Calculate Quartiles in Pandas (With Example)

Authored by

Mohammed looti

November 16, 2025

Introduction: The Significance of Quartiles in Data Analysis

In the realm of statistics and data science, gaining a comprehensive understanding of the underlying data distribution is fundamental for robust analysis. While measures like the mean provide insight into the central tendency, they often fail to capture the spread, symmetry, and potential existence of outliers within a dataset. This is where **Quartiles** become indispensable tools. Quartiles are measures of position that delineate the distribution by dividing the ordered data into four equal parts, offering a clear view of how values are clustered around the median.

The application of quartiles extends far beyond simple descriptive reporting. They are crucial for performing detailed exploratory data analysis (EDA), allowing analysts to quickly gauge data variability and identify critical thresholds. Understanding the quartile values helps in diagnosing the shape of the distribution, such as identifying if the data is skewed positively or negatively. Furthermore, quartiles form the backbone of the Interquartile Range (IQR), a metric highly valued for its robustness against extreme values, which is commonly used for outlier detection.

Every quartile corresponds precisely to a specific percentile marker. These values are the demarcation points that partition the data into the four segments. When analyzing any numerical distribution, we are typically interested in three primary quartile values which define the boundaries of these four sections:

First Quartile (**Q1**): Also known as the lower quartile, this is the value located at the 25th percentile. Twenty-five percent of the data values fall below **Q1**, meaning it marks the upper boundary of the lowest quarter of the data.
Second Quartile (**Q2**): This value represents the 50th percentile, which is universally equivalent to the median of the entire dataset. It splits the data exactly in half.
Third Quartile (**Q3**): Known as the upper quartile, this is the value located at the 75th percentile. Seventy-five percent of the data values fall below **Q3**, marking the point where the top 25% of the values begin.

Utilizing Pandas for Efficient Quartile Calculation

Working with large or complex datasets requires powerful, optimized tools. The Pandas library, a cornerstone of the Python data science ecosystem, is meticulously designed to handle such calculations efficiently. For determining quartiles, the most critical function provided by Pandas is the highly versatile `.quantile()` method. This method acts as a universal tool for calculating any percentile rank, making it ideal for pinpointing the standard quartile values.

The `.quantile()` method can be applied flexibly, either to an entire DataFrame, calculating quartiles for every numerical column simultaneously, or to an individual Series (a specific column), allowing for highly focused analysis. The core input parameter required is the `q` argument. This

argument defines the specific quantile or list of quantiles (represented as floats between 0 and 1) that the user wishes to calculate. For standard quartiles, we typically pass the list .

In practice, data analysts often face two primary scenarios when calculating these descriptive statistics within a Pandas environment. The first approach targets a specific feature for granular detail, while the second provides a rapid, comprehensive summary across the entire numerical landscape of the data. Both leverage the power of `.quantile()` but differ significantly in their application syntax and resulting output structure.

The two fundamental approaches available for calculating these descriptive statistics are structured around where the `.quantile()` method is called. Notice how the first method is chained to a specific column selection, whereas the second operates globally on the DataFrame object itself:

Method 1: Calculate Quartiles for a Specific Column (Pandas Series)

df.quantile()

Method 2: Calculate Quartiles for Each Numeric Column (Entire DataFrame)

df.quantile(q=, axis=0, numeric_only=True)

Setting Up the Sample Pandas DataFrame

To properly demonstrate the practical implementation of these two powerful methods, we must first initialize a suitable sample DataFrame. This example simulates a small collection of sports statistics, which is ideal because it contains both categorical data (team names) and quantitative data (points and assists). This mixed data structure allows us to explicitly illustrate how Pandas correctly handles quartile calculation by selectively operating only on the numerical features.

The following standard Python code snippet utilizes the core functionality of the Pandas library to generate the required data structure. We first import the library, define the specific data points for ten hypothetical teams, and then construct the DataFrame. Displaying the resulting structure immediately confirms the data is correctly loaded and ready for statistical analysis, preventing potential errors in subsequent steps.

import pandas as pd

# Create the sample DataFrame structure
df = pd.DataFrame({'team': ,
'points': ,

'assists': })

# View the resulting DataFrame to verify structure
print(df)

team points assists
0 A 12 2
1 B 14 2
2 C 14 3
3 D 16 3
4 E 24 4
5 F 26 6
6 G 28 7
7 H 30 8
8 I 31 10
9 J 35 15

The resulting DataFrame, named `df`, now holds ten observations and three variables. The 'points' column, representing the scores achieved, is a continuous numerical feature that will serve as the primary target for our single-column analysis in the subsequent section. The 'assists' column, another quantitative feature, will be included when we execute the comprehensive, multi-column analysis.

Example 1: Calculating Quartiles for a Specific Column (Series)

Often, analytical tasks require a deep focus on the distribution characteristics of just one variable, isolating its spread from the rest of the dataset. To calculate the quartiles for the 'points' column, we apply the `.quantile()` method directly to the selected Series object. This targeted approach, which treats the column as an independent Pandas Series, is highly memory-efficient and provides clear, focused descriptive statistics for that specific feature.

