Restructuring Data: Moving Alternating Rows to Columns in Excel

Authored by

Mohammed looti

November 10, 2025

Introduction to Strategic Data Restructuring in Excel

In the realm of advanced data analysis and preparation, practitioners using Microsoft Excel frequently encounter complex scenarios that demand the strategic rearrangement of data structures. One common, yet often challenging, requirement is the ability to extract or move alternating rows--such as every second row--into an adjacent column. This operation is indispensable when dealing with source datasets containing vertically paired information, where, for instance, one row holds a product attribute and the subsequent row holds its corresponding value. Our focus here is on separating interleaved data based on inherent patterns, such as "New" versus "Old" product categories that fall on alternating lines.

Attempting to perform this task manually on extensive spreadsheets is not only incredibly time-consuming but also highly susceptible to human error. Fortunately, Excel provides a potent combination of logical and informational functions that enable the automated identification and segregation of rows based purely on their positional index within the sheet. By leveraging the inherent properties of the spreadsheet grid--specifically the row number--we can construct robust formulas capable of achieving precise, automated data manipulation efficiently.

The core methodology relies on mathematically checking the parity (meaning the evenness or oddness) of the row index. This comprehensive guide offers a detailed, step-by-step walkthrough of the formulas required to execute this extraction seamlessly, ensuring your data transformation process is both efficient and highly accurate. To achieve the desired outcome, we will utilize a powerful sequence of nested functions: the conditional function IF, the positional function **ROW**, and the specialized parity functions **ISEVEN** and **ISODD**.

Mastering the Formulas for Alternating Row Extraction

Successfully moving alternating rows into a new column hinges upon mastering two distinct conditional formulas. These formulas utilize core logical components to determine if a specific cell's row index satisfies a predetermined parity condition. The primary mechanism involves the ROW function, which dynamically returns the current row number of the cell being referenced, paired with the specialized parity functions (**ISEVEN** and **ISODD**) to test that numerical output.

Presented below are the two foundational formulas tailored specifically for extracting data based on its row position. These robust formulas are designed to be entered once into the top cell of your designated target column (e.g., cell C2) and then copied down to apply the precise logic across the entirety of the source dataset. It is critical to ensure that the cell reference used within the formula (such as B2 or B3) correctly aligns with the first cell of the source column you intend to analyze and separate.

Formula 1: Isolating Data from Even-Numbered Rows

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

This formula meticulously checks if the row number corresponding to the reference cell B2 is an even number. If this condition evaluates to **TRUE**, the formula returns the actual value found in cell B2. If the condition is not met (i.e., the row is odd), it returns an empty string (`""`), thereby leaving the target cell blank and filtering out the unwanted rows.

Formula 2: Isolating Data from Odd-Numbered Rows

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

Conversely, this second formula executes the exact same logical test but specifically seeks out odd-numbered rows. If the row number corresponding to the reference cell B3 is odd, the value of B3 is returned. The strategic choice between Formula 1 and Formula 2 depends entirely on which specific set of alternating data points--the first row of a pair or the second row of a pair--you need to isolate and transfer to the new column.

Practical Application: Separating Even-Numbered Data Points

To demonstrate the power of this technique, let us consider a typical scenario involving a combined sales dataset. In this example, the data is structured such that "New" product sales consistently occupy the odd-numbered rows, while "Old" product sales are found in the even-numbered rows, creating a distinct interleaved pattern. Our defined objective is to cleanly separate all "Old Product Sales" data into a new, dedicated column (Column C).

