Excel: Move Every Other Row to Column

Authored by

stats writer

November 17, 2025

The Challenge of Data Reorganization in Excel

Working with large datasets in Excel often requires specialized reorganization techniques. A common task encountered by data analysts is the need to separate alternating data points, specifically moving every other row into an adjacent column. This scenario typically arises when data is recorded sequentially but represents two distinct categories--such as alternating measurements, old vs. new product versions, or sequential time stamps that need categorization.

While manual sorting and copying is feasible for small datasets, it quickly becomes inefficient and prone to error when dealing with hundreds or thousands of rows. Fortunately, Microsoft Excel provides powerful logical and positional functions that allow us to automate this process seamlessly. By utilizing a combination of the IF function and row parity checks, we can precisely control which rows are extracted and where they are placed, ensuring data integrity and saving significant time.

This tutorial details the exact methods using specific logical formulas to achieve this sophisticated data rearrangement, focusing on extracting both even-numbered and odd-numbered rows into separate, dedicated columns.

Understanding the Key Excel Functions

To successfully segment our data based on row position, we rely on three foundational Excel functions: IF, ROW, and the parity checking functions (ISEVEN or ISODD). Understanding how these elements interact is crucial for mastering this technique.

ROW function: This simple yet powerful function returns the row number of a reference cell. For example, **=ROW(B2)** returns the value 2. When applied across a range, it provides a unique identifier for every row, which we use as the basis for our conditional logic.
ISEVEN and ISODD functions: These logical functions test whether a number is even or odd, returning **TRUE** or **FALSE**. By nesting the ROW function inside ISEVEN or ISODD, we generate the conditional test needed to filter every other row.
IF function: This is the control structure that executes the desired action. If the row test is **TRUE** (i.e., the row is even or odd, as specified), the formula returns the original cell value; otherwise, it returns an empty string ("").

Formula 1: Isolating Even-Numbered Rows

The first crucial formula is designed to extract data points found on the even-numbered rows within your source range. This is often necessary when one category of data consistently occupies the 2nd, 4th, 6th, and subsequent rows.

The core logic dictates: if the row number of the current cell is even, return the data from that cell; otherwise, return nothing. This operation uses the **ISEVEN** function to perform the parity check.

=IF(ISEVEN(ROW(B2)),B2,"")

In this construction, **ROW(B2)** dynamically determines the row number as the formula is dragged down. **ISEVEN()** checks if this number is even. If it evaluates to TRUE, the **IF** statement returns the content of cell **B2** (the second argument); otherwise, it returns a blank string (the third argument).

Formula 2: Isolating Odd-Numbered Rows

Conversely, if you need to extract the data located on the odd-numbered rows (1st, 3rd, 5th, etc.), we utilize an almost identical structure, substituting **ISEVEN** with **ISODD**. This ensures that the primary data point is extracted, along with every subsequent odd-positioned entry.

This formula is perfect for extracting the first category of data when the two categories alternate strictly by row position. It flips the condition used in Formula 1 to target the inverse set of rows.

=IF(ISODD(ROW(B3)),B3,"")

When this formula is copied down a column, it evaluates the row number of the corresponding source cell (e.g., B3, B4, B5, etc.). If that row number is odd, the value is returned; otherwise, the cell remains empty, successfully isolating the odd-row data.

Detailed Example: Moving Alternating Sales Data

To illustrate the practical utility of these formulas, let us consider a sample dataset. Suppose we have a continuous list of sales figures where the entries alternate between "New Product" sales and "Old Product" sales. Our goal is to separate these two categories into distinct, dedicated columns for easier analysis and reporting.

The original dataset is structured as follows, with the product type implicitly tied to the row order:

	A	B	C	D	E
1	**Product**	**Sales**			
2	A Old	450			
3	A New	225			
4	B Old	220			
5	B New	229			
6	C Old	187			
7	C New	175			
8	D Old	189			
9	D New	140			
10	E Old	150			
11	E New	113			
12	F Old	125			
13	F New	118			
14					
15					

In this example, we assume that the "Old" product sales always fall on the even-numbered rows (Row 2, 4, 6, etc.), and the "New" product sales fall on the odd-numbered rows (Row 3, 5, 7, etc.). We will apply the appropriate formulas to columns C and D, respectively, to perform the necessary data segregation.

