How to Calculate Quartiles in PySpark: A Step-by-Step Guide

Authored by

stats writer

February 6, 2026

Calculating quartiles within the PySpark framework is a fundamental step in exploratory data analysis, enabling data scientists to understand the underlying distribution of large datasets. This process involves determining specific threshold values that partition the data into four segments, known statistically as the 25th, 50th, and 75th percentiles. PySpark facilitates this through powerful statistical functions like `approxQuantile` or `percentile_approx`. These functions are designed to operate efficiently on massive data volumes utilizing the distributed processing power of the Spark SQL engine, taking a target column (or list of columns) and the desired quantile fractions as input parameters. Integrating these statistical tools into your PySpark workflow allows for rapid calculation of quartiles, yielding immediate and valuable insights into the spread, central tendency, and skewness of your data.

Calculate Quartiles in PySpark (With Example)

Deconstructing Statistical Quartiles

In the realm of statistics, **quartiles** represent critical measures of position that effectively divide a numerical dataset into four segments, each containing 25% of the total observations. Understanding these division points is essential for constructing box plots, detecting outliers, and summarizing the shape of a distribution without relying solely on the mean, which can be heavily influenced by extreme values. The ability to perform this calculation efficiently across large-scale data systems like PySpark provides analysts with immediate, actionable insights into data variability.

When conducting distribution analysis, we primarily focus on three distinct quartile values that define the boundaries between these four segments. These values are mathematically equivalent to specific percentiles, offering a standardized way to describe data spread.

The quartiles we are typically interested in and their corresponding percentiles are:

First Quartile (Q1): This is the value situated at the 25th percentile. It marks the point where 25% of the data falls below it.

Second Quartile (Q2): Often referred to as the **median**, this value resides at the 50th percentile. It represents the central point of the dataset, dividing the data exactly in half.

Third Quartile (Q3): Located at the 75th percentile, this value indicates the point below which 75% of the observations lie.

Choosing the Right PySpark Function: approxQuantile

When dealing with vast datasets managed by Spark, calculating exact quantiles can be

computationally prohibitive due to the need for precise sorting across distributed partitions. To address this challenge, PySpark provides the `approxQuantile` function. As the name suggests, this function calculates approximate quantiles, prioritizing speed and resource efficiency while maintaining a high degree of statistical accuracy. This method is the preferred standard for calculating quartiles (and any percentile) in big data environments.

The `approxQuantile` function implements an algorithm that is scalable and optimized for parallel execution within the distributed architecture of PySpark. This approach makes it possible to determine reliable quartile estimates on petabytes of data in minutes, rather than hours or days. The output of this function is a list of approximate values corresponding to the requested quantile probabilities.

It is important to differentiate `approxQuantile` from methods used in traditional statistical software. Since Spark operates on distributed data, the exact calculation might require immense shuffling and memory. By accepting a small, tunable margin of error, `approxQuantile` provides the necessary balance between statistical precision and operational performance required for large-scale data processing.

Understanding the approxQuantile Syntax and Parameters

To successfully calculate the quartiles for a column within a PySpark DataFrame, you must utilize the `approxQuantile` method, specifying three crucial parameters. This syntax is concise yet powerful, leveraging the column-based operations inherent in the Spark SQL engine.

The required syntax for calculating quartiles is structured as follows, where the function is called on the DataFrame object itself:

#calculate quartiles of 'points' column
df.approxQuantile('points', , 0)

The three primary arguments passed to the `approxQuantile` function are:

Column Name (String): This specifies the name of the column in the DataFrame for which the quartiles are to be calculated. In the example above, this is `'points'`.

Quantile Probabilities (List): This is a list of floating-point values representing the desired quantile points. Since quartiles correspond to the 25th, 50th, and 75th percentiles, the list must contain .

Relative Error (Float): This parameter controls the precision of the approximation. It specifies the maximum allowable relative error tolerance. Setting this value to `0` (as shown in the example)

requests the calculation of exact quantiles, which can be resource-intensive, but often acceptable for smaller datasets or when absolute precision is non-negotiable. For very large datasets, a small error tolerance (e.g., `0.01`) significantly speeds up computation time.

Practical Example: Setting Up the PySpark Environment

To demonstrate the practical application of calculating quartiles, we will first establish a sample PySpark DataFrame. This example simulates real-world data containing performance metrics, specifically focusing on points scored by various professional basketball teams. The setup involves initializing a Spark session and defining the structure and contents of our dataset.