The following illustration displays the initial structure of our sales data, which is currently contained entirely within Column B:

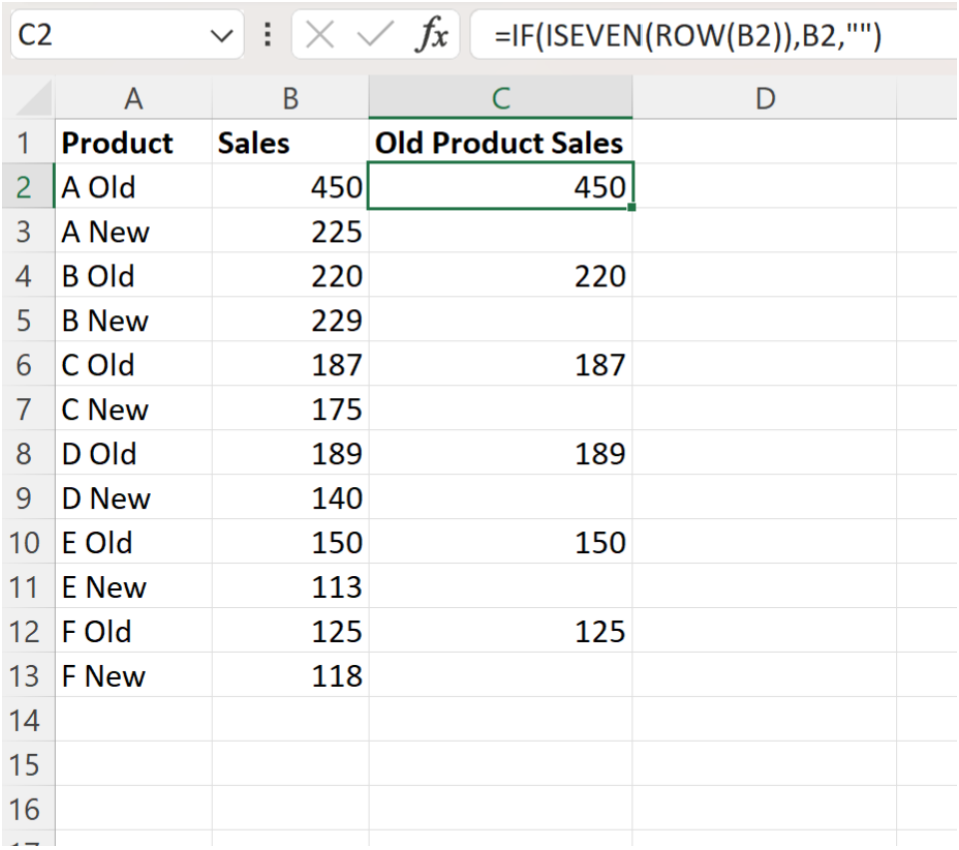
	A	B	C	D	E
1	**Product**	**Sales**			
2	A Old	450			
3	A New	225			
4	B Old	220			
5	B New	229			
6	C Old	187			
7	C New	175			
8	D Old	189			
9	D New	140			
10	E Old	150			
11	E New	113			
12	F Old	125			
13	F New	118			
14					
15					

Since the "Old" sales data resides in the even-numbered rows (B2, B4, B6, and so on), we must leverage the **ISEVEN** function to accurately filter and extract only this specific subset of information. We will place the resulting output into a new column labeled "Old Product Sales."

To initiate this extraction, we must type the following formula into cell **C2**, ensuring that the cell reference `B2` correctly points to the very beginning of the source data range:

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

After successfully inputting the formula into cell **C2**, the final step is to apply this logic across the entire data range. We achieve this by clicking and dragging the fill handle located at the bottom right corner of cell **C2** down to cover all corresponding rows in Column C. Excel's powerful relative referencing feature automatically ensures that the reference B2 adjusts sequentially to B3, B4, B5, and so on, meticulously testing the parity of each corresponding row index.



	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				

As clearly illustrated in the resulting visualization, the values that correspond only to the even-numbered rows have been successfully isolated and transferred into the designated **Old Product Sales** column. The rows corresponding to the odd-numbered data points are left blank. This effective separation preserves the integrity of the original data while providing a clean, restructured view that is immediately ready for further detailed analysis.

Dissecting the ISEVEN and ROW Formula Logic

A thorough understanding of the nested mechanism within the formula `=IF(ISEVEN(ROW(B2)),B2,"")` is absolutely crucial for achieving confident and accurate data manipulation. This powerful formula relies on the sequential, nested execution of three distinct functions. The process begins with the ROW function, which operates on the cell reference B2. Critically, as the formula is copied down the column, this reference automatically adjusts to B3, B4, B5, and subsequent cells, always returning the current physical row number for each individual calculation. For example, when the formula resides in cell C4, `ROW(B4)` returns the integer value 4.

