How do you calculate percentiles in Python?

Authored by

stats writer

December 19, 2025

RECOMMENDED CITATION

stats writer (2025). *How do you calculate percentiles in Python?*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=107916

ARABPSYCHOLOGY.COM

Calculating **percentiles** is a fundamental operation in statistical analysis and data science. In the Python ecosystem, this task is efficiently handled by the industry-standard numerical computing library, NumPy. Specifically, the `numpy.percentile()` function provides a robust and quick method for determining the value below which a given percentage of observations in a dataset falls.

Understanding how to use this function is crucial for tasks ranging from identifying outliers to normalizing data distributions. When you call `numpy.percentile(data_array, q)`, where `q` represents the desired percentile (e.g., 75 for the 75th percentile), the function scans the array, sorts the values internally, and returns the specific data point corresponding to that cutoff percentage. This method ensures that the calculation adheres to standard statistical definitions even when dealing with large volumes of data, which is critical for maintaining integrity in complex analyses.

This comprehensive guide will detail the structure and application of percentile calculations using both NumPy for basic arrays and Pandas for complex DataFrame structures. We will explore how to calculate single percentiles, extract common metrics like quartiles, and efficiently analyze data spread across multiple variables, ensuring you can accurately derive these crucial metrics in your analytical workflows.

Understanding the Concept of a Percentile in Statistics

Statistically, the nth **percentile** of a given dataset is defined as the numerical value below which *n* percent of the observations may be found. Essentially, it is a non-parametric measure of position used in statistics to indicate the value below which a specified percentage of observations in a group of observations falls. This calculation requires that the entire dataset be sorted internally from the smallest value to the largest, establishing the foundational order necessary for identifying the cutoff points.

To illustrate, if a student scores in the 90th percentile on an exam, it means their score is higher than 90% of the scores achieved by other test-takers. Conversely, the 10th percentile would define the score below which the bottom 10% of students scored. Identifying key percentiles, such as the 25th, 50th, and 75th (known as quartiles), allows us to construct the crucial five-number summary (minimum, Q1, median, Q3, maximum) necessary for building box plots and gaining immediate insight into data distribution, skewness, and overall spread.

The Core NumPy Function for Percentile Calculation

We can rapidly and accurately calculate percentiles in Python using the high-performance `numpy.percentile()` function. This function is the cornerstone for percentile calculations in the scientific Python stack due to its speed, especially when handling large, multi-dimensional array

ARABPSYCHOLOGY.COM

data structures. The standard syntax for invoking this function is robust and flexible:

numpy.percentile(a, q, axis=None, out=None, interpolation='linear', keepdims=False)

While the function includes several optional parameters for advanced usage (such as specifying an `axis` for multi-dimensional calculations or defining the `interpolation` method), the core usage involves the first two arguments, **a** and **q**. Understanding these parameters is essential for successful implementation:

a: This required parameter represents the input array or list of values for which the percentile must be computed. It is typically a one-dimensional NumPy array but can also accept Python lists.
q: This argument defines the specific percentile or sequence of percentiles (e.g.,) to calculate. Critically, the value(s) must be provided as an integer or list of integers between 0 and 100, inclusive, representing the percentage cutoff.

The `numpy.percentile()` function defaults to using a linear interpolation method, which is the standard technique used to estimate percentile values that fall between two observed data points in the sorted sequence. This tutorial will focus on practical examples demonstrating how to apply this function effectively across various data structures, starting with simple NumPy arrays.

Calculating Specific Percentiles for a NumPy Array

When working directly with numerical data stored as a NumPy array, the calculation of a single, specific percentile is a fundamental step in descriptive analysis. We begin by importing the library, generating a sample dataset (for reproducibility, we use a fixed random seed), and then applying the `numpy.percentile()` function to the generated data. This method is the fastest way to extract this metric from raw numerical sequences.

