How to Calculate Median Absolute Deviation in Python

Authored by

stats writer

December 16, 2025

The Median Absolute Deviation (MAD) is a critical measure of statistical dispersion, offering a robust alternative to the traditional variance or standard deviation. It serves as an essential metric for quantifying the spread or variability within a dataset, particularly when working in Python environments using scientific computing libraries. The calculation of MAD begins by first determining the dataset's median, which acts as the center point. Next, the absolute difference between every data point and this median is calculated. Finally, the MAD itself is defined as the median of these calculated absolute differences. This methodology ensures that MAD is a highly robust measure of variation, meaning it is significantly less sensitive to the influence of extreme values, or outliers, compared to mean-based metrics.

Understanding Median Absolute Deviation (MAD)

The **median absolute deviation** measures the spread of data points around the center in a dataset. Unlike measures like variance, which rely on the squared difference from the mean, MAD provides a straightforward and intuitive measure of variability based entirely on medians. This makes it a foundational tool in exploratory data analysis, particularly when dealing with non-normal or skewed distributions.

MAD is a particularly useful metric because it falls under the category of robust statistics. This means its value is less affected by severe deviations in the data. When outliers are present--perhaps due to measurement error or inherent skewness--the standard deviation can become inflated, leading to a misleading interpretation of the dataset's true dispersion. MAD reliably resists this inflation, providing a more truthful reflection of the typical spread.

The process involves three steps: finding the median of the data, calculating the absolute residuals (differences) from that median, and then finding the median of those residuals. This iterative application of the median operator guarantees the robustness of the resulting measure against data contamination or extreme values.

The Mathematical Foundation of MAD

The formal mathematical definition used to calculate Median Absolute Deviation, often abbreviated MAD, is derived directly from the steps outlined above:

MAD = median(|xi - xm|)

where the terms represent the following key components:

xi: Represents the ith individual observation or value in the entire dataset being analyzed.
xm: Represents the median value calculated across all observations in the dataset (i.e., the measure of central tendency).

|...|: Denotes the absolute value, ensuring that negative differences are treated identically to positive ones, quantifying distance rather than direction.

Implementation Setup using Python Libraries

While base Python can calculate the median, calculating the MAD efficiently for large arrays typically requires specialized libraries. The following examples demonstrate how to calculate the Median Absolute Deviation in Python by leveraging the powerful array manipulation capabilities of NumPy and the dedicated statistical functions found within the statsmodels package, specifically the `robust.mad` function. These tools streamline complex calculations, making statistical analysis highly accessible.

Example 1: Calculating MAD for a NumPy Array

To demonstrate the basic calculation, we will use a small, defined dataset stored as a NumPy array. This approach is standard practice when dealing with numerical computations in data science. The following code imports the necessary modules and applies the `mad` function to calculate the median absolute deviation:

```
import numpy as np
from statsmodels import robust

#define data
data = np.array()

#calculate MAD
robust.mad(data)
```

11.1195

As shown by the output, the calculated median absolute deviation for this specific dataset turns out to be **11.1195**. This value represents the typical spread of the data points around their central median.

Understanding this initial result is key to interpreting the dataset's variability. A higher MAD suggests greater dispersion, while a lower MAD indicates that data points are tightly clustered around the median.

Interpreting and Adjusting the Scaling Factor

It is essential to note a critical detail about how the `robust.mad` function operates: by default, it

does not simply return the median of the absolute deviations. Instead, the function computes a robust estimate of the standard deviation assuming the underlying data follows a normal distribution. This transformation is achieved by scaling the raw MAD result by a correction factor, typically around 1.4826 (which is equivalent to 1/0.6745). This scaling allows the MAD to be directly comparable to the standard deviation for normally distributed data.

If the goal is strictly to obtain the mathematically defined MAD--the pure median of the absolute residuals--and avoid using this normalizing scaling factor, the user must explicitly set the scaling parameter `c` equal to 1 within the function call. This modification ensures that the output is the unadjusted robust statistic.

To calculate the exact, unscaled MAD value for the previously defined array, we simply adjust the `c` parameter as follows:

#calculate MAD without scaling factor
robust.mad(data, c=1)

7.5

In this adjusted calculation, the true MAD is **7.5**. The difference between 11.1195 and 7.5 highlights the impact of the scaling factor designed to estimate the standard deviation. Researchers must decide whether they need the normalized estimate (default) or the raw robust statistic (using `c=1`).

Example 2: Calculating MAD Across Pandas DataFrame Columns

In real-world data analysis, data is often stored and managed within pandas DataFrames. Calculating MAD for a single column within a DataFrame requires using the `apply` method in conjunction with the `robust.mad` function imported from statsmodels. This powerful combination allows column-wise application of custom statistical functions.

We first set up a reproducible example DataFrame using NumPy and pandas:

#make this example reproducible
np.random.seed(1)

#create pandas DataFrame
data = pd.DataFrame(np.random.randint(0, 10, size=(5, 3)), columns=)

#view DataFrame
data

A B C
0 5 8 9
1 5 0 0
2 1 7 6
3 9 2 4
4 5 2 4

#calculate MAD for column *B*
data].apply(robust.mad)

B 2.965204
dtype: float64

The output confirms that the Median Absolute Deviation for column B, using the default scaling factor (robust standard deviation estimate), is **2.965204**. This method is highly effective for isolating the dispersion of individual features within a complex dataset.

Calculating MAD for Multiple DataFrame Variables

The power of the `apply` function is truly realized when calculating MAD simultaneously across multiple or all columns in the pandas DataFrame. By selecting multiple columns before applying the function, we can efficiently compute the MAD for each variable independently, providing a comparative view of dispersion across the entire dataset without needing a loop.

We can use similar syntax to calculate MAD for multiple columns by passing a list of column names to the DataFrame subsetting operation:

#calculate MAD for all columns
data].apply(robust.mad)

A 0.000000
B 2.965204
C 2.965204
dtype: float64

The median absolute deviation calculated for Column A is **0.000000**, indicating that all values in that column are identical to the column median (in this case, all values are 5, so the median is 5, and the absolute deviations are all 0). The MAD is **2.965204** for Column B and **2.965204** for Column C. This parallel calculation capability makes MAD a streamlined metric for assessing variability across large, multivariate datasets.

Conclusion and Applications

The Median Absolute Deviation provides analysts with a powerful, robust alternative to traditional dispersion metrics. Its resistance to outliers makes it invaluable in fields where data integrity is often compromised by extreme values, such as financial modeling, quality control, and environmental monitoring. By understanding the implementation details, including the scaling factor used by libraries like statsmodels, data scientists can accurately interpret and utilize MAD to characterize the true spread of their data in Python.