Calculate the Median by Group in PySpark

Authored by

stats writer

November 16, 2025

The Power of PySpark Aggregation

Analyzing large datasets efficiently often requires calculating statistical summaries across defined subsets of data. In the realm of big data processing, PySpark serves as a critical tool for performing these operations at scale. One common requirement in data analysis is calculating the Median--a robust measure of central tendency that is less susceptible to outliers compared to the mean. This guide details exactly how to utilize PySpark's powerful grouping and Aggregation functions to determine the median value broken down by one or multiple categories within a DataFrame.

We will explore two fundamental approaches to calculating the median by group. The first approach demonstrates simple grouping using a single categorical column, while the second expands this concept to handle complex hierarchies defined by multiple grouping columns. Mastering these methods is essential for anyone working with distributed data analysis using the PySpark framework.

Core PySpark Grouping Logic: The `groupBy().agg()` Pattern

To perform any group-based calculation in PySpark, we rely on the intuitive `groupBy()` method followed by the `agg()` method. The `groupBy()` function partitions the data based on the specified column(s), and `agg()` then applies the desired statistical function--in our case, `F.median()`--to the remaining columns within those partitions. This combination ensures that the calculation respects the defined categorical boundaries, resulting in highly informative summary statistics.

Below outlines the syntactic structure for calculating the median value by group in a PySpark DataFrame. Note the reliance on the pyspark.sql.functions module, typically imported as `F`, which provides the necessary aggregation functions for distributed computing environments.

Method 1: Calculating Median Grouped by a Single Column

import pyspark.sql.functions as F

#calculate median of 'points' grouped by 'team'
df.groupBy('team').agg(F.median('points')).show()

This first method is ideal when you need to understand the central tendency of a numerical variable (like 'points') strictly within the boundaries of a primary categorical variable (like 'team'). This provides a high-level overview of performance or metric distribution across major segments of the dataset. This simple aggregation is highly performant and often serves as a valuable first step in any exploratory data analysis.

Method 2: Calculating Median Grouped by Multiple Columns

import pyspark.sql.functions as F

#calculate median of 'points' grouped by 'team' and 'position'
df.groupBy('team', 'position').agg(F.median('points')).show()

When greater granularity is required, Method 2 extends the grouping functionality to multiple columns. By adding 'position' alongside 'team', the calculation is partitioned into significantly smaller, more specific buckets, allowing for fine-grained statistical analysis. This pattern is crucial for complex relational data where metrics vary widely based on compounded characteristics, offering a deeper dimensional view of the dataset.

Setting Up the Environment and Sample PySpark DataFrame

To demonstrate these methods practically, we must first establish a PySpark session and define a sample DataFrame. This DataFrame models basketball player statistics, including their assigned team, playing position, and the total points scored. This structured data allows us to clearly observe how the median calculation shifts based on the chosen grouping keys, providing a tangible example of PySpark's data manipulation capabilities.

We begin by importing the necessary components from `pyspark.sql`, initializing the `SparkSession`, defining our raw data structure (a list of lists), specifying the column schema, and finally constructing the DataFrame. This comprehensive setup ensures replicability and provides a strong foundation for the subsequent aggregation examples. The detailed output of the DataFrame is also provided for reference below, showing the initial state of the dataset.