In the example below, we calculate the 37th percentile of a randomly generated dataset containing 100 integers distributed between 0 and 500. This single value tells us the precise boundary below which 37% of our collected data points reside, giving us a highly specific metric of distribution.

import numpy as np

#make this example reproducible
np.random.seed(0)

#create array of 100 random integers distributed between 0 and 500
data = np.random.randint(0, 500, 100)

#find the 37th percentile of the array
np.percentile(data, 37)

ARABPSYCHOLOGY.COM

173.26

The resulting value, 173.26, indicates that 37% of the 100 generated random integers have a value of 173.26 or less. This calculation is fast and robust, making it the preferred method for assessing distribution characteristics in large, single-dimensional datasets before they are potentially loaded into more complex structures like data frames. The precise calculation involves sorting the 100 elements and finding the interpolated value at position 37.

Calculating Multiple Percentiles (Quartiles) Simultaneously

A frequent requirement in exploratory data analysis (EDA) is the calculation of quartiles, which are specific percentiles that divide the data into four equal segments. These correspond precisely to the 25th, 50th (the median), and 75th percentiles. Instead of calling `numpy.percentile()` three separate times, NumPy allows for highly optimized batch processing by permitting us to pass a list of percentile values to the `q` argument.

This approach significantly streamlines the process and returns an array containing all requested percentile values in the order specified in the input list. Calculating these quartiles provides immediate insight into the central tendency (the 50th percentile) and the interquartile range (IQR), which measures the spread of the middle 50% of the data, thereby offering a metric of dispersion that is robust to outliers.

#Find the quartiles (25th, 50th, and 75th percentiles) of the array
np.percentile(data,)

array()

From the output, we observe that the **first quartile** (25th percentile) is 116.5, the **median** (50th percentile) is 243.5, and the **third quartile** (75th percentile) is 371.5. These values are crucial for creating descriptive statistics and are the exact metrics required for visualizing data spread using box plots, making this batch calculation essential for rapid data profiling.

Finding Percentiles of a Single DataFrame Column Using NumPy

While NumPy is primarily designed for array manipulation, it can seamlessly integrate with Pandas data structures. When we need to calculate a percentile for a specific column within a DataFrame, we can extract that column as a Pandas Series object. Since a Series is built upon NumPy arrays, it can be passed directly to `numpy.percentile()` without any conversion issues.

This method is particularly effective when you are already utilizing NumPy extensively in your script

ARABPSYCHOLOGY.COM

or need the specific, detailed interpolation method parameters offered by the NumPy function. Below, we first construct a simple DataFrame and then calculate the 95th percentile for the 'var1' column. This high percentile is often used in performance testing scenarios to set upper bounds or define confidence limits that exclude rare, extreme events.

import numpy as np
import pandas as pd

#create DataFrame
df = pd.DataFrame({'var1': ,
'var2': ,
'var3': })

#find 95th percentile of var1 column
np.percentile(df.var1, 95)

34.1

The result, 34.1, signifies that 95% of the values in 'var1' are less than or equal to this number. When calculating percentiles on DataFrame columns, it is crucial to ensure the column contains numerical data; otherwise, calling `numpy.percentile()` may raise a TypeError or produce unexpected results if applied to strings or mixed types.

Calculating Percentiles Across All DataFrame Columns Using Pandas

For users working predominantly within the Pandas environment, using the built-in `quantile()` method is often the most idiomatic and efficient approach. This function is specifically engineered to calculate quantiles (where the percentile *q* corresponds to the quantile *p = q/100*) directly across one or more columns of a DataFrame. It offers significant syntactic convenience over manually extracting Series objects for NumPy processing.

A key advantage of `df.quantile()` is its seamless application across an entire DataFrame. By passing the desired quantile value (expressed as a decimal between 0.0 and 1.0, unlike NumPy which uses 0 to 100), Pandas automatically iterates through all numerical columns, calculates the statistic for each one, and returns the results as a new Series object indexed by the original column names. This is especially useful for quickly profiling a dataset with dozens of variables.

import numpy as np
import pandas as pd

#create DataFrame

ARABPSYCHOLOGY.COM

df = pd.DataFrame({'var1': ,
'var2': ,
'var3': })

#find 95th percentile (0.95 quantile) of each column
df.quantile(.95)

var1 34.10
var2 14.55
var3 14.65

Targeting Specific Columns for Percentile Analysis

If the analytical task requires focusing only on a subset of columns within a large DataFrame, the `Pandas` method remains superior due to its inherent support for DataFrame indexing. We can select the desired columns using standard DataFrame indexing (by passing a list of column names, such as `]`) and then chain the `quantile()` method directly onto the resulting subsetted DataFrame.

