How do I read an excel file with merged cells using pandas?

Authored by

stats writer

November 21, 2025

Working with real-world data often involves dealing with complexities like Excel spreadsheets that utilize merged cells for better visual presentation. While this formatting is excellent for human readability, it poses a significant challenge when importing the data into programmatic environments, particularly when using the powerful Pandas library in Python. The goal of using Pandas is to structure this raw information into a coherent DataFrame for efficient data analysis and manipulation. Understanding how Pandas interprets these visually merged cells is the first crucial step toward achieving a clean dataset.

The standard approach involves using the essential read_excel() function. Unlike some older or niche tools, Pandas is designed to handle the structure of merged cells during the import process by default. However, instead of replicating the visual merger, which would break the rectangular structure required for a DataFrame, it applies a specific mechanism: the top-most cell in the merged region retains its value, and all subsequent cells within that region are marked as missing data. This results in the insertion of NaN (Not a Number) values, which must then be systematically addressed through data cleaning techniques.

Understanding Pandas' Default Behavior with Merged Cells

The fundamental challenge arises because the Pandas DataFrame requires every row and column intersection to hold a single, distinct value. When you read an Excel file containing merged cells, the default behavior of the Pandas reader ensures structural integrity by filling the duplicated positions with the missing value indicator, NaN. This is a crucial distinction: the data is not lost; it is simply unrepresented in the merged rows, awaiting imputation.

Fortunately, Pandas provides highly optimized methods specifically designed for imputing these missing values. The easiest and most robust way to fill in these NaN values after importing the file is to employ the Pandas **fillna()** function, combined with a directional filling method. This technique allows us to propagate the valid value found in the top row of the merged region downwards, thereby restoring the logical consistency of the dataset. The specific implementation looks like this:

df = df.fillna(method='ffill', axis=0)

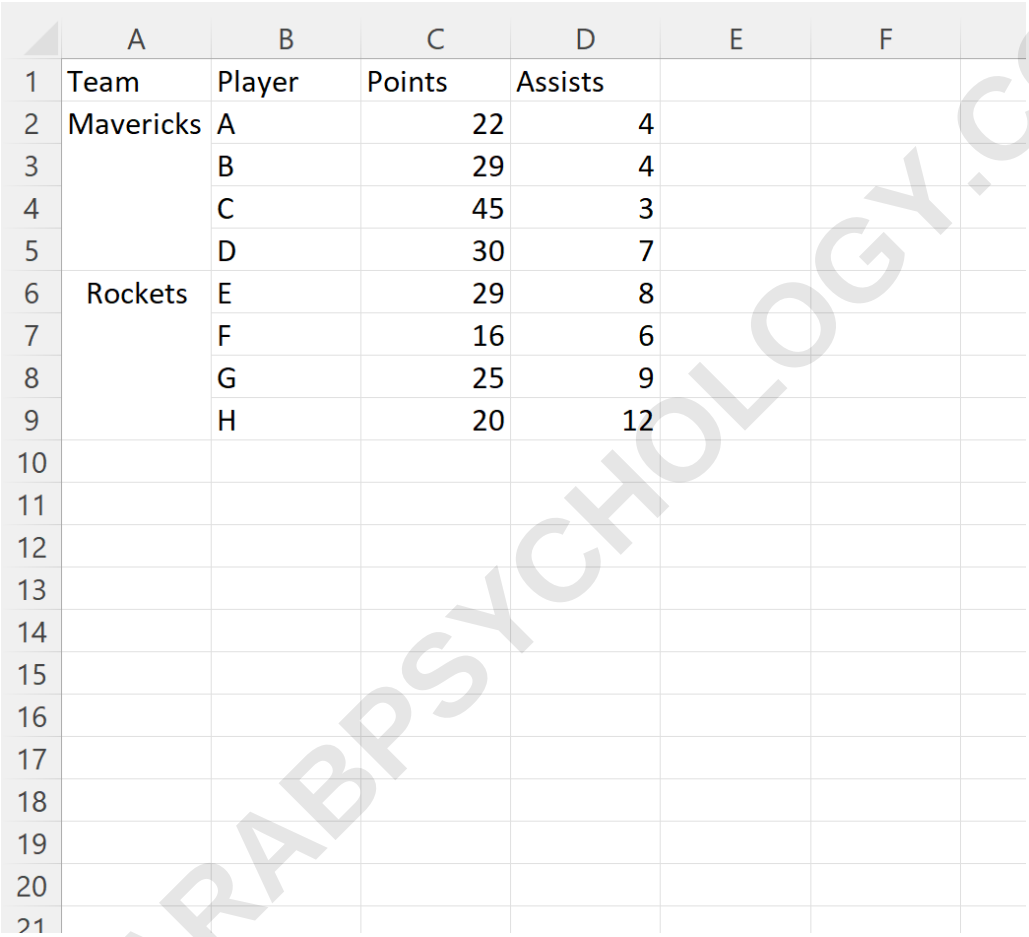
This single line of code is often the complete solution for cleaning data imported from spreadsheets with vertically merged cells. The following detailed explanation and example will show how to utilize this syntax in a practical scenario, transforming messy spreadsheet data into a clean analytical format.

Practical Implementation: Importing and Cleaning Spreadsheet Data

To fully grasp the process of handling merged cells, let us examine a concrete example. We will

use a hypothetical Excel file named **merged_data.xlsx**. This file contains performance metrics for various basketball players, where the team names are grouped using vertical merged cells--a common pattern in manually prepared reports meant for human consumption.

The structure of the raw data in the Excel file clearly shows two distinct groups: the Mavericks and the Rockets. Visually, the team name spans multiple rows, correlating to the players listed below it. However, the underlying data structure treats only the top cell of that merged block as containing the actual value, while the remaining cells are technically empty.



	A	B	C	D	E	F
1	Team	Player	Points	Assists		
2	Mavericks	A	22	4		
3		B	29	4		
4		C	45	3		
5		D	30	7		
6	Rockets	E	29	8		
7		F	16	6		
8		G	25	9		
9		H	20	12		
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						

As illustrated in the source file image above, observe closely the values in the **Team** column. The cell corresponding to Player A contains "Mavericks," but the cells for Players B, C, and D are visually merged with it. Similarly, "Rockets" occupies a merged block corresponding to Players E, F, G, and H. This visual merging is what causes the data integrity challenge during import.

Specifically, Players A through D are associated with the Mavericks, and players E through H are associated with the Rockets. Our ultimate goal is to ensure that the DataFrame accurately reflects this association by filling in the team names for every row.

Step 1: Reading the Raw Excel File into Pandas

The first step involves utilizing the read_excel() function to ingest the data. When we execute this command, Pandas successfully parses the file structure but, adhering to the standard data model, inserts NaN values where the merged cells were encountered, as these spots lack independent values.

```
import pandas as pd

#import Excel fie
df = pd.read_excel('merged_data.xlsx')

#view DataFrame
print(df)
```

```
Team Player Points Assists
0 Mavericks A 22 4
1 NaN B 29 4
2 NaN C 45 3
3 NaN D 30 7
4 Rockets E 29 8
5 NaN F 16 6
6 NaN G 25 9
7 NaN H 20 12
```

As demonstrated by the output above, rows 1, 2, 3, 5, 6, and 7 in the "Team" column are now populated with **NaN**. This result confirms the default behavior of Pandas: it prioritizes a rectangular, structured data representation over replicating the visual appearance of the source Excel sheet. This is the dataset we must now clean for analytical use.

Step 2: Leveraging the Forward Fill Method (`ffill`)

The pattern of missing data—where the value exists at the beginning of a block and is missing in the subsequent rows of that block—is perfectly addressed by a technique known as Forward Filling, or ffill in Pandas terminology. Forward filling works by propagating the last observed valid value forward to fill any subsequent missing values until the next valid value is encountered. This precisely mirrors how merged cells are meant to be interpreted: the merged value applies to all rows it visually encompasses.