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|position|points|
+----+--------+------+
| A| Guard| 11|
| A| Guard| 8|
| A| Forward| 22|
| A| Forward| 22|
| B| Guard| 14|
| B| Guard| 14|
| B| Guard| 13|
| B| Forward| 7|
| C| Guard| 8|
| C| Forward| 5|
+----+--------+------+
```

The resulting DataFrame, `df`, contains ten rows of player data with three descriptive columns. We are specifically interested in calculating the central score distribution--the Median of the `points` column--when segregated by the `team` and `position` attributes. This initial setup is the foundational step before proceeding to the specific Aggregation examples, demonstrating the input data structure before distributed processing begins.

Example 1: Calculating Median Grouped by a Single Column

Our first example focuses on calculating the team-wide median points. This involves grouping the entire dataset based solely on the unique entries in the `team` column (A, B, or C). We apply the `F.median()` function from pyspark.sql.functions to the `points` column, producing a summary DataFrame where each row represents a team and its corresponding median point score.

This methodology allows data engineers and analysts to quickly identify which groups exhibit a higher or lower typical performance level, filtering out the noise that might be introduced by outlier scoring events. The syntax is concise yet powerful, leveraging PySpark's optimized execution

engine for distributed computation across the worker nodes. We use the `show()` action to materialize and display the result.

import pyspark.sql.functions as F

#calculate median of 'points' grouped by 'team'
df.groupBy('team').agg(F.median('points')).show()

+----+--------------+
|team|median(points)|
+----+--------------+
A	16.5
B	13.5
C	6.5
+----+--------------+

Analyzing the Results of Single-Column Grouping

The output table reveals the centralized scoring performance for each team. It is essential to understand how these median values are derived. For Team A, the point scores are (11, 8, 22, 22). When sorted, these are (8, 11, 22, 22). Since there is an even number of observations, the median is the average of the two middle values (11 and 22), resulting in **16.5**.

Similarly, the median for Team B is calculated from scores (14, 14, 13, 7). Sorted: (7, 13, 14, 14). The average of the middle values (13 and 14) yields **13.5**. Finally, Team C scores are (8, 5). The average of these two values is **6.5**. This confirms that the Median function in PySpark correctly handles both odd and even counts within the groups, calculating the 50th percentile of the numerical data within each distinct partition.

From the resulting summary DataFrame, we can clearly deduce the following central tendencies for scoring performance based on team affiliation:

The median points value for players on **team A** is **16.5**. This suggests a higher central performance compared to the other teams in the sample dataset.
The median points value for players on **team B** is **13.5**.
The median points value for players on **team C** is **6.5**. Team C demonstrates the lowest central scoring output among the three teams, suggesting a need for further investigation into their scoring dynamics.

Example 2: Calculating Median Grouped by Multiple Columns

In real-world data science scenarios, metrics often depend on interactions between multiple categorical variables. For instance, a player's performance likely varies based not only on their team but also on their assigned position. Example 2 demonstrates how to calculate the median points by creating composite groups using both the `team` and the `position` columns simultaneously, providing a more detailed breakdown of performance metrics.

By specifying `df.groupBy('team', 'position')`, we instruct PySpark to segment the data into six distinct groups (A-Guard, A-Forward, B-Guard, B-Forward, C-Guard, C-Forward). The subsequent application of `F.median('points')` performs the statistical calculation independently within each of these finely defined partitions. This provides invaluable context, showing whether scoring differences are team-specific, position-specific, or a combination of both factors, which is often the case in observational data.

import pyspark.sql.functions as F

#calculate median of 'points' grouped by 'team' and 'position'
df.groupBy('team', 'position').agg(F.median('points')).show()

+----+--------+--------------+
|team|position|median(points)|
+----+--------+--------------+
A	Guard	9.5
A	Forward	22.0
B	Guard	14.0
B	Forward	7.0
C	Forward	5.0
C	Guard	8.0
+----+--------+--------------+

Interpreting Multi-Dimensional Aggregation

The results of the multi-column Aggregation offer a much deeper insight into the data distribution. We can now see nuances that were obscured in the simple team-level aggregation. For instance, while Team A's overall median was 16.5, this masks a significant performance difference between their positions. The Guards (scores 11, 8) have a median of 9.5, whereas the Forwards (scores 22, 22) have a perfect median of 22.0. This critical distinction informs coaching or resource allocation decisions, highlighting high-performing subsets of the data.

It is important to note the calculation logic for smaller groups. When a group contains only one data point (e.g., Team B Forward with a score of 7, or Team C Forward with a score of 5), the Median is simply that single value, as it is the central tendency of a set of one. When calculating the median of an odd-sized dataset (e.g., Team B Guards: 14, 14, 13, sorted: 13, 14, 14), the median is the middle value, **14.0**. This demonstrates the robustness of the `F.median()` function across varying group sizes.

Detailed interpretation of the output reveals the following specific performance metrics:

The median points value for **Guards on team A** is **9.5**, which is lower than the team average, suggesting that the Forwards are driving Team A's high overall median.
The median points value for **Forwards on team A** is **22.0**, showcasing high consistency in scoring for that specific position within that team.
The median points value for **Guards on team B** is **14.0**.
The median points value for **Forwards on team B** is **7.0**.
The median points value for **Forwards on team C** is **5.0**.
The median points value for **Guards on team C** is **8.0**.

Conclusion: Advanced PySpark Aggregation Techniques

Calculating the median by group in PySpark is a straightforward yet essential operation for deep data profiling and exploratory analysis. By effectively using the `groupBy()` and `agg()` combination, along with functions provided by pyspark.sql.functions, analysts can quickly generate accurate, distributed statistical summaries, transforming raw data into actionable insights at massive scale.

Furthermore, while this guide focused on `F.median()`, the same Aggregation framework can be applied using other statistical functions like `F.min()`, `F.max()`, `F.avg()`, or `F.stddev()`. Understanding how to define the appropriate grouping key(s) is paramount to unlocking the full analytical potential of big data using the PySpark DataFrame API. Always prioritize using the most specific grouping necessary to answer the analytical question at hand, whether it requires one column or many, to ensure the derived statistics accurately reflect the underlying data distribution.