The numerical result generated by the **ROW** function is immediately passed as an argument to the ISEVEN function. This function performs a straightforward boolean test: Is the input number perfectly divisible by two? If the row number is 4 (even), **ISEVEN** returns the logical value **TRUE**. Conversely, if the row number is 5 (odd), **ISEVEN** returns the logical value **FALSE**. This

TRUE/FALSE output forms the central conditional test that dictates the formula's ultimate action.

Finally, the entire expression is encapsulated within the IF function, which directs the output based on the result of the **ISEVEN** test. The standard syntax for the **IF** function is `IF(logical_test, value_if_true, value_if_false)`.

The specific arguments used in our data extraction formula are as follows:

Logical Test: `ISEVEN(ROW(B2))` determines if the current row number is even.
Value if True: `B2`. If the test passes, the formula returns the actual content from the source cell.
Value if False: `""`. If the test fails, the formula returns an empty string, ensuring the cell remains visually blank.

This structured, conditional approach guarantees that data is only displayed in the target column when the corresponding source row meets the mathematically specified even-numbered criterion, achieving perfect segregation.

Practical Application: Isolating Odd-Numbered Data Points

In stark contrast to the previous example, let us now shift our objective to isolating the "New Product Sales" data. Given that this information is located consistently in the odd-numbered rows (B3, B5, B7, etc., assuming row 1 holds headers), we must strategically pivot our formula to employ the **ISODD** function. This critical inversion allows us to capture the complementary set of data points that the **ISEVEN** function intentionally skipped, thereby completing the separation of the interleaved records.

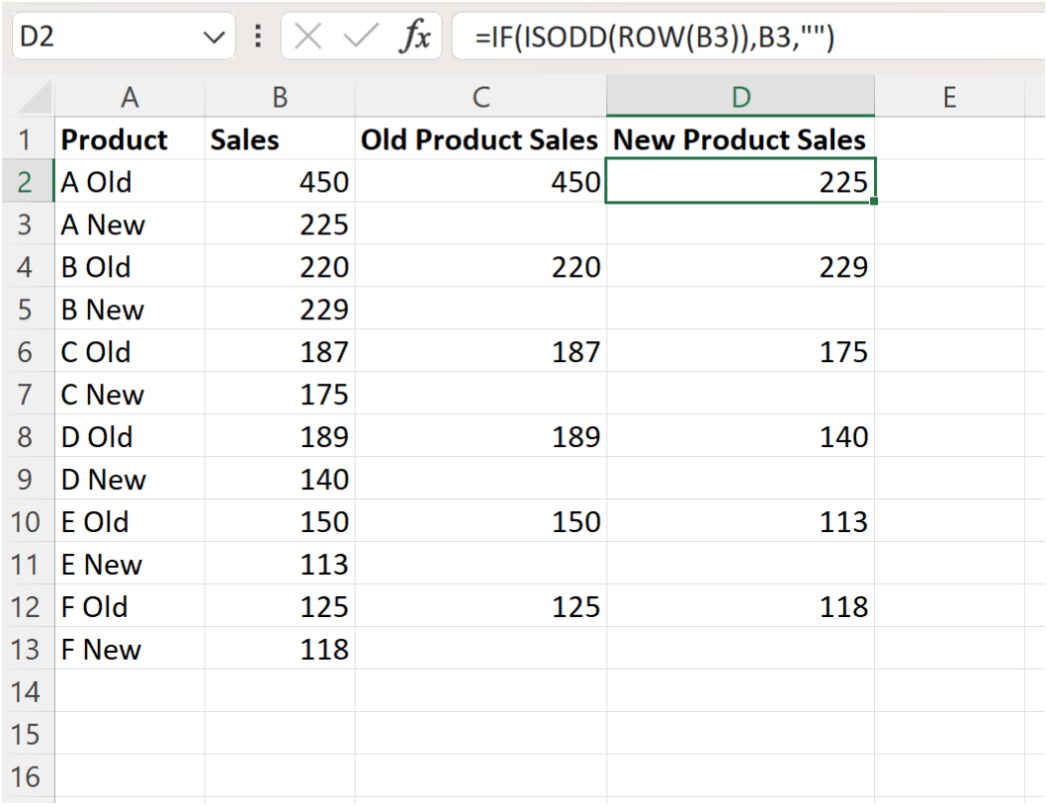
For this specific isolation task, we will target a new column, Column D, and label it "New Product Sales." It is vital to ensure that the starting reference cell within our formula aligns correctly with the very first data point we wish to evaluate. For consistency with our alternating data pattern, we are isolating the data that starts on the odd rows below the header, starting the formula in D2 but referencing B3.

