpark, calculating the exact median can be resource-intensive for massive DataFrames because it requires shuffling and sorting the entire dataset. Fortunately, PySpark provides highly optimized functions, such as the standard `median` function, which is available via the pyspark.sql.functions module. This guide details two primary, straightforward methods for calculating the exact median of one or more columns within a PySpark DataFrame, followed by crucial considerations for data handling.

Calculate the Median of a Column in PySpark

To effectively calculate the median of a column within a PySpark DataFrame, two common patterns utilize the built-in aggregation functions provided by the library. These methods are essential for precise statistical analysis on distributed data.

The following sections demonstrate these methods using practical code examples.

Method 1: Calculating the Median for a Single Column using Aggregation

This technique is ideal when you need to find the median value for one specific column and retrieve it as a single scalar result, often useful for integration into further non-Spark processes or reporting. It uses the `agg` function combined with the `F.median()` function.

from pyspark.sql import functions as F

#calculate median of column named 'game1'
df.agg(F.median('game1')).collect()

The result of this operation is collected into a list of rows, and we access the first element of the first row `` to extract the scalar median value.

Method 2: Calculating Medians Across Multiple Columns Simultaneously

If your goal is to compute medians for several columns at once and display the results in a tabular format, the `select` function is often more convenient. This approach is highly efficient for

generating summary statistics viewable directly within the PySpark environment.

from pyspark.sql.functions import median

#calculate median for game1, game2 and game3 columns
df.select(median(df.game1), median(df.game2), median(df.game3)).show()

Before diving into the practical examples, we must first define the sample DataFrame that will be used for demonstration purposes throughout this tutorial.

PySpark Environment Setup and Sample Data Preparation

To ensure reproducibility, we start by initializing a SparkSession and defining a sample dataset representing team scores across three games. This DataFrame will serve as the input for both median calculation methods.

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 contains six rows, making it easy to manually verify the subsequent median calculations. Note the distinct scores in the `game1`, `game2`, and `game3` columns, which will yield different median results.

Example 1: Implementing Single Column Median Calculation

We apply the first method to calculate the median specifically for the column named **game1**. This calculation utilizes the aggregation function, which is optimized for summarizing data across all partitions.

from pyspark.sql import functions as F

#calculate median of column named 'game1'
df.agg(F.median('game1')).collect()

18.5

The output confirms that the median of the values in the **game1** column is **18.5**. This method is concise and highly effective when a single summary statistic is required for a designated column.

Detailed Verification of Single Column Median (game1)

To ensure the accuracy of the PySpark calculation, we can manually verify the median for the **game1** column. The calculation confirms that when the dataset has an even number of elements, the median is the arithmetic mean of the two central values.

First, we list and sort all values in the **game1** column:

10
14
15 (Middle value 1)
22 (Middle value 2)
25
30

Since there are six values (an even number), the median is the average of the two middle values,

15 and **22**. Calculating their average yields (15 + 22) / 2 = 37 / 2 = **18.5**. This confirms the accuracy of the PySpark function.

Example 2: Implementing Median Calculation for Multiple Columns

For scenarios requiring simultaneous calculation of medians across several columns--here, **game1**, **game2**, and **game3**--we use the `select` function alongside imported `median` function from `pyspark.sql.functions`.

from pyspark.sql.functions import median

#calculate median for game1, game2 and game3 columns
df.select(median(df.game1), median(df.game2), median(df.game3)).show()

```
+-------------+-------------+-------------+
|median(game1)|median(game2)|median(game3)|
+-------------+-------------+-------------+
| 18.5| 14.0| 13.0|
+-------------+-------------+-------------+
```

The resulting DataFrame clearly presents the calculated median for each specified column:

The median of values in the **game1** column is **18.5**.
The median of values in the **game2** column is **14.0**.
The median of values in the **game3** column is **13.0**.

Important Considerations: Handling Null Values

It is critical for data quality management to understand how PySpark handles missing data during statistical calculations. By default, standard aggregation functions in PySpark, including the `median` function, are designed to automatically ignore **null** values within the specified column.

This behavior means that rows containing nulls in the target column will be excluded from the median calculation entirely, ensuring that only valid numeric data contributes to the final result. If you require different handling, such as replacing nulls with zeros or mean values, you must perform explicit preprocessing steps using functions like `fillna()` before applying the median calculation.

Advanced Technique: Utilizing Approximate Quantiles for Large Datasets

While the standard `median` function provides an exact result, calculating it can necessitate

significant data shuffling across clusters, which slows down processing for extremely large datasets. For performance-critical applications where a slight loss of precision is acceptable, PySpark offers the `approxQuantile` function.

The median is mathematically equivalent to the 0.5 quantile. The `approxQuantile` function calculates quantiles with a defined relative error tolerance, significantly reducing the computational cost associated with sorting massive columns. This function requires specifying the column name, the quantile probability (0.5 for median), and a tolerance value. Using this method provides a highly scalable way to estimate the median quickly across petabytes of data, offering a powerful alternative to the exact methods shown above.

This guide has demonstrated the practical and efficient methods available in PySpark for calculating the exact median of columns, setting a strong foundation for robust statistical analysis in a distributed computing environment.