Learn How to Calculate Averages by Cell Color in Excel Using VBA

Authored by

Mohammed looti

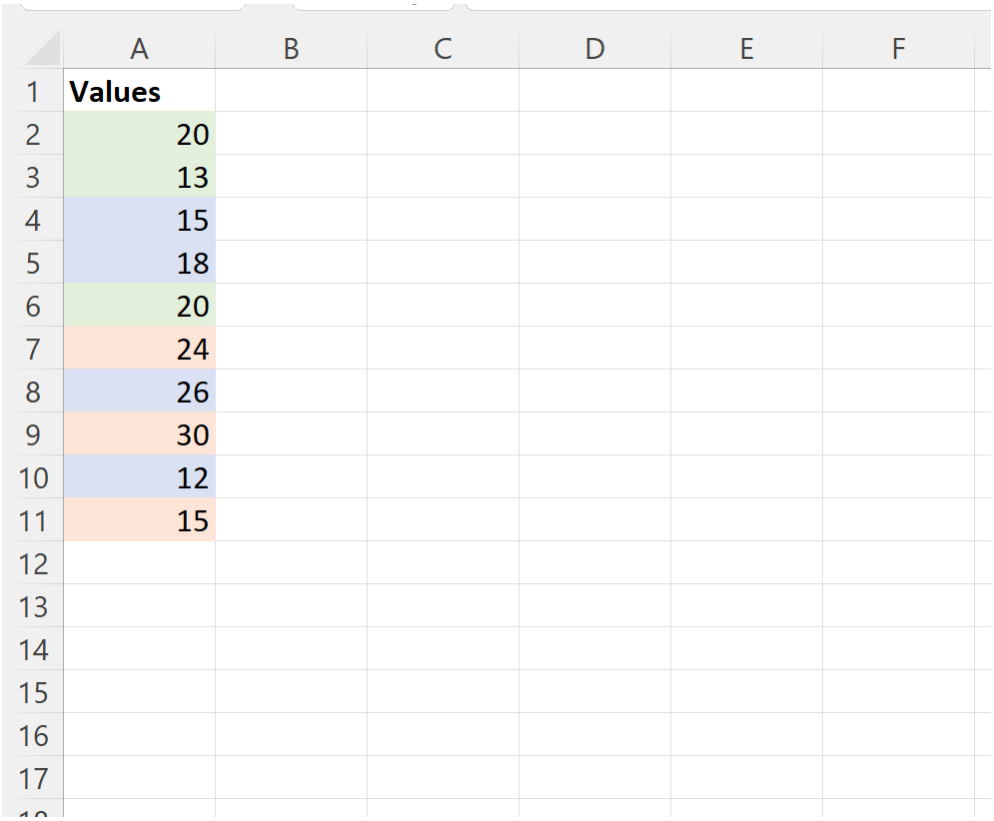
November 10, 2025

The Challenge of Averaging by Cell Color in Excel

Oftentimes, complex data analysis requires calculating statistical measures, such as the average, based on specific formatting criteria. A common request in Excel is to calculate the average of values contained within cells that share a particular background color. Standard built-in functions, such as **AVERAGEIF** or **SUMIF**, are designed to work with logical conditions based on cell content (text, numbers, dates) but cannot directly reference cell formatting attributes like color or font style.

To overcome this limitation, we must employ a custom-written function, commonly referred to as a Macro, developed using **Visual Basic for Applications (VBA)**. While the prospect of writing code might seem daunting to non-programmers, the process is highly standardized and straightforward. This guide will walk you through the precise steps required to implement a powerful custom function that enables reliable averaging based purely on cell color.

For instance, consider a scenario where we have the following dataset, and we need to calculate the average for values grouped by their corresponding cell colors:



