Learn How to Remove Columns with NaN Values from Pandas DataFrames

Authored by

Mohammed looti

February 13, 2026

Introduction to Handling Missing Data in Pandas

Data cleaning is a fundamental step in any data preparation workflow. When analyzing real-world datasets, encountering missing entries is inevitable. In the Pandas ecosystem, these missing values are typically denoted as NaN (Not a Number). The prevalence of NaN values can significantly impair statistical models, distort descriptive statistics, and complicate subsequent feature engineering steps. Therefore, developing robust strategies for managing missing data is crucial for maintaining the integrity and reliability of the final analysis.

One powerful strategy for handling missing data, especially when certain features are excessively sparse, is the removal of columns containing insufficient valid information. The Pandas library offers the flexible `dropna()` method, which allows data scientists to precisely control how missing values trigger the deletion of rows or columns. By setting the appropriate parameters, we can quickly eliminate columns that fail to meet a specified completeness threshold.

The following sections detail the three principal ways to use `dropna()` to remove columns (using `axis=1`) based on the quantity of missing data they contain. Understanding these nuances allows for targeted data cleaning, ensuring that only necessary and useful features are retained in your DataFrame.

Three Essential Methods for Column Deletion

Effective handling of missing data requires choosing the right balance between data preservation and feature quality. The `dropna()` function provides three distinct modes, controlled primarily by the `how` and `thresh` arguments, which determine the condition under which a column is dropped:

Method 1: Drop Columns with Any NaN Values (Default behavior): If a column contains even a single instance of NaN, it is removed entirely. This is achieved by default or by explicitly setting `how='any'`.

Method 2: Drop Columns with All NaN Values: Only columns where every single cell is NaN values are dropped. This is the most conservative approach and is set using `how='all'`.

Method 3: Drop Columns Using a Threshold: Columns are kept only if they have at least a minimum number of non-missing values (specified by the `thresh` parameter).

Below is the concise syntax for implementing each of these powerful column-dropping strategies within a DataFrame:

Method 1: Drop Columns with Any NaN Values

df = df.dropna(axis=1)

Method 2: Drop Columns with All NaN Values

df = df.dropna(axis=1, how='all')

Method 3: Drop Columns with Minimum Number of Non-NaN Values (Threshold)

df = df.dropna(axis=1, thresh=2)

The following examples demonstrate how to use each method in practice, utilizing a sample DataFrame specifically constructed to showcase the differences in output based on the missing data criteria.

Setting Up the Example DataFrame

To properly test the behavior of the `dropna()` parameters, we first create a sample DataFrame using NumPy for generating explicit missing values (`np.nan`). This sample dataset contains columns with varying levels of missingness: a column with zero missing values, a column with one missing value, and a column that is entirely empty.

import pandas as pd
import numpy as np

#create DataFrame
df = pd.DataFrame({'team': ,
'position': ,
'points': ,
'rebounds': })

#view DataFrame
print(df)

team position points rebounds
0 A NaN 11 NaN
1 A G 28 NaN
2 A F 10 NaN
3 B F 26 NaN
4 B C 6 NaN
5 B G 25 NaN

The columns exhibit the following characteristics: **team** (0 NaN values), **points** (0 NaN values),

position (1 NaN value), and **rebounds** (6 NaN values). We will now proceed to apply the three defined methods, starting with the most stringent criterion.

Example 1: Strict Dropping (Any NaN Value)

In many high-precision analytical contexts, data scientists require absolute completeness. The strictest method for cleaning columns is achieved by simply calling `dropna()` with `axis=1`. This defaults to `how='any'`, meaning any column containing at least one missing value will be removed from the DataFrame.

Applying this method to our sample data will result in the loss of both the **position** and **rebounds** columns, as both contain NaN values, even though **position** is mostly complete.

#drop columns with any NaN values
df = df.dropna(axis=1)

#view updated DataFrame
print(df)

team points
0 A 11
1 A 28
2 A 10
3 B 26
4 B 6
5 B 25

Notice that the **position** and **rebounds** columns were successfully dropped. This outcome confirms that when using the default `how='any'` setting, the presence of even a single NaN value is enough to disqualify a column from retention. The resulting DataFrame is perfectly clean in terms of missing data.

Example 2: Permissive Dropping (All NaN Values)

If the goal is to remove only entirely empty features--those that offer absolutely no information--we employ the `how='all'` parameter alongside `axis=1`. This strategy maximizes the retention of features that have any valid data points, regardless of how many missing values they might contain otherwise.

When we execute this method on our initial DataFrame, we anticipate that the **position** column will be preserved because it contains five valid entries, while only the **rebounds** column should be

eliminated, as it is 100% missing data.

#drop columns with all NaN values
df = df.dropna(axis=1, how='all')

#view updated DataFrame
print(df)

team position points
0 A NaN 11
1 A G 28
2 A F 10
3 B F 26
4 B C 6
5 B G 25

The **rebounds** column was correctly dropped, as it was the sole column composed entirely of NaN values. The **position** column, despite having one missing entry, was retained because it contained some valid data, thereby failing the `how='all'` removal condition. This method is highly effective for eliminating features that are completely non-contributory.

Example 3: Conditional Dropping (Using Threshold)

The `thresh` parameter provides the most adaptable control over column removal. Instead of relying on boolean flags (all or any), `thresh` requires a column to possess a minimum count of non-missing observations to be retained. If the count of non-NaN values in a column falls below this threshold, the column is dropped. This is particularly useful when defining a specific data quality baseline, for example, requiring that 70% of data points must be present for a column to be usable.

For demonstration, we set `thresh=2`, meaning a column must have at least two valid entries to be preserved. Since our 'rebounds' column has zero valid entries, it will be removed, but 'position' (five valid entries) will be kept.

#drop columns with fewer than two non-NaN values
df = df.dropna(axis=1, thresh=2)

#view updated DataFrame
print(df)

team position points

0 A NaN 11
1 A G 28
2 A F 10
3 B F 26
4 B C 6
5 B G 25

The **rebounds** column was dropped because it failed to satisfy the condition of having at least two non-missing entries. This conditional dropping mechanism is invaluable for data analysts seeking to enforce a minimum data density requirement across their feature set, ensuring that only sufficiently populated columns are carried forward into modeling or reporting phases.

Further Exploration and Resources

The methods detailed above provide a robust toolkit for managing missing data at the column level within Pandas DataFrames. While we focused on `axis=1`, remember that `dropna()` is equally powerful for managing rows (`axis=0`). Furthermore, for more complex scenarios, consider using the `subset` parameter to apply these missingness checks only to a specific list of columns. For instance, you could demand that a row must be complete only within the first five columns before being retained.

For analysts engaged in serious data preprocessing, mastering the flexibility of the `dropna()` function is essential. It is highly recommended to consult the official documentation for a comprehensive overview of all parameters and advanced usage patterns related to handling missing data in the Pandas library.

Note: You can find the complete documentation for the **dropna()** function in Pandas.

Additional Resources

The following tutorials explain how to perform other common tasks in pandas: