How to Calculate Percentiles in PySpark with `approxQuantile()`

Authored by

stats writer

February 9, 2026

How to Calculate Percentiles in PySpark (With Detailed Examples)

Introduction to Percentile Calculation in PySpark

Calculating statistical measures like the Percentile is essential when analyzing large datasets, especially those managed within distributed computing frameworks like PySpark. Percentiles provide critical insight into the distribution of data, allowing analysts to identify cutoff points, understand data skewness, and evaluate performance relative to a defined population. The method chosen in Spark often depends on whether the requirement is for an exact result or a performant approximation suitable for massive-scale operations.

For extremely large data volumes where speed is paramount, **PySpark** provides the `approxQuantile()` function, accessible via the pyspark.sql.functions module. This function is highly efficient as it uses an approximation algorithm, requiring three parameters: the column name, the desired quantile values (as a list of decimals), and the relative error tolerance (e.g., 0.01 for 1% tolerance). It is an excellent choice for exploratory data analysis or when generating features for machine learning models that tolerate minor statistical deviations.

However, when precision is critical, such as in regulatory reporting or detailed statistical studies, analysts typically turn to the exact percentile calculation method. This is achieved by invoking the SQL function `percentile` directly through the **DataFrame** API using `F.expr()`. This approach guarantees the precise, interpolated quantile value, albeit sometimes at the cost of increased computation time compared to the approximate method. Our focus in the following examples will be on demonstrating this highly accurate, exact calculation method.

Understanding Quantiles and the Need for Exact Calculation

A Percentile defines the boundary value below which a specified percentage of observations fall. For instance, the 50th percentile is the median. Calculating these quantiles accurately is paramount for benchmarks; imagine calculating the 90th percentile response time for a critical application--an approximation might mask a genuine performance bottleneck that only exact calculation can reveal.

Since PySpark manages data across a distributed cluster, calculating exact quantiles requires careful orchestration to ensure all data points are considered and sorted correctly before interpolation occurs. The Spark SQL function `percentile` handles this complexity internally, providing a robust and reliable way to derive these measures from vast datasets.

The standard syntax for obtaining exact percentiles involves using the `.agg()` method on the

DataFrame, passing the percentile calculation as a Spark SQL expression. This flexibility allows us to integrate complex statistical operations directly into our data transformation pipelines with ease and clarity.

The Two Primary PySpark Percentile Methods

When choosing a method for calculating quantiles in **PySpark**, developers must weigh speed against accuracy.

The Approximate Method: Utilizes the approxQuantile() function. This method is highly scalable and fast, designed to handle columns with billions of rows by guaranteeing that the returned value is close to the true quantile within a specified error margin (e.g., 0.01). It is essential for initial data exploration or tasks where absolute statistical accuracy is not the primary constraint.

The Exact Method: Utilizes the SQL expression `percentile(column, array(quantiles))`. This method provides the mathematically precise quantile value. It is preferred for financial calculations, detailed statistical reporting, or any scenario demanding zero deviation from the true distribution. Because this method requires sorting and handling of potentially large intermediate results, its performance profile differs significantly from the approximate method.

The examples provided below exclusively use the exact method via `F.expr('percentile(...)')` because it offers the precision required for rigorous data analysis and can be easily adapted for complex grouped aggregations.

Practical Setup: Initializing Spark and Creating the DataFrame

Before diving into the percentile calculations, we must first establish the necessary environment and create a sample **DataFrame**. We use a SparkSession to interact with the Spark cluster and define a small dataset concerning basketball player statistics (points scored) across two teams. This setup mimics the common workflow of data professionals using PySpark.

We import `SparkSession` to manage the distributed environment and use the defined data and column schema to construct our initial **DataFrame**, which will be the subject of our statistical queries.