To retrieve the three standard quartiles (Q1, Q2, and Q3), we pass a list containing the desired percentile thresholds--0.25, 0.50, and 0.75--as the argument to the function. It is important to remember that the output will itself be a Pandas Series. In this output, the index labels correspond to the quantile level requested (e.g., 0.25), and the values represent the calculated score at that specific percentile rank.

 # Calculate quartiles specifically for the points column
df.quantile()

0.25 14.5

0.50 25.0
0.75 29.5
Name: points, dtype: float64

The results yield a precise statistical summary of the 'points' distribution. These values define the exact cut-off points that segment the teams' scores into four equal bins, based on widely accepted statistical methodology:

The first quartile (**Q1**) is located at **14.5**. This signifies that 25% of the teams achieved a score of 14.5 points or less.
The second quartile (**Q2**), which is the statistical median, is **25.0**. This means exactly half of the teams scored below 25 points, and half scored above.
The third quartile (**Q3**) is located at **29.5**. This indicates that 75% of the teams scored 29.5 points or less. Conversely, the top 25% of performers scored above 29.5 points.

By calculating these three measures, analysts obtain a robust snapshot of the distribution's center and spread. For example, calculating the Interquartile Range (IQR) as Q3 minus Q1 (29.5 - 14.5 = 15) provides the range covering the middle 50% of the data. This measure is crucial for identifying moderate and extreme outliers and forms the mathematical foundation for powerful visualization tools like box plots, enabling clearer interpretation of data dispersion.

Example 2: Comprehensive Quartile Calculation Across the DataFrame

When performing initial exploratory data analysis on a multivariate dataset, efficiency dictates that we calculate distributional characteristics simultaneously for all relevant features. To achieve this holistic view, we apply the `.quantile()` method directly to the entire `df` object itself. This instructs Pandas to iterate through every column and perform the calculation, returning a consolidated summary table. This methodology is particularly efficient for generating rapid, comparative statistics across multiple variables.

To ensure the operation executes smoothly and exclusively on quantitative data, two critical parameters must be correctly configured. First, we set `numeric_only=True`. This is a safeguard that tells Pandas to ignore categorical columns (like 'team'), preventing type errors. Second, setting `axis=0` confirms that the calculation is performed column-wise, meaning the quartiles are derived from the values down each feature, independent of the other features. The input list `q=` specifies the desired quartiles.

 # Calculate quartiles for each numeric column in the DataFrame
df.quantile(q=, axis=0, numeric_only=True)

points assists

0.25 14.5 3.00
0.50 25.0 5.00
0.75 29.5 7.75

The resulting output is a succinct, highly readable table, itself a Pandas DataFrame, where the rows represent the quantile levels (0.25, 0.50, 0.75) and the columns correspond to the original numerical features ('points' and 'assists'). This structure facilitates direct comparison of the distributional properties of both variables. For instance, comparing the median values (Q2), we see that the median points (25.0) is five times higher than the median assists (5.0). Such immediate comparisons are invaluable for quickly identifying differences in scale and central tendency between features in a dataset.

Understanding Different Interpolation Methods

Achieving accurate descriptive statistics requires analysts to acknowledge that there is not one single, universally mandated method for calculating quartiles and percentiles, particularly when the sample size is small or the data is discrete. The `.quantile()` function in Pandas, which relies on the underlying NumPy implementation, is designed to be highly flexible, offering several interpolation methods to handle scenarios where the target percentile rank falls exactly between two observed data points.

By default, the `'linear'` interpolation method is employed. This standard technique estimates the quantile value by drawing a straight line between the two closest data points and finding the value on that line corresponding to the desired percentile rank. However, different academic disciplines, financial standards, or regulatory bodies may require alternative calculation methods. Pandas supports alternatives like `'lower'`, `'higher'`, `'nearest'`, and `'midpoint'`. These methods can yield slightly differing results, particularly for Q1 and Q3.

The selection of the appropriate interpolation method is a critical step in ensuring the consistency and validity of statistical reporting. Therefore, analysts must be acutely aware of the statistical conventions applicable to their field of study. We strongly recommend consulting the official Pandas documentation for a detailed technical explanation of the various interpolation algorithms that the pandas **quantile()** function uses to calculate quartiles. Choosing the correct method guarantees that the calculated quartiles accurately meet the required statistical definition.

Additional Resources for Pandas Analysis

While mastering the calculation of quartiles and percentiles is essential for understanding data distribution, it represents only one facet of effective data analysis within the Python ecosystem. To truly unlock the potential of data science, users must expand their proficiency across various data

manipulation and statistical processing techniques supported by Pandas. This includes skills such as data cleaning, advanced grouping operations (using `groupby()`), time series analysis, and powerful data visualization methods.

The techniques demonstrated here form the foundational skills for any data professional. Expanding your knowledge base to include these core functions is crucial for maximizing the utility of Python's most popular data analysis library and enabling deeper, more actionable insights into complex datasets.