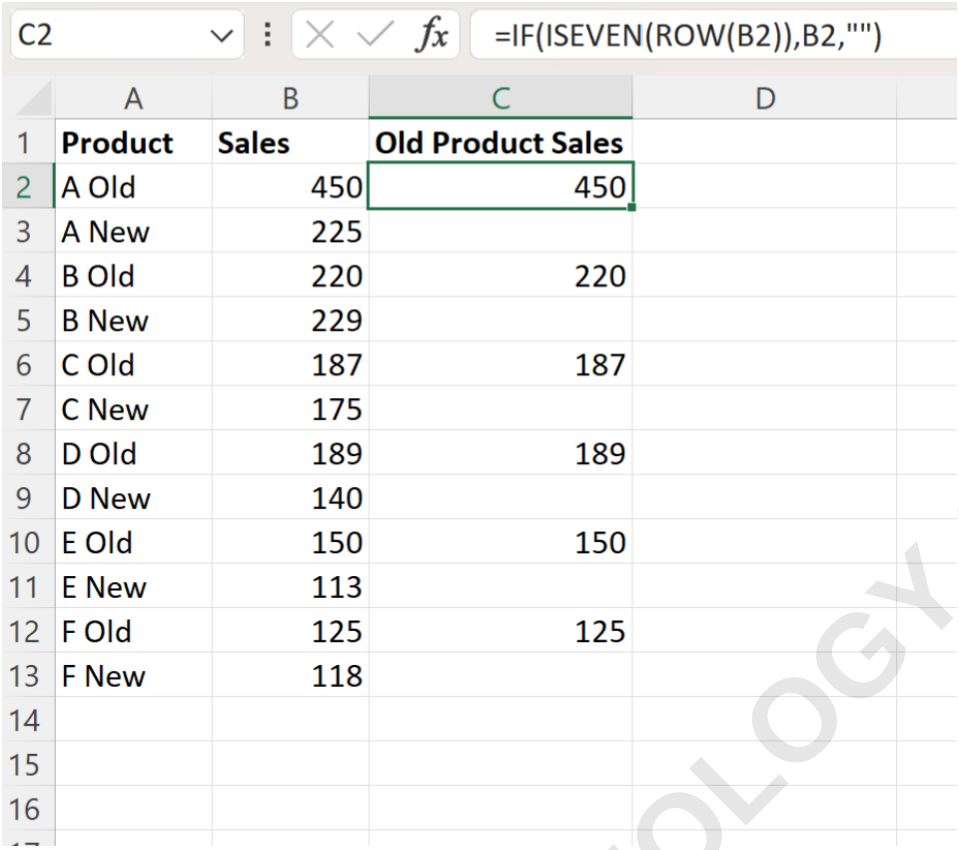
Step 1: Extracting Old Product Sales (Even Rows)

Our initial task is to create the "Old Product Sales" column, which corresponds to the data residing in the even-numbered rows of the original **Sales** column (Column B). We will enter the formula designed for even rows into cell **C2**.

We type the following specific formula into cell **C2**:

=IF(ISEVEN(ROW(B2)),B2,"")

After inputting the formula into **C2**, we must apply it to the entire dataset range. This is achieved by clicking and dragging the fill handle (the small square at the bottom right corner of the selected cell) down through all relevant rows in column C. This action automatically adjusts the cell reference (B2) in the formula for each corresponding row (B3, B4, B5, and so on).



	A	B	C	D
1	**Product**	**Sales**	**Old Product Sales**	
2	A Old	450	450	
3	A New	225		
4	B Old	220	220	
5	B New	229		
6	C Old	187	187	
7	C New	175		
8	D Old	189	189	
9	D New	140		
10	E Old	150	150	
11	E New	113		
12	F Old	125	125	
13	F New	118		
14				
15				
16				

Observe the result: only the values from the **Sales** column that correspond to an even row number have been extracted and displayed in the new **Old Product Sales** column. The rows corresponding to odd numbers remain blank, fulfilling the requirement of moving only the alternate data points.

Step 2: Extracting New Product Sales (Odd Rows)