We initiate the extraction of the odd-row data by typing the following formula into cell **D2**:

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

A key point of consideration here is the starting cell reference inside the **ROW** function: `ROW(B3)`. If we were to start the formula in D2 using `ROW(B2)`, it would return the number 2 (even), causing the **ISODD** test to fail for the first data row. By referencing `B3`, we ensure that the parity test begins checking from the correct sequential starting point, allowing the formula to capture the data that begins the odd-row pattern.

Once the formula is entered into **D2**, we click and drag the formula down to instantly populate the remainder of Column D, allowing the relative cell references to update automatically and apply the odd-row logic throughout the range.



	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 resulting visualization confirms the successful transfer: every other row--specifically those corresponding to the odd-numbered rows in the source data--has been accurately transferred into the **New Product Sales** column, thereby achieving the desired data segregation based on positional criteria.

Understanding the ISODD and ROW Implementation

The formula designed for odd-row extraction, `=IF(ISODD(ROW(B3)),B3,"")`, operates functionally identically to its even counterpart, yet the core logical condition is precisely inverted through the strategic use of the ISODD function. This inversion is exactly what allows us to target the complementary set of data points, guaranteeing that the entire source dataset can be cleanly and reliably split into two separate columns based purely on their position.

As the formula is copied down Column D, the nested ROW function dynamically returns the physical row number associated with the current referenced cell (shifting sequentially from B3 to B4, B5, and so on). This numerical output is then immediately evaluated by the **ISODD** function, which returns **TRUE** only if the row number is an odd integer.

For example, when the formula evaluates cell B5, `ROW(B5)` returns 5. Since 5 is odd, `ISODD(5)` returns **TRUE**. The encapsulating IF function then executes the "value if true" argument, which is the content of cell B5. Conversely, when the formula evaluates cell B6, `ROW(B6)` returns 6. Since 6 is even, `ISODD(6)` returns **FALSE**, prompting the **IF** function to execute the "value if false" argument (`""`), resulting in a blank cell. This powerful use of conditional logic based on row position is a staple technique in Excel for restructuring complex data without resorting to complicated VBA code or specialized add-ins.

Conclusion and Advanced Data Wrangling Applications

We have successfully demonstrated an efficient and reliable method for moving every other row into a new column within Excel. This technique is achieved through the strategic combination of the **IF**, ROW function, **ISEVEN**, and **ISODD** functions. This skill is invaluable whenever you need to reorganize vertically stacked data records into a more coherent, analysis-ready horizontal format. The critical insight derived from this tutorial is the use of the **ROW** function to provide a dynamic numerical index that the parity functions can accurately test.

While this specific tutorial focused on the basic segregation of data into two alternating columns, this foundational logic can be readily extended to far more complex sorting or filtering tasks. For example, by modifying the conditional test within the **IF** statement, one could filter data based on multiples of three (utilizing the **MOD** function) or apply conditional formatting exclusively to every third or fourth row.

Mastery of these fundamental logical and positional functions significantly enhances your overall capability to perform advanced data wrangling operations both efficiently and effectively within any spreadsheet environment.

Additional Resources for Excel Mastery

For those interested in exploring more sophisticated data manipulation techniques and expanding their knowledge of efficient spreadsheet operations, the following tutorials provide guidance on other common data tasks in Excel: