# How to copy rows to another sheet based on criteria in VBA?

Authored by

**stats writer**

November 19, 2025

When managing large datasets in VBA, a common requirement is the ability to selectively transfer information between different Worksheets based on specific criteria. Manually filtering and copying rows can be tedious and prone to human error, especially when dealing with dynamic data that updates frequently. Fortunately, Visual Basic for Applications provides a robust and efficient solution to automate this process. By utilizing structured programming techniques, we can build a macro that intelligently scans a source sheet and extracts only the rows meeting predefined conditions, placing them neatly into a designated destination sheet.

This approach centers on combining structured iteration, often through a looping construct, with conditional logic using `If` statements. The core mechanism involves iterating row by row across the source data. Within each iteration, the macro evaluates whether the cell value in a specific column satisfies the defined criteria. If the condition is met, the entire row is copied. If not, the macro simply moves on to the next row, ensuring that only relevant records are transferred.

Mastering this technique is fundamental for automating data management tasks, allowing users to consolidate, archive, or analyze subsets of data without manual intervention. This article will walk you through the precise VBA syntax required to implement conditional row copying, ensuring your data workflows are both fast and accurate.

## Understanding the Core Mechanism: Iteration and Comparison

The foundation of conditional row copying in VBA relies heavily on the concept of iteration and conditional flow control. We must systematically examine every record in the source Worksheets to determine its suitability for transfer. This systematic inspection is achieved using a `For i = 1 To LastRow` loop, where `LastRow` is dynamically calculated to ensure the macro processes the entire dataset, regardless of its size.

Within this looping structure, the critical decision-making step is the `If...Then` statement. This statement checks the value of a specific cell (e.g., in Column A of the current row `i`) against the required criteria. For example, if we are filtering sales data, the criteria might be checking if the region is "North" or if the sales amount exceeds a certain threshold.

Once a row satisfies the condition specified in the `If` statement, the macro executes the copy operation. We use the Range.Copy method, which is highly efficient for transferring data. Crucially, we must also maintain a separate counter variable (often named `j`) to track the next available row in the destination sheet. This ensures that the copied rows are appended sequentially without overwriting existing data.

## Implementing the Core VBA Syntax for Conditional Copying

To efficiently copy rows from a source sheet (e.g., Sheet1) to a destination sheet (e.g., Sheet2)

based on a specific value in a key column, you can employ the following standardized VBA structure. This template handles identifying the limits of the source data and determining the correct starting point for pasting in the destination sheet before initiating the row-by-row inspection.

```
Sub CopyToAnotherSheet()

Dim LastRow As Long

'Find last used row in a Column A of Sheet1
With Worksheets("Sheet1")
LastRow = .Cells(.Rows.Count, "A").End(xlUp).Row
End With

'Find first row where values should be posted in Sheet2
With Worksheets("Sheet2")
j = .Cells(.Rows.Count, "A").End(xlUp).Row + 1
End With

'Paste each row that contains "Mavs" in column A of Sheet1 into Sheet2
For i = 1 To LastRow
With Worksheets("Sheet1")
If .Cells(i, 1).Value = "Mavs" Then
.Rows(i).Copy Destination:=Worksheets("Sheet2").Range("A" & j)
j = j + 1
End If
End With
Next i

End Sub
```

This macro, `CopyToAnotherSheet`, is designed specifically to filter data based on the text value "Mavs" found in the first column (Column A) of the source sheet. The rows that successfully meet this filtering criteria are then appended to the next available row in the target Worksheets, ensuring a clean transfer of selected data. The efficiency of this code stems from using the `End(xlUp)` property to dynamically determine data boundaries, making the macro robust across datasets of varying sizes.

## Step-by-Step Breakdown of the Script Logic

Understanding the flow of execution is key to customizing and troubleshooting your VBA code. The script starts by declaring necessary variables, specifically `LastRow` (Long) for the source data

boundary, and implicitly using `i` (the looping counter) and `j` (the destination row counter).

**Determine Source Boundary:** The first `With Worksheets("Sheet1")` block calculates `LastRow`. It efficiently navigates from the very bottom of the sheet upwards in Column A (`.Cells(.Rows.Count, "A").End(xlUp)`) to find the last occupied cell, thereby defining the range the loop needs to process.

**Determine Destination Start Row:** The second `With Worksheets("Sheet2")` block determines `j`, which is the starting row for pasting. It finds the last used row in Sheet2 and adds one (`+ 1`), guaranteeing that the copied data starts immediately below any existing content.

**Iterate and Evaluate:** The `For i = 1 To LastRow` loop begins the row-by-row inspection. For each row `i` in Sheet1, the script executes the `If` condition: `If .Cells(i, 1).Value = "Mavs" Then`. This condition checks if the value in Column 1 (A) of the current row matches the string "Mavs".

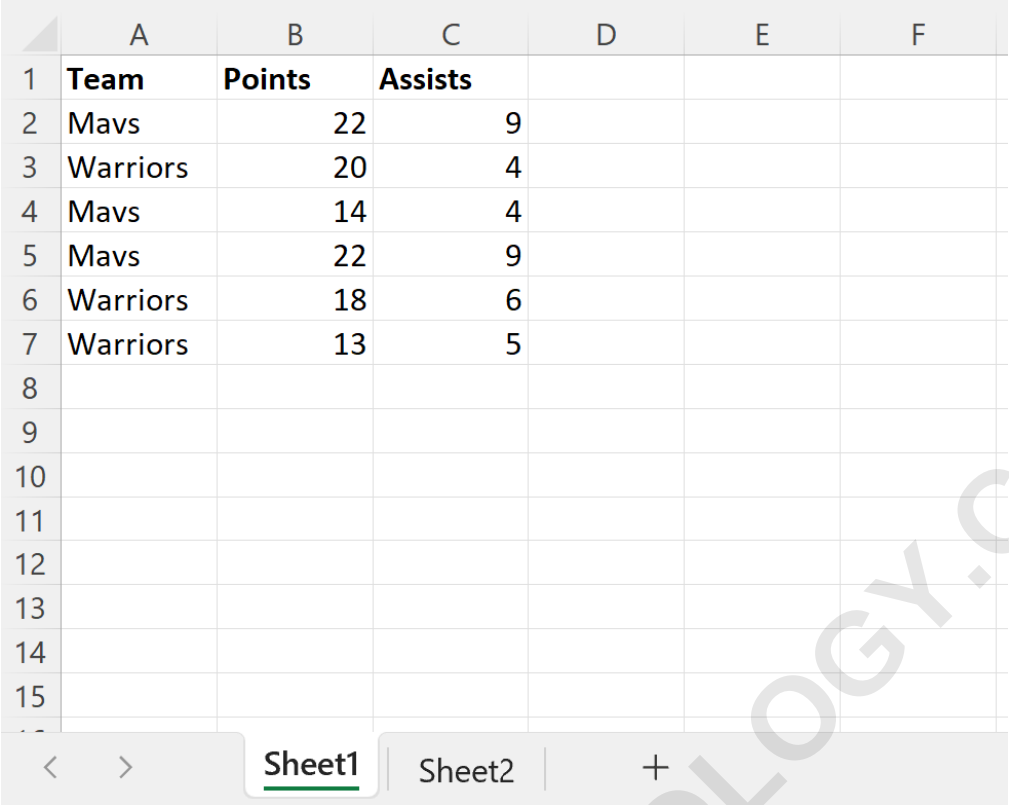
**Copy and Increment:** If the condition is **True**, the entire row `i` is copied using the Range.Copy method, specifying Sheet2, Column A, and row `j` as the destination. Immediately following the copy operation, the destination counter `j` is incremented (`j = j + 1`). This crucial step ensures that the next row that meets the criteria will be pasted into the subsequent empty row in Sheet2, maintaining proper data stacking.

## Example Scenario: Setting Up Data for Filtering

To illustrate this conditional copying mechanism, let us consider a practical dataset involving basketball player statistics. We will utilize two Worksheets: **Sheet1**, which serves as our master data source containing information on players from various teams, and **Sheet2**, which is the destination sheet where we wish to consolidate data based on our filter criteria.

Suppose **Sheet1** contains a comprehensive roster, including columns for Player Name, Team, Points, and Assists. This sheet represents our primary source, which the VBA script will iterate through. We will focus our filtering efforts on the 'Team' column, which corresponds to Column A in this particular layout.