This selectivity is critical in analyses where some columns might be categorical, text-based, or otherwise irrelevant to the numerical distribution being assessed. By isolating 'var1' and 'var2', as shown below, we maintain clean code, targeted results, and avoid errors that might arise from attempting numerical calculations on non-numerical data. This process demonstrates the optimized workflow designed by Pandas for data manipulation and statistical summation.

#find 95th percentile of just columns var1 and var2
df].quantile(.95)

var1 34.10
var2 14.55

It is important to remember the difference in input parameters: NumPy requires the percentile value (0-100), whereas Pandas requires the quantile value (0.0-1.0). Both functions are mathematically calculating the same value, but the choice between them generally depends on whether you prioritize the raw speed and array control of NumPy or the structured, column-based convenience of Pandas.

Advanced Considerations: Interpolation Methods

When the desired percentile boundary falls between two actual data points within the sorted list,

ARABPSYCHOLOGY.COM

statistical packages must employ an interpolation method to estimate the precise value. Understanding which method is used is vital, as it can subtly affect the resulting percentile, especially in datasets with a small number of observations or those with discrete values. Both NumPy and Pandas offer various interpolation options, though the defaults are designed for consistency.

NumPy's `numpy.percentile()` defaults to **'linear' interpolation**. Linear interpolation assumes a straight-line relationship between the two nearest data points and calculates the weighted average based on the exact fractional position of the percentile. However, NumPy also supports methods like 'lower' (returning the data point just below the index), 'higher' (returning the data point just above the index), 'nearest' (rounding to the closest data point), and 'midpoint' (averaging the two nearest data points). These options allow analysts to choose the estimation technique that best fits their specific data distribution or statistical modeling assumptions.

Pandas, specifically the `df.quantile()` method, also defaults to **'linear'** interpolation, ensuring computational consistency with NumPy for basic usage. For instance, if you were analyzing discrete survey data and needed the 80th percentile, selecting 'lower' would offer a conservative estimate by returning the existing value closest to, but not exceeding, the 80% mark. Conversely, 'higher' would return the next existing value, providing a slightly more optimistic estimate. These fine controls ensure accuracy and defensibility in sensitive analyses where interpolation choice matters.

Practical Applications of Percentile Calculations

Percentiles are not just abstract statistical measures; they are indispensable tools across numerous fields in real-world data analysis, providing concrete reference points for performance assessment and threshold setting. Their use provides context that simple averages often obscure, particularly in skewed distributions.

Performance Benchmarking (SLA Monitoring): In IT and systems operations, the 95th or 99th percentile is the industry standard for measuring service latency or response time. For example, a 95th percentile latency of 500ms means 95% of user requests are successfully completed within half a second. This metric is far more reliable and informative than simply using the mean, which can be easily inflated and skewed by just a few extreme outliers or slow server responses.

Healthcare and Biological Studies: Pediatricians routinely use growth charts based on percentiles to monitor a child's height, weight, and head circumference relative to peer populations. A child consistently outside the 3rd or 97th percentile might warrant further medical investigation, using these boundaries as clear diagnostic thresholds.

Financial Risk Assessment: Value at Risk (VaR) calculations, a cornerstone of financial risk management, frequently utilize high percentiles (e.g., the 99th percentile) to estimate the maximum

ARABPSYCHOLOGY.COM

potential financial loss of a portfolio over a specific time frame with a certain confidence level. The 99th percentile loss represents the expected loss exceeded only 1% of the time.

Outlier Detection and Data Cleaning: Data points that fall below the 1st percentile or above the 99th percentile are strong candidates for outliers. Calculating these extreme boundaries quickly using NumPy or Pandas provides a robust, distribution-based method for identifying and potentially removing anomalous data points before running sensitive machine learning models.

Mastering the calculation of percentiles in Python, whether through the fine-grained control of `numpy.percentile()` for arrays or the data-frame-centric efficiency of `pandas.DataFrame.quantile()`, is a crucial skill for any analyst aiming to derive meaningful, robust insights from raw numerical data. The strategic choice between these two primary methods should always be dictated by the structure of the data and the overall efficiency required by the analytical pipeline.

ARABPSYCHOLOGY.COM