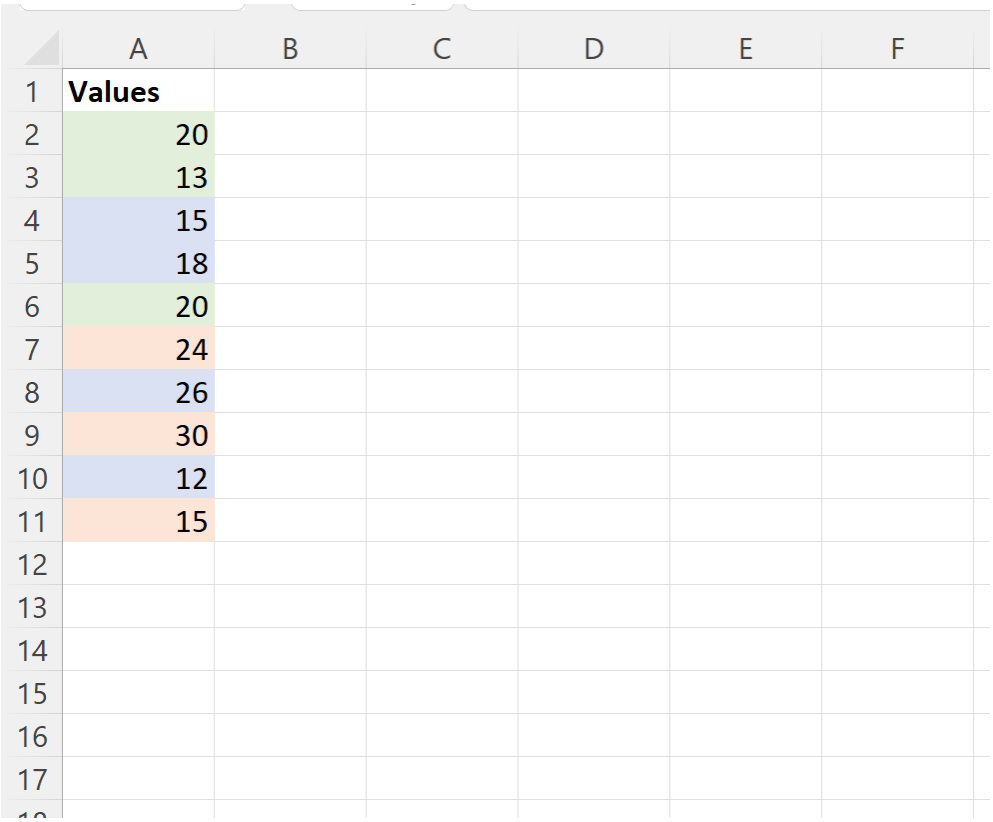
	A	B	C	D	E	F
1	**Values**					
2	20					
3	13					
4	15					
5	18					
6	20					
7	24					
8	26					
9	30					
10	12					
11	15					
12						
13						
14						
15						
16						
17						

By implementing a dedicated **VBA** script, we can create a reusable tool that handles this calculation efficiently, allowing for dynamic updates whenever the data or colors change. The following steps detail the entire implementation process, ensuring a smooth transition from data setup to final result.

Step 1: Preparing Your Data and Enabling Advanced Features

The initial phase involves two critical actions: entering your raw data into the worksheet and ensuring that the necessary tools for code implementation are accessible within the Excel interface.

First, accurately input the data values into your Excel sheet. For this example, we will use the following structure, where the cells have already been conditionally or manually formatted with distinct background colors:



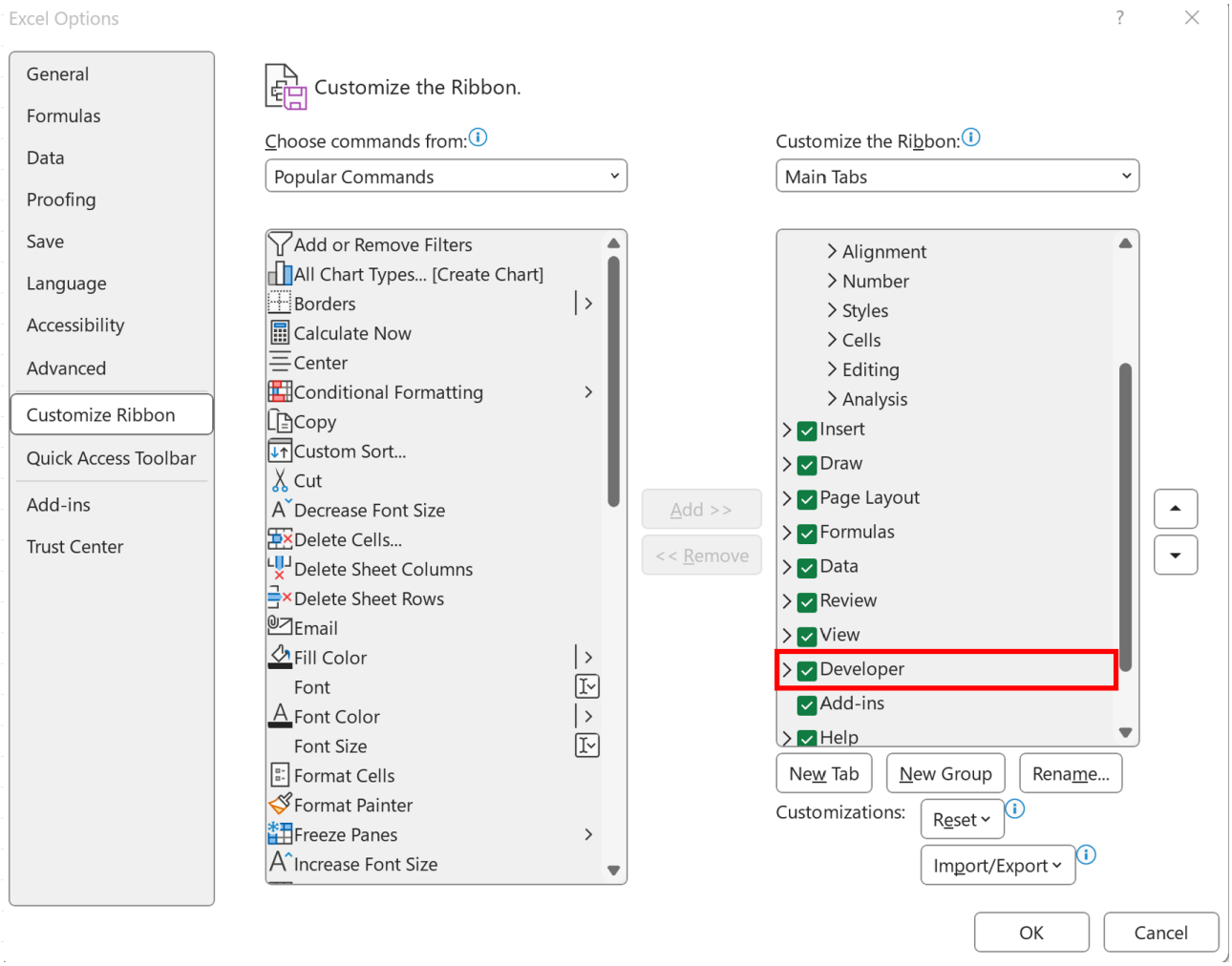
	A	B	C	D	E	F
1	**Values**					
2	20					
3	13					
4	15					
5	18					
6	20					
7	24					
8	26					
9	30					
10	12					
11	15					
12						
13						
14						
15						
16						
17						

Second, and most importantly, we must ensure that the **Developer** tab is visible on the main ribbon. This tab contains the essential tools needed to access the **Visual Basic Editor (VBE)** and manage code. If the tab is not currently displayed, follow these steps:

Click the **File** tab located in the upper-left corner of the application window.
Select **Options** from the menu to open the Excel Options dialog box.
Navigate to the **Customize Ribbon** section on the left panel.
Under the list labeled **Main Tabs**, locate and check the box corresponding to the Developer tab.
Click **OK** to save the changes and close the dialog box.

Upon completion, the **Developer** tab will appear on your Excel ribbon, granting access to the

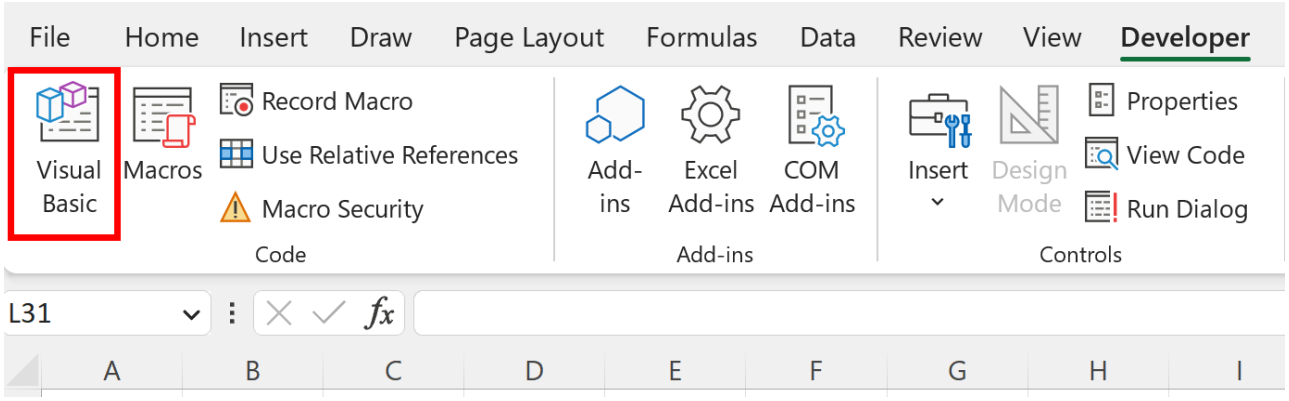
advanced tools required for the subsequent steps. The visual confirmation of this process is shown below:



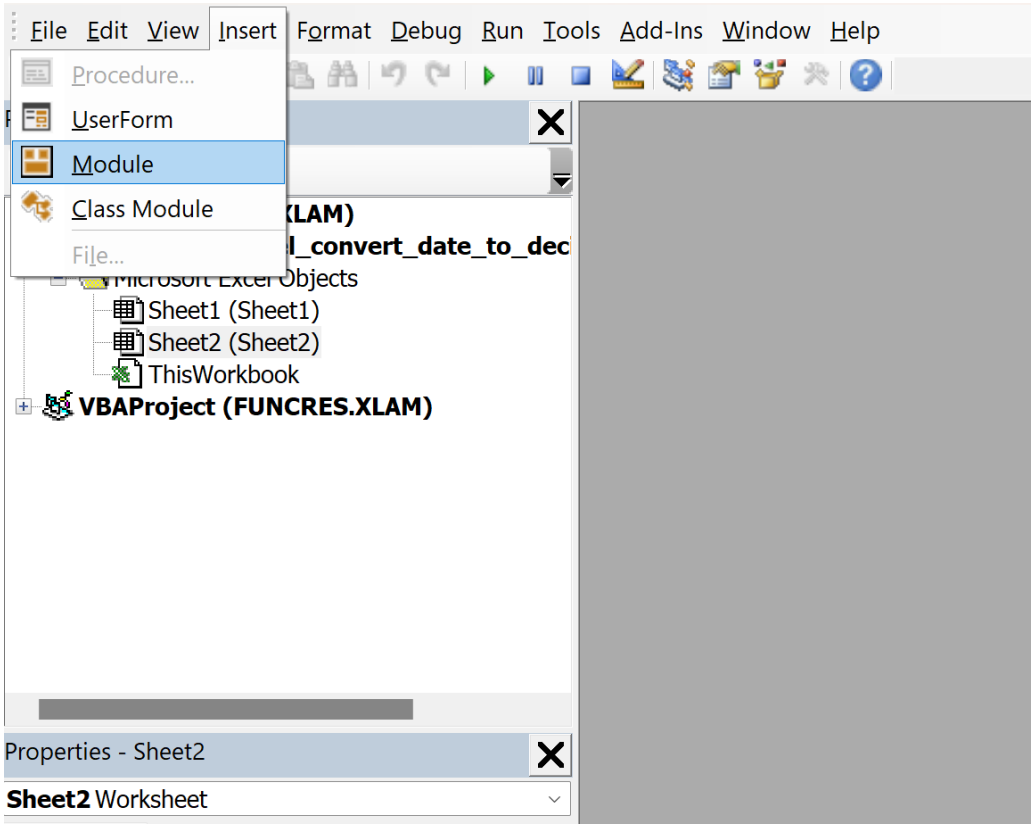
Step 2: Creating a Macro Using VBA

With the **Developer** tab enabled, we can now proceed to create the custom function using the **Visual Basic Editor (VBE)**. This function will be designed to iterate through a specified range of cells, identify the background color index of each cell, and calculate the sum and count only for those cells matching a reference color.

Begin by clicking the **Developer** tab on the ribbon and then selecting the **Visual Basic** icon, which is usually the leftmost button in the ribbon section. This action will open the separate VBE window, the environment where **VBA** code is written and managed.



Once the VBE window is open, we need to insert a new module. Modules are containers for custom code that can be called from within the Excel worksheet. Navigate to the **Insert** tab within the VBE window, and select **Module** from the dropdown menu.



A blank code editor window for the new module will appear. This is where the specialized Function code for averaging by color will be placed.

Step 3: Implementing the VBA Code for Color-Based Averaging

The following code defines a custom function named **AvgCellsByColor**. This function requires two arguments: the range of data cells to be analyzed (`CellRange`) and a single cell containing the reference background color (`CellColor`). The script works by first determining the specific numerical color index of the reference cell and then looping through every cell in the data range, checking if its interior color index matches the reference. If a match is found, the value is added to a running sum, and a counter is incremented. Finally, the function returns the average (sum divided by count).

Carefully paste the following code exactly as written into the module code editor:

Function AvgCellsByColor(CellRange As Range, CellColor As Range)

Dim CellColorValue As Integer
Dim RunningAvg As Long
Dim RunningSum As Long
Dim RunngingCount As Long