Examine the initial data setup for **Sheet1**, noting the mix of teams present:

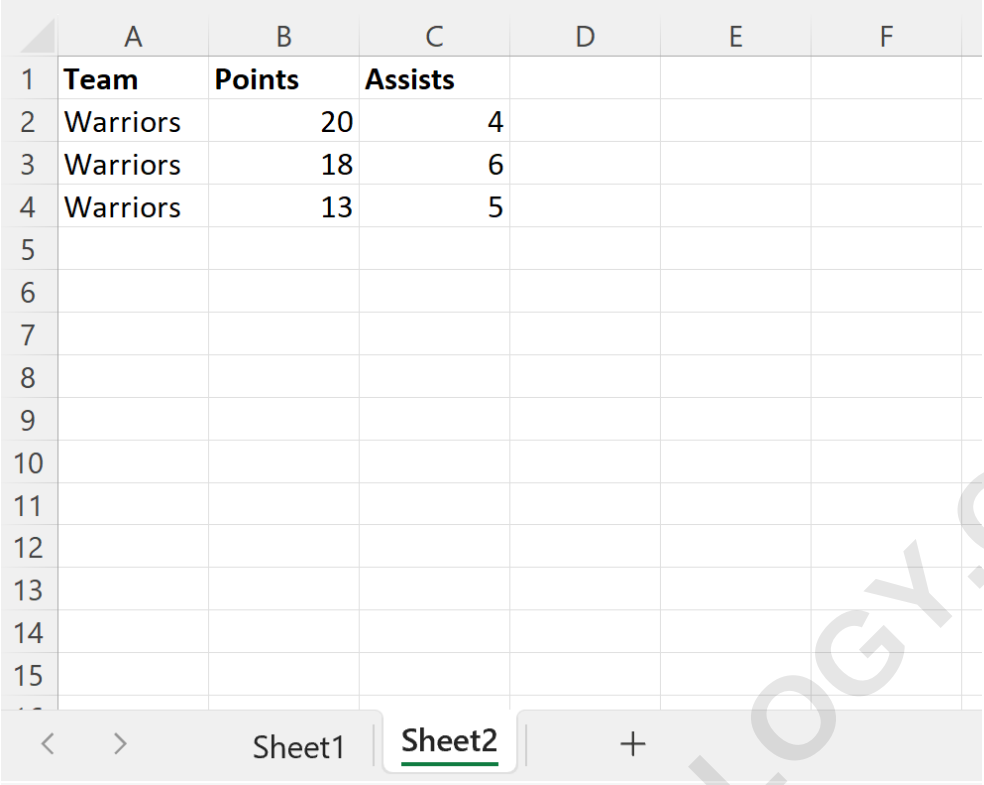


| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | **Team** | **Points** | **Assists** | | | |
| 2 | Mavs | 22 | 9 | | | |
| 3 | Warriors | 20 | 4 | | | |
| 4 | Mavs | 14 | 4 | | | |
| 5 | Mavs | 22 | 9 | | | |
| 6 | Warriors | 18 | 6 | | | |
| 7 | Warriors | 13 | 5 | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |

Sheet1 Sheet2

Now, consider the initial state of **Sheet2**. This sheet already holds some data, specifically records pertaining only to the "Warriors" team. It is essential that our macro correctly identifies the last row of this existing data so that the new, filtered data (the "Mavs" players) is appended seamlessly, rather than overwriting existing records.

The initial layout of **Sheet2** is shown below:



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | **Team** | **Points** | **Assists** | | | |
| 2 | Warriors | 20 | 4 | | | |
| 3 | Warriors | 18 | 6 | | | |
| 4 | Warriors | 13 | 5 | | | |
| 5 | | | | | | |
| 6 | | | | | | |
| 7 | | | | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |

Sheet1 Sheet2 +



## Applying the Macro for Conditional Transfer

Our objective is clear: we need to isolate all rows in **Sheet1** where the Team name (found in Column A) is equal to **Mavs** and transfer these complete rows into the immediate next available space in **Sheet2**. This task requires deploying the previously discussed conditional VBA macro.

We insert the following code into a standard module within the VBA Editor. Note that the criteria is hardcoded as "Mavs" and the column reference is `.Cells(i, 1)`, which specifically targets Column A for the conditional check.