To fill in each of these NaN values with the preceding team names, we employ the powerful **fillna()**

function, specifying the forward fill method:

#fill in NaN values with team names
df = df.fillna(method='ffill', axis=0)

#view updated DataFrame
print(df)

Team Player Points Assists
0 Mavericks A 22 4
1 Mavericks B 29 4
2 Mavericks C 45 3
3 Mavericks D 30 7
4 Rockets E 29 8
5 Rockets F 16 6
6 Rockets G 25 9
7 Rockets H 20 12

Analyzing the Result and the `fillna` Parameters

Upon reviewing the final DataFrame output, we can confirm the successful transformation. Every NaN value in the "Team" column has been accurately replaced with the correct team name, corresponding to the structure implied by the original merged cells. The dataset is now clean, rectangular, and ready for advanced statistical analysis, such as group-by operations or performance comparisons between the two teams.

This success hinges on correctly setting two critical parameters within the **fillna()** method: `method='ffill'` and `axis=0`. While `method='ffill'` dictates the logic (forward propagation), the `axis` parameter defines the direction of that propagation, which is essential for processing vertically merged cells.

Understanding the `axis` Parameter in Imputation

The `axis` parameter is fundamental in determining how Pandas applies a function across a DataFrame. Since merged cells in Excel are typically vertical (spanning multiple rows within a single column), we need the filling operation to occur downwards, or along the index. This movement corresponds to `axis=0`.

`axis=0`: Vertical Filling (Downwards). This is the default behavior and is used when processing columns. When merged cells are arranged vertically (spanning multiple rows), setting `axis=0` tells Pandas to look at the preceding row (index) for a valid value and fill the current row's NaN. This is

the standard setting for handling common merged cell layouts.
`axis=1`: Horizontal Filling (Across Columns). If, instead, the merged cells were arranged horizontally (spanning multiple columns within a single row), you would need to fill the NaN values across the columns. In such a rare scenario, you can specify **axis=1**, which directs Pandas to propagate the last valid value horizontally from the left.

In almost all cases involving data sourced from spreadsheets designed for reporting, the data cleaning task requires vertical filling, making `axis=0` the appropriate setting.

Advanced Considerations and Best Practices

While the forward fill method is incredibly effective for standard merged cells, data analysts should be aware of a few advanced considerations to ensure robust data cleaning pipelines. It is important to remember that **fillna()** works on the entire DataFrame by default, meaning it will attempt to fill NaN values in all columns. If other columns legitimately contain missing data that should not be forward-filled (e.g., missing scores), applying the imputation globally might introduce errors.

For cleaner operations and preventing unintended data corruption, it is often best practice to apply the **fillna()** operation selectively to only the column(s) known to contain the merged cell structure. For example, to only target the 'Team' column, the syntax would be modified as follows:

df = df.fillna(method='ffill', axis=0)

This targeted approach ensures that the imputation logic is confined precisely to where the merged cell artifacts exist, leaving other columns untouched, thereby enhancing the reliability and clarity of the data preparation script.

Summary of the Data Cleaning Workflow

Effectively reading an Excel file with merged cells using Pandas is less about specialized import settings and more about smart post-processing. The workflow involves three essential, logical steps:

Import the Data: Use `pd.read_excel()`, which correctly converts the merged cell structure into a DataFrame containing NaN markers.
Identify the Missing Data Pattern: Recognize that the NaN values are not truly missing but are artifacts of the vertical merging, requiring the value above them to be propagated downwards.
Impute the Values: Apply the forward filling method using `.fillna(method='ffill', axis=0)` (ideally on specific columns) to restore the logical structure of the data based on the original spreadsheet design.

By following this robust methodology, data professionals can efficiently convert visually appealing but programmatically complex spreadsheets into clean, analytical DataFrames, significantly streamlining the initial stages of any data science project.