Calculating Percentile Rank in Pandas: A Step-by-Step Guide

Authored by

Mohammed looti

October 27, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Calculating Percentile Rank in Pandas: A Step-by-Step Guide*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=4435

The **percentile rank** of a specific value is a fundamental concept in statistics, indicating the percentage of scores or values within a **dataset** that are equal to or less than that particular value. Understanding percentile rank is crucial for comparing individual performance within a group or assessing the distribution of data points.

When working with large datasets in **Pandas**, calculating this rank efficiently requires leveraging built-in methods designed for high performance. The core functionality relies on the powerful `rank()` method, specifically when setting the `pct` parameter to `True`.

We will explore two primary methods for calculating the **percentile rank** using a **DataFrame** in **Pandas**:

Method 1: Calculate Percentile Rank for a Single Column

This method calculates the percentile rank across all values in the specified column, treating the entire column as one continuous distribution. It uses the standard `rank()` function directly on the Series object, ensuring a quick, vectorized calculation.

df = df.rank(pct=True)

Method 2: Calculate Percentile Rank by Group (Conditional Ranking)

Often, you need to calculate the percentile rank relative to specific subgroups within your data--for instance, ranking sales performance only within each region, or scores only within each team. This requires incorporating the `groupby()` method coupled with the `transform()` function. The `transform` function applies the `rank()` operation independently to each defined group while maintaining the original index alignment of the **DataFrame**.

df = df.groupby('group_var').transform('rank', pct=True)

Setting Up the Example DataFrame

To demonstrate these methods practically, we will use a sample **DataFrame** containing scores (`points`) achieved by two different `team` groups (A and B). This setup allows us to clearly illustrate the difference between column-wide ranking and group-specific ranking.

import pandas as pd

#create DataFrame
df = pd.DataFrame({'team': ,

'points': })

#view DataFrame
print(df)

team points
0 A 2
1 A 5
2 A 5
3 A 7
4 A 9
5 A 13
6 A 15
7 B 17
8 B 22
9 B 24
10 B 30
11 B 31
12 B 38
13 B 39

As shown above, our **DataFrame**, `df`, contains 14 observations, split evenly between Team A and Team B. We will now apply the methods described previously to calculate the **percentile rank** for the `points` column.

Example 1: Calculating Percentile Rank Across the Entire Dataset

In this first example, we treat all 14 scores in the `points` column as a single population. The `rank()` function is invoked with `pct=True`. The `pct=True` argument is critical; it automatically normalizes the rank values such that the resulting output represents the percentile (a value between 0 and 1.0) rather than a simple ordinal rank (1, 2, 3, etc.).

The calculation accounts for ties by assigning the average rank to tied values. For instance, if two values are tied for the 2nd and 3rd rank positions, both will receive a rank of 2.5. When converted to a percentile, this ensures fairness and statistical accuracy across the **dataset**.

#add new column that shows percentile rank of points
df = df.rank(pct=True)

#view updated DataFrame
print(df)

team points percent_rank
0 A 2 0.071429
1 A 5 0.178571
2 A 5 0.178571
3 A 7 0.285714
4 A 9 0.357143
5 A 13 0.428571
6 A 15 0.500000
7 B 17 0.571429
8 B 22 0.642857
9 B 24 0.714286
10 B 30 0.785714
11 B 31 0.857143
12 B 38 0.928571
13 B 39 1.000000

The resulting `percent_rank` column provides the proportional rank for each score relative to the total population of 14 scores. Note that the two scores of 5 (indices 1 and 2) share the same percentile rank of 0.178571, demonstrating how **Pandas** handles tied values by averaging their rank positions.

Interpreting Column-Wide Percentile Ranks

The values generated in the `percent_rank` column are fractions, which can be easily converted to percentages for clearer interpretation. These values state what percentage of the total scores fall at or below the score in question. This provides immediate context for how well a score performs across the entire combined sample:

The score of 2 has a percentile rank of 0.071429. This means that **7.14%** of all observed `points` values in the combined dataset are equal to or less than 2.

The score of 5 has a rank of 0.178571. This indicates that approximately **17.86%** of all scores are equal to or less than 5. Since two entries have this score, they share this rank.

The score of 15 has a rank of 0.500000. This is the median (or 50th percentile rank), meaning exactly half of the scores in the entire dataset are 15 or below.

This perspective is useful when the overall performance of the scores, irrespective of the grouping variable (`team`), is the primary interest.

Example 2: Calculating Percentile Rank by Group (Conditional Ranking)

When statistical analysis requires comparison within specific subgroups, calculating the **percentile rank** conditionally becomes necessary. In this example, we calculate the rank of a score only relative to other scores within its respective `team` (A or B). This is achieved by combining the `groupby()` method with the `transform()` function.

The `groupby('team')` operation segments the **DataFrame** into two temporary sub-DataFrames (Team A and Team B). The `transform('rank', pct=True)` then applies the percentile ranking logic independently to the `points` column of each subgroup. Since there are 7 scores in Team A and 7 scores in Team B, the ranking is normalized based on a maximum rank count of 7, rather than 14.

#add new column that shows percentile rank of points, grouped by team
df = df.groupby('team').transform('rank', pct=True)

#view updated DataFrame
print(df)

team points percent_rank
0 A 2 0.142857
1 A 5 0.357143
2 A 5 0.357143
3 A 7 0.571429
4 A 9 0.714286
5 A 13 0.857143
6 A 15 1.000000
7 B 17 0.142857
8 B 22 0.285714
9 B 24 0.428571
10 B 30 0.571429
11 B 31 0.714286
12 B 38 0.857143
13 B 39 1.000000

Notice how the percentile ranks change dramatically compared to Example 1. For instance, the score of 15 (index 6) achieved a rank of 0.500000 when calculated globally, but now achieves a rank of 1.000000 when ranked only among Team A, reflecting that it is the highest score within its group.

Interpreting Grouped Percentile Ranks

When ranks are calculated conditionally using `groupby()`, the interpretation must reference the specific subgroup: the percentile value now represents the score's standing relative only to its peers. This is essential for fairness in comparative reporting.

The lowest score for Team A, 2, has a rank of 0.142857. This means that **14.3%** of the points values for Team A are equal to or less than 2.

The score of 5 for Team A has a rank of 0.357143. This indicates that **35.7%** of Team A's scores are equal to or less than 5.

The score of 17 for Team B is the lowest score for that team, also yielding a rank of 0.142857. This shows that **14.3%** of Team B's scores are equal to or less than 17, demonstrating equivalent relative performance despite the vastly different absolute score values compared to Team A.

Using the grouped approach provides meaningful metrics for internal comparison, allowing analysts to gauge performance relative to cohort size and distribution rather than the entire dataset.

Additional Resources for Data Ranking and Analysis

Mastering the `rank()` function is a key step in advanced data analysis using **Pandas**. To further enhance your proficiency in data manipulation and statistical processing, consider exploring these related topics:

Calculating Rolling Means and Statistics in Pandas

Understanding the Different Tie-Breaking Methods in Pandas Rank

Using the `groupby()` and `transform()` methods for Time Series Data

These tutorials explain how to perform other common tasks in **Pandas**, building upon the foundational knowledge of ranking and grouping demonstrated here.