The following code snippet illustrates how to import necessary libraries, create the data structure, define column schema, and generate the DataFrame, which is the foundational structure for analysis in PySpark:

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|points|
+-------+------+
| Mavs| 18|

Nets	33
Lakers	12
Kings	15
Hawks	19
Wizards	24
Magic	28
Jazz	40
Thunder	24
Spurs	13
+-------+------+

This resulting DataFrame, `df`, contains ten records, each providing a team name and the corresponding points scored. Our objective is to calculate the three primary **quartiles** for the `points` column to understand the distribution of scoring across these teams.

Executing the Quartile Calculation

Once the DataFrame is initialized, the calculation phase is straightforward, utilizing the specific syntax of the `approxQuantile` function. We specify the column of interest (`'points'`), the desired quartiles as decimal probabilities (), and for this introductory example, a relative error of `0` to ensure an exact calculation.

Applying the function executes the distributed computation across the Spark cluster. The function returns a Python list containing the calculated quartile values in the order they were requested (Q1, Q2, Q3).

We execute the following code block to calculate the quartiles for the **points** column:

#calculate quartiles of 'points' column
df.approxQuantile('points', , 0)

The resulting list provides the three critical measures that define the data spread for the scores. These values are highly effective summaries compared to merely observing the raw data list.

Interpreting the Results and Data Distribution

The numerical output obtained from the `approxQuantile` function provides immediate statistical context for the `points` column. By mapping the output values back to their corresponding quartile definitions, we can derive meaningful conclusions about the central tendency and spread of the

scoring data.

From the calculated output , we can interpret the following key findings:

The **First Quartile (Q1)** is located at **15.0**. This means that 25% of the observed basketball scores are 15 points or less.

The **Second Quartile (Q2)**, or the median, is located at **19.0**. This indicates that half of the team scores fall below 19 points, providing a robust measure of central performance unaffected by extreme high scores.

The **Third Quartile (Q3)** is located at **28.0**. This point signifies that 75% of the team scores are 28 points or less, and conversely, 25% of the teams scored higher than 28 points.

By knowing just these three quartile values, analysts gain a solid understanding of the distribution of values in the **points** column. For instance, the Interquartile Range (IQR), calculated as Q3 minus Q1 (28.0 - 15.0 = 13.0), provides a measure of statistical dispersion, showing the range over which the central 50% of the data lies. This is invaluable for comparative analysis or establishing benchmarks.

Considerations Regarding Relative Error

While the previous example used a relative error of `0` for simplicity and exactness on a small dataset, it is crucial to understand the implications of the relative error parameter in production-level big data analytics using PySpark. When processing billions of records, requesting exact quantiles (error = 0) may lead to performance bottlenecks and substantial memory demands due to data shuffling requirements across the cluster.

The primary strength of the `approxQuantile` function lies in its ability to provide accurate results quickly when a tolerable margin of error is accepted. Setting the relative error (e.g., `0.001`) means that the result returned is guaranteed to be within that proportion of the true quartile value. For most large-scale analytical tasks, a small approximation error is a worthwhile trade-off for significant gains in computational speed and efficiency.

Analysts should always select the smallest relative error that meets their analytical precision needs while balancing the performance limitations of their cluster configuration. This functional flexibility ensures that PySpark remains a powerful tool for statistically rigorous analysis in an environment where volume and velocity are paramount constraints.

Summary and Further Analytical Steps

Calculating **quartiles** in PySpark is highly efficient using the `approxQuantile` function on a

DataFrame. This method rapidly delivers the 25th, 50th (median), and 75th percentiles, providing a quick, robust summary of data distribution. By correctly specifying the column name, the list of quantile probabilities, and an appropriate relative error tolerance, data scientists can unlock crucial descriptive statistics for their large datasets.

Note: You can find the complete documentation for the PySpark **approxQuantile** function here for advanced usage scenarios, including calculating quantiles for multiple columns simultaneously.

The following tutorials explain how to perform other common tasks in PySpark, building upon this foundational knowledge of distribution analysis:

How to calculate standard deviation in PySpark.

Methods for filtering and subsetting PySpark DataFrames.

Using window functions for rolling averages in Spark.