Learning Quartiles with PySpark: A Step-by-Step Guide

Authored by

Mohammed looti

November 11, 2025

Understanding Quartiles in Statistical Analysis

In the realm of statistics and data analysis, quartiles are fundamental descriptive metrics. They serve as crucial markers, partitioning a sorted dataset into four equal segments, with each segment containing 25% of the data points. Understanding quartiles allows analysts to quickly grasp the spread, skewness, and central tendency of a numerical distribution, providing a robust summary beyond simple averages or standard deviations.

Specifically, when examining any data distribution, three key quartiles are utilized to define these boundaries:

First Quartile (**Q1**): This value marks the 25th percentile of the data. One-quarter of the observations fall below this point, defining the lower boundary of the central 50% of the distribution.
Second Quartile (**Q2**): This is equivalent to the 50th percentile, also known universally as the **median**. It is the central value that divides the data exactly in half.
Third Quartile (**Q3**): This value represents the 75th quartile. Three-quarters of the observations fall below this point, forming the upper boundary of the central 50%.

The difference between Q3 and Q1 is known as the **Interquartile Range (IQR)**, which is a key measure of statistical dispersion. Utilizing quartiles is often preferred over measures dependent on the mean, especially when dealing with data distributions that are significantly skewed or contain numerous outliers, as the median (Q2) and the quartiles are resistant to extreme values.

The Role of PySpark in Distributed Data Quantification

When statistical analysis moves from small, manageable collections to massive, petabyte-scale datasets, traditional statistical software often fails due to memory and processing limitations. This is precisely where PySpark, the Python API for Apache Spark, becomes an essential tool. PySpark is specifically engineered for distributed computing, allowing complex calculations like quartile determination to be executed efficiently across a cluster of machines.

Calculating exact quantiles (quartiles) on a massive, distributed DataFrame can still be computationally expensive, as it often requires sorting the entire dataset, which necessitates significant data shuffling across the cluster. To mitigate this computational overhead while maintaining high accuracy, PySpark provides the `approxQuantile` function. This function uses advanced approximation algorithms that significantly reduce processing time and resources compared to finding the exact values, making it the standard approach in big data analytics.

The efficiency of the `approxQuantile` method ensures that data scientists can calculate descriptive statistics almost instantaneously, even for extremely large data volumes. This function is fundamental to data exploration in a distributed environment, allowing for rapid identification of

central tendencies and data spread without sacrificing crucial time waiting for exhaustive, exact calculations. The following section details the specific syntax required to leverage this powerful function.

Syntax and Parameters of the `approxQuantile` Function

To calculate the quartiles for a numerical column within a PySpark DataFrame, we call the `approxQuantile` method directly on the DataFrame object. This function accepts three primary parameters, each critical for defining the scope and precision of the calculation:

The general structure of the function call is as follows:

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

The three arguments passed to the function are:

Column Name (String): The first argument is a string specifying the name of the column for which the quantiles must be calculated (e.g., `'points'`).
Probabilities (List of Floats): This is a list defining the specific quantiles or percentile probabilities we are interested in. For standard quartiles (Q1, Q2, Q3), the list must contain .
Relative Error (Float): This parameter, often denoted as epsilon (ε), dictates the precision of the approximation. A value of `0` signifies that the calculation should attempt to find the exact quantiles, which is recommended only for smaller datasets or when absolute precision is non-negotiable. For larger datasets, setting a small positive value (e.g., `0.01`) speeds up computation dramatically while maintaining acceptable accuracy, sacrificing a minimal degree of precision for immense performance gains.

For the purposes of this demonstration, we will set the relative error parameter to `0` to achieve exact quartiles for our small sample data, ensuring that the results are perfectly accurate and easily verifiable.

Practical Example: Setting Up the PySpark Environment

To illustrate the application of the `approxQuantile` function, let us consider a practical example involving a PySpark DataFrame. Suppose we have a dataset containing simulated performance data--specifically, the points scored by various professional basketball teams across a series of games. This example demonstrates the initial steps required to define the Spark environment and structure the data before performing the statistical calculation.

The first step involves initializing a Spark Session, which is the entry point for all Spark

functionality. Following this, we define the sample data structure and column names, and finally, we use `spark.createDataFrame` to transform this standard Python data into a distributed PySpark DataFrame, ready for processing. The following code block details this setup:

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

+-------+------+

The resulting DataFrame, `df`, now holds ten records, each representing a team and its corresponding score in the `points` column. With the data structure validated and the Spark environment configured, we are prepared to proceed with the quantile calculation for the numerical `points` column.

Calculating and Interpreting Quartile Results

Having established our DataFrame, we can now apply the `approxQuantile` function to the `points` column. We specify the probabilities 0.25, 0.5, and 0.75 to retrieve the first, second, and third quartiles, respectively, setting the relative error to zero for maximum precision in this small dataset.

The execution of the function yields a list of floating-point numbers corresponding to the calculated quartile boundaries in ascending order. This list provides the three key values necessary to summarize the distribution of the scores.

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

From the output list , we can derive the following essential insights into the distribution of points:

The first quartile (**Q1**) is located at **15.0**. This means 25% of the teams scored 15 points or fewer.
The second quartile (**Q2**), or median, is located at **19.0**. This indicates that half of the teams scored 19 points or fewer, and half scored more than 19 points.
The third quartile (**Q3**) is located at **28.0**. This implies that 75% of the teams scored 28 points or fewer.

By knowing only these three values, we gain a comprehensive understanding of the central tendency and the overall spread of values in the **points** column. For instance, the Interquartile Range (IQR) is 28.0 - 15.0 = 13.0, signifying that the central 50% of the scores are spread across a range of 13 points. This rapid statistical summary is highly valuable for initial data exploration and identifying potential outliers.

Limitations and Considerations for Approximate Quantiles

While the `approxQuantile` function is a cornerstone of scalable data analysis in PySpark, users must be mindful of the inherent trade-off between speed and accuracy, which is governed by the `relativeError` parameter. Choosing a non-zero error tolerance (e.g., 0.01) significantly reduces

computation time, particularly for massive datasets, but introduces a small, bounded deviation from the true quantile value.

For applications demanding extremely high precision, such as financial modeling or regulatory reporting, setting the relative error to `0` might be necessary, even if it results in longer processing times. Conversely, for large-scale exploratory data analysis (EDA) where identifying general trends and distributions is the priority, accepting a minor approximation error is usually the optimal strategy to maximize efficiency and resource utilization. Data practitioners should always select the error parameter based on the specific constraints and objectives of their analytical task.

It is worth reiterating that the `approxQuantile` function is designed specifically for the distributed nature of Spark. For users seeking further technical documentation regarding the implementation details, performance characteristics, and algorithmic foundation of this critical function, the official documentation serves as the most authoritative source.

Additional Resources

For those looking to expand their knowledge of PySpark functionalities and perform other common data analysis tasks, the following resources are recommended:

Note: You can find the complete documentation for the PySpark **approxQuantile** function here.

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