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| 8|
| A| Guard| 15|
| A| Forward| 22|
| A| Forward| 22|
| A| Guard| 32|
| B| Guard| 9|
| B| Guard| 15|
| B| Forward| 28|
| B| Guard| 31|
| B| Forward| 40|
+----+--------+------+
```


Method 1: Calculating Percentiles for a Single Column

When analyzing the overall distribution of a variable, such as all player points regardless of team, we calculate the percentiles across the entire column. This establishes a baseline understanding of the central tendency and spread of the data. We use the `.agg()` transformation, which is designed to compute aggregate statistics over all rows of the **DataFrame**.

To find the 25th Percentile (Q1) for the **points** column, we use `F.expr('percentile(points,`

`array(0.25))')`. Note the necessity of wrapping the quantile value (0.25) within an array, as the function is designed to calculate multiple quantiles simultaneously. We then extract the first element of the resulting array using and assign a descriptive alias. This computation is robust, providing an exact measure of the value separating the bottom quarter of the scores from the rest.

This technique is vital for setting universal benchmarks; for instance, determining the minimum acceptable score that falls within the bottom quartile of historical performance data.

import pyspark.sql.functions as F

#calculate 25th percentile for 'points' column
df.agg(F.expr('percentile(points, array(0.25))').alias('%25')).show()

Example 1: Demonstrating Single Column Percentile Calculation

Executing the syntax defined in Method 1 against our sample player data provides the overall 25th percentile for all players combined. This result gives us a standardized measure across the entire dataset without regard to team affiliation.

import pyspark.sql.functions as F

#calculate 25th percentile for 'points' column
df.agg(F.expr('percentile(points, array(0.25))').alias('%25')).show()

+----+
| %25|
+----+
|15.0|
+----+

The calculation yields **15.0**. This means that 25% of all players in our sample scored 15 points or fewer. The calculation is based on the full sorted list of points: . Since the total count is 10, the 25th percentile falls exactly at the third position (10 * 0.25 = 2.5, which typically rounds up or interpolates to the third element in standard statistical software). This straightforward aggregation demonstrates the power of using pyspark.sql.functions for quick statistical insights.

Method 2: Calculating Percentiles Grouped by Another Column

A more advanced and frequently used analytical task is calculating percentiles relative to specific subgroups. If we need to compare performance distributions between Team A and Team B, we must calculate the percentiles grouped by the **team** column. This is facilitated by chaining the

`.groupby()` method before the `.agg()` action.

When calculating grouped percentiles, **PySpark** performs the necessary data partitioning and sorting operations specific to each group key. We can calculate multiple quantiles (25th, 50th, and 75th percentiles) simultaneously for each team within a single aggregation step. This is highly efficient as it minimizes data shuffling across the cluster. We simply list the required `F.expr('percentile(...)')` calls, each with a distinct quantile value (0.25, 0.50, 0.75) and alias, inside the `.agg()` function.

This grouped calculation is indispensable for comparative analytics, allowing data scientists to quickly identify how the statistical distribution of a metric (like points) varies across different categories (like teams, regions, or product types).

import pyspark.sql.functions as F

#calculate 25th, 50th and 75th percentile of 'points', grouped by 'team'
df_new = df.groupby('team').agg(F.expr('percentile(points, array(0.25))').alias('%25'),
F.expr('percentile(points, array(0.50))').alias('%50'),
F.expr('percentile(points, array(0.75))').alias('%75'))

Example 2: Demonstrating Grouped Percentile Calculation

Executing the grouped aggregation demonstrates how PySpark processes the statistical computation independently for each team. Since both Team A and Team B have five observations each, the percentile calculation for each team is distinct and highly accurate.

The resulting **DataFrame**, `df_new`, provides a side-by-side comparison of the quartiles for each team, offering immediate insight into their respective scoring distributions.

import pyspark.sql.functions as F

#calculate 25th, 50th and 75th percentile of 'points', grouped by 'team'
df_new = df.groupby('team').agg(F.expr('percentile(points, array(0.25))').alias('%25'),
F.expr('percentile(points, array(0.50))').alias('%50'),
F.expr('percentile(points, array(0.75))').alias('%75'))

#view new DataFrame
df_new.show()

+----+----+----+----+
|team| %25| %50| %75|
+----+----+----+----+

| A|15.0|22.0|22.0|
| B|15.0|28.0|31.0|
+----+----+----+----+

Detailed analysis of the output confirms the distributional characteristics of each team:

For **Team A**: The 50th Percentile (median) is **22.0**, indicating a central tendency of scoring around that mark. The 75th percentile is also **22.0**, which, due to the discrete nature of the data points , shows that 75% of players scored 22 points or less.

For **Team B**: The median is significantly higher at **28.0**. The 75th percentile is **31.0**. This comparison reveals that while both teams share the same 25th percentile (15.0), Team B exhibits a higher overall scoring potential and a wider spread in the upper half of its point distribution compared to Team A.

Conclusion and Further PySpark Resources

The ability to accurately and efficiently compute quantiles is a cornerstone of statistical analysis in a big data environment. Whether deploying the highly scalable approxQuantile() for speed or relying on the exact SQL expression method for precision, **PySpark** offers robust mechanisms for data profiling and distribution analysis.

By mastering the combination of `.groupby()` and `.agg()` with powerful functions imported from pyspark.sql.functions, analysts can derive sophisticated statistical summaries that drive critical business intelligence and scientific research. These techniques are fundamental for tasks ranging from defining performance thresholds to informing complex machine learning feature engineering.

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