**Sub CopyToAnotherSheet()**

Dim LastRow As Long

'Find last used row in a Column A of Sheet1
With Worksheets("Sheet1")
LastRow = .Cells(.Rows.Count, "A").End(xlUp).Row
End With

'Find first row where values should be posted in Sheet2
With Worksheets("Sheet2")
j = .Cells(.Rows.Count, "A").End(xlUp).Row + 1

```
End With

'Paste each row that contains "Mavs" in column A of Sheet1 into Sheet2
For i = 1 To LastRow
With Worksheets("Sheet1")
If .Cells(i, 1).Value = "Mavs" Then
.Rows(i).Copy Destination:=Worksheets("Sheet2").Range("A" & j)
j = j + 1
End If
End With
Next i

End Sub
```

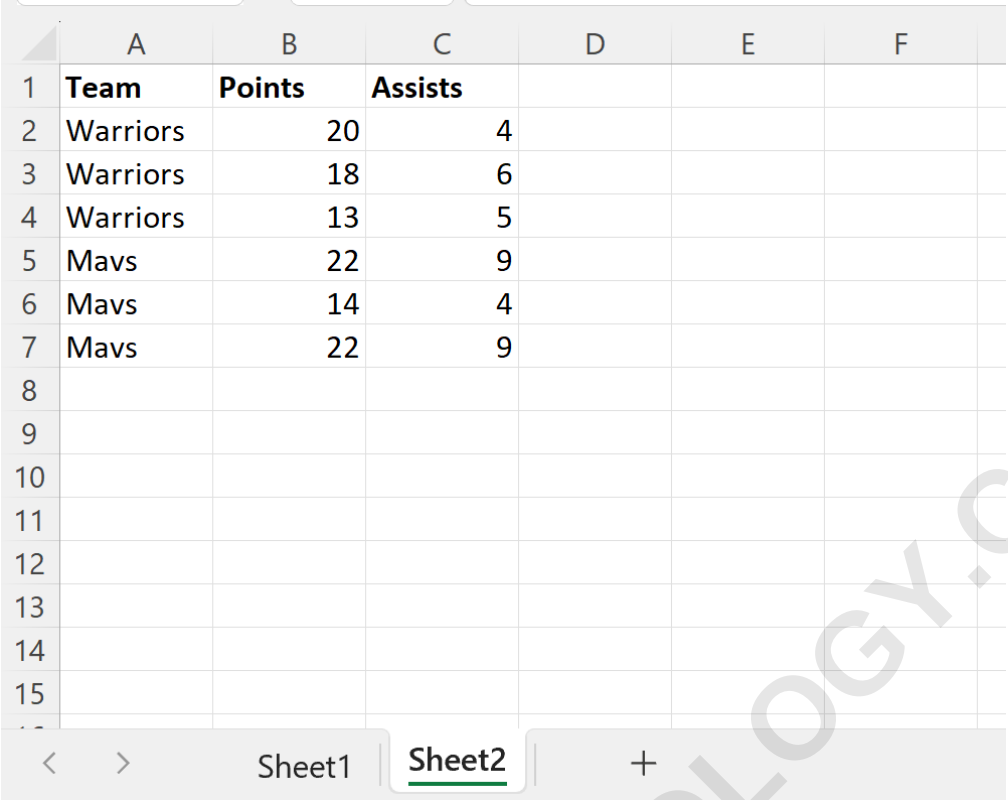
Upon execution, the looping structure iterates through every row of Sheet1. For rows 3 and 6, where the team is "Mavs," the `If` condition is met. These rows are then copied using the Range.Copy method and directed to the location specified by the dynamically increasing counter `j`, ensuring precise placement in Sheet2 directly after the existing "Warriors" data.

## Reviewing the Final Output

After running the `CopyToAnotherSheet` macro, we observe the changes reflected in **Sheet2**. The macro successfully identified the necessary rows in **Sheet1** based on the "Mavs" criteria and appended them to the existing dataset in **Sheet2**.

The resulting data in **Sheet2** now contains the original "Warriors" data followed immediately by the newly transferred "Mavs" data. This demonstrates the effectiveness of dynamically calculating the destination row using the `j` variable, preventing data overlap.

The resulting Worksheets content appears as follows:



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | **Team** | **Points** | **Assists** | | | |
| 2 | Warriors | 20 | 4 | | | |
| 3 | Warriors | 18 | 6 | | | |
| 4 | Warriors | 13 | 5 | | | |
| 5 | Mavs | 22 | 9 | | | |
| 6 | Mavs | 14 | 4 | | | |
| 7 | Mavs | 22 | 9 | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |

Sheet1 Sheet2

As clearly visible, each row from **Sheet1** where the Team column was equal to **Mavs** has been accurately pasted into the next available rows in **Sheet2**. This automated process ensures data integrity and saves considerable time compared to manual selection and copying, particularly in large workbooks. For further information regarding the technical parameters and usage of the core copying function, you can consult the official documentation for the VBA **Copy** method.

By mastering conditional VBA looping and the Range.Copy method, you gain significant control over data flow within your Excel applications.