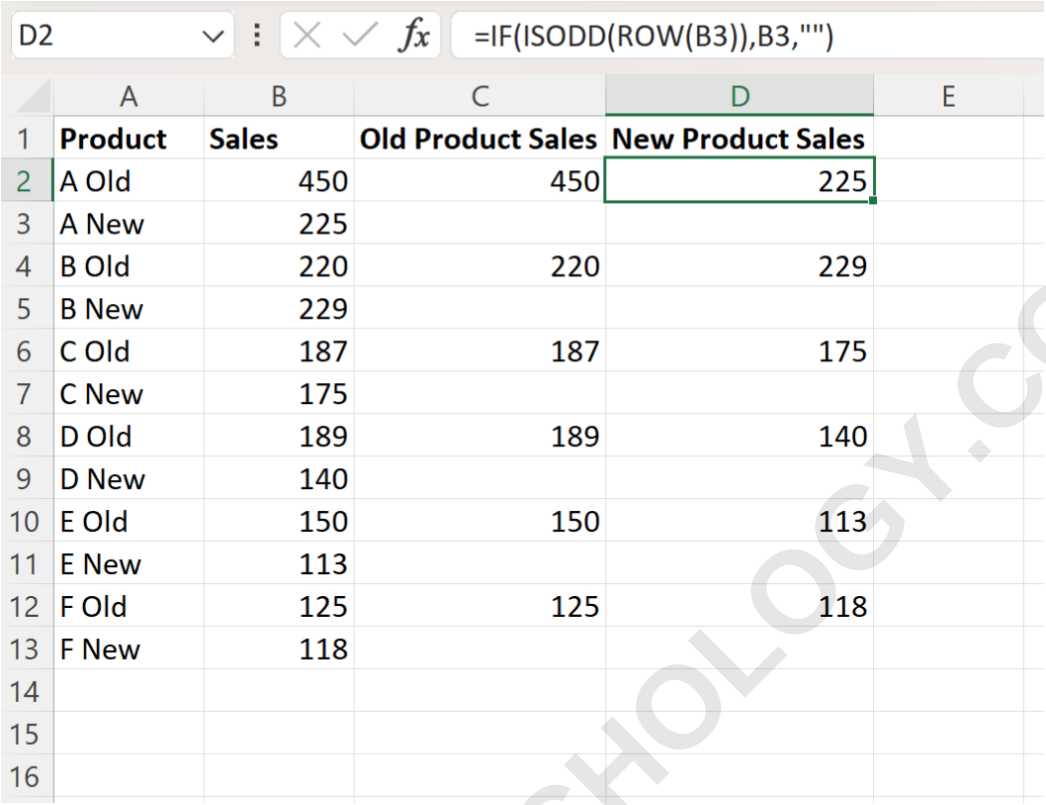
Next, we pivot to extracting the data for "New Product Sales," which we identified as corresponding to the odd-numbered rows in the original dataset. This data will be placed into column D.

We type the following formula, which uses the **ISODD** function, into cell **D2**:

=IF(ISODD(ROW(B3)),B3,"")

Note: While we typically start by referencing B2 when applying the formula to D2, in this specific example, the original formula provided referenced **B3**, which is common if the odd-row extraction needs to align the first entry with the second row (D2 pulls B3, D3 pulls B4, etc.) to skip the header row or align different datasets. For consistency with the provided code snippet, we maintain the **B3** reference, assuming the intention is to check the parity of the row referenced by the cell offset.

We then drag this formula down the remaining cells in column D, allowing Excel's relative referencing system to automatically update the cell addresses for each row:



	A	B	C	D	E
1	**Product**	**Sales**	**Old Product Sales**	**New Product Sales**	
2	A Old	450	450	225	
3	A New	225			
4	B Old	220	220	229	
5	B New	229			
6	C Old	187	187	175	
7	C New	175			
8	D Old	189	189	140	
9	D New	140			
10	E Old	150	150	113	
11	E New	113			
12	F Old	125	125	118	
13	F New	118			
14					
15					
16					

The final result shows that the data has been successfully bifurcated. The **New Product Sales** column now contains only the values from the odd-numbered rows of the original **Sales** column, completing the reorganization task. This approach provides a robust and dynamic solution for managing interleaved data structures.

Considering Alternatives: Power Query or VBA

While the combination of IF, ISEVEN, and ROW functions is highly effective and requires no external tools, it is important to acknowledge alternative methods for data reorganization in Excel, especially when dealing with massive datasets or complex transformation requirements.

One popular alternative is **Power Query** (or Get & Transform Data), which is built into modern Excel versions. Power Query allows users to easily add an Index Column and then filter based on the modulus of that index (e.g., Index mod 2 = 0 for even rows, Index mod 2 = 1 for odd rows). This method avoids the need for helper columns and results in static, non-formulaic data output, which can be preferable for performance.

Another option involves using **VBA (Visual Basic for Applications)** scripting. A simple loop can

be written to iterate through the source column and, based on the loop counter's parity, copy the cell value to the target column. While VBA offers ultimate flexibility, it requires programming knowledge and may not be accessible to all users. For the specific task of separating every other row, the formulaic approach demonstrated here remains the simplest and most robust solution for most intermediate Excel users.

Summary and Best Practices

The technique of using nested logical functions (IF, ISEVEN/ISODD, and ROW function) provides an efficient and dynamic way to separate alternating rows in Excel. This method is particularly valuable when the underlying data source is subject to change, as the formulas automatically recalculate, maintaining the segregation without manual intervention.

Key takeaways for effective implementation include:

Know Your Parity: Clearly identify whether the data you wish to extract resides in even or odd-numbered rows to select the correct parity function (**ISEVEN** or **ISODD**).
Ensure Relative Referencing: When dragging the formula down, ensure that the cell reference inside the **ROW()** function (e.g., B2) is relative, allowing it to accurately assess the position of each cell in the original dataset.
Handling Blanks: The use of **""** (an empty string) as the **value_if_false** argument in the IF function prevents the target column from being populated with unnecessary values, resulting in a cleaner output that is ready for further processing or analysis.