How to Calculate Mean, Median, and Mode Using Pandas: A Step-by-Step Guide

Authored by

stats writer

November 21, 2025

Pandas is an indispensable library within the Python ecosystem, specifically designed to handle and analyze data efficiently. It offers robust tools for performing complex calculations, particularly in the realm of descriptive statistics. Among the most frequently sought measures are the indicators of central tendency: the mean, the median, and the mode. This powerful library simplifies the process of obtaining these metrics, transforming raw data into meaningful insights with minimal code.

For data analysts and scientists, understanding the central characteristics of a dataset is a foundational requirement. Pandas provides dedicated, high-performance functions--specifically, **.mean()**, **.median()**, and **.mode()**--that rapidly calculate these measures across entire columns or specific data subsets. Utilizing these built-in functions ensures both computational speed and accuracy, making **Pandas** the preferred tool for initial data exploration and statistical summarization in Python environments.

Accessing Central Tendency Measures

When working with a Pandas DataFrame, you often need to calculate the measures of central tendency for multiple columns simultaneously. The methods **.mean()**, **.median()**, and **.mode()** are applied directly to the DataFrame object. It is crucial to use the argument **numeric_only=True** to ensure these calculations are restricted solely to the columns containing numerical data, thereby preventing errors when non-numeric columns (like strings or dates) are present.

The following snippet demonstrates the standard syntax used to calculate the **mean**, **median**, and **mode** for every applicable column within your dataset. The output will be a Pandas Series object, where the index represents the column name and the values represent the calculated statistic.

```
print(df.mean(numeric_only=True))
print(df.median(numeric_only=True))
print(df.mode(numeric_only=True))
```

To illustrate these concepts effectively, we will walk through a complete, hands-on example, starting with the creation of a sample DataFrame tailored for this statistical analysis.

Example: Setting up the Sample Data

For our practical demonstration, we will analyze hypothetical performance data. Imagine a scenario where we track the points scored by several basketball players across four different games. This data structure naturally fits into a Pandas DataFrame, allowing us to easily apply statistical functions to the numerical game columns.

The code below initializes the DataFrame named **df**. It includes one categorical column (**player**) and four quantitative columns (**game1** through **game4**), which represent the individual scores achieved by eight distinct players.

import pandas as pd

# Create the DataFrame containing player scores across four games
df = pd.DataFrame({'player': ,
'game1': ,
'game2': ,
'game3': ,
'game4': })

# Display the complete DataFrame
print(df)

player game1 game2 game3 game4
0 A 18 5 11 9
1 B 22 7 8 8
2 C 19 7 10 10
3 D 14 9 6 9
4 E 14 12 6 14
5 F 11 9 5 15
6 G 20 9 9 10
7 H 28 4 12 11

Calculating the Mean (Average) Score

The arithmetic mean is perhaps the most common measure of central tendency. It is calculated by summing all the values in a dataset and dividing by the total count of observations. In Pandas, the **.mean()** function calculates this average across all specified numerical columns swiftly, providing a single typical value for each game's performance.

We apply the **.mean()** function to our DataFrame **df**, again specifying **numeric_only=True** to target only the scoring columns. The resulting output clearly shows the average points scored across all players for each respective game, giving us a baseline understanding of player performance for each matchup.

 # Calculate the arithmetic mean of each numerical column
print(df.mean(numeric_only=True))

game1 18.250
game2 7.750
game3 8.375
game4 10.750
dtype: float64

Based on the results generated by the **.mean()** function, we can draw the following conclusions regarding the average performance in each game:

The average score (mean value) recorded in the **game1** column is **18.25** points.
The average score (mean value) recorded in the **game2** column is **7.75** points.
The average score (mean value) recorded in the **game3** column is **8.375** points.
The average score (mean value) recorded in the **game4** column is **10.75** points.

Determining the Median (Middle Value) Score

While the mean is sensitive to outliers and skewed distributions, the median provides a more robust measure of central tendency. The **median** is defined as the value that physically separates the higher half from the lower half of a data sample when the data is sorted.

When dealing with an even number of observations, the median is calculated as the average of the two central values. To compute this midpoint for each game, we utilize the **.median()** function. This calculation is vital because it offers a center point that is not disproportionately influenced by extremely high or low individual scores, giving a clearer picture of typical player performance without statistical bias from extremes.

 # Calculate the median value of each numerical column
print(df.median(numeric_only=True))

game1 18.5
game2 8.0
game3 8.5
game4 10.0
dtype: float64

Examining the output of the **.median()** function reveals the following middle scores for our dataset:

The median score in the **game1** column is **18.5** points.
The median score in the **game2** column is **8** points.
The median score in the **game3** column is **8.5** points.

The median score in the **game4** column is **10** points.

Finding the Mode (Most Frequent Value) Score

The mode represents the value that appears most frequently in a dataset. Understanding the mode is essential for categorical data, but it also highlights the most common occurrences in numerical data. Unlike the mean and median, a dataset can be unimodal, multimodal (having multiple modes), or have no mode at all if all values are unique. The **.mode()** function in Pandas is specifically designed to handle this complexity by returning all modes found within the data.

Because a column might possess multiple modes, the output structure of **.mode()** differs from the single-value results of the mean and median functions. It returns a DataFrame where multiple rows may be used to list all modes for a specific column. This structure ensures that no frequent value is missed, with missing modes represented by **NaN** values in the subsequent rows.

 # Calculate the mode(s) of each numerical column
print(df.mode(numeric_only=True))

game1 game2 game3 game4
0 14.0 9.0 6.0 9
1 NaN NaN NaN 10

Analyzing the resulting DataFrame provides a comprehensive view of the most common scores for each game:

The mode in the **game1** column is **14**.
The mode in the **game2** column is **9**.
The mode in the **game3** column is **6**.
The **game4** column is bimodal, possessing two modes: **9** and **10**.

It is important to notice how **game4** required two rows to display its results. This indicates that the values 9 and 10 occurred with equal, highest frequency in that particular game column. If a column had only one mode, the subsequent rows for that column would contain **NaN** (Not a Number) entries.

Summary and Further Exploration

We have successfully demonstrated how to leverage the core statistical functions in **Pandas** to rapidly calculate the three primary measures of central tendency across a DataFrame containing numerical data. The functions **.mean()**, **.median()**, and **.mode()** are essential tools for any data analysis workflow, providing immediate, crucial insights into the typical values within a dataset.

For researchers requiring a broader overview that goes beyond central tendency, Pandas provides an even more comprehensive function: **.describe()**. This method generates a table containing eight key descriptive statistics, including measures of dispersion (like standard deviation) and location (quartiles).

Note: You can also use the **.describe()** function in pandas to generate a richer set of descriptive statistics for each numerical column in your DataFrame, which is highly recommended for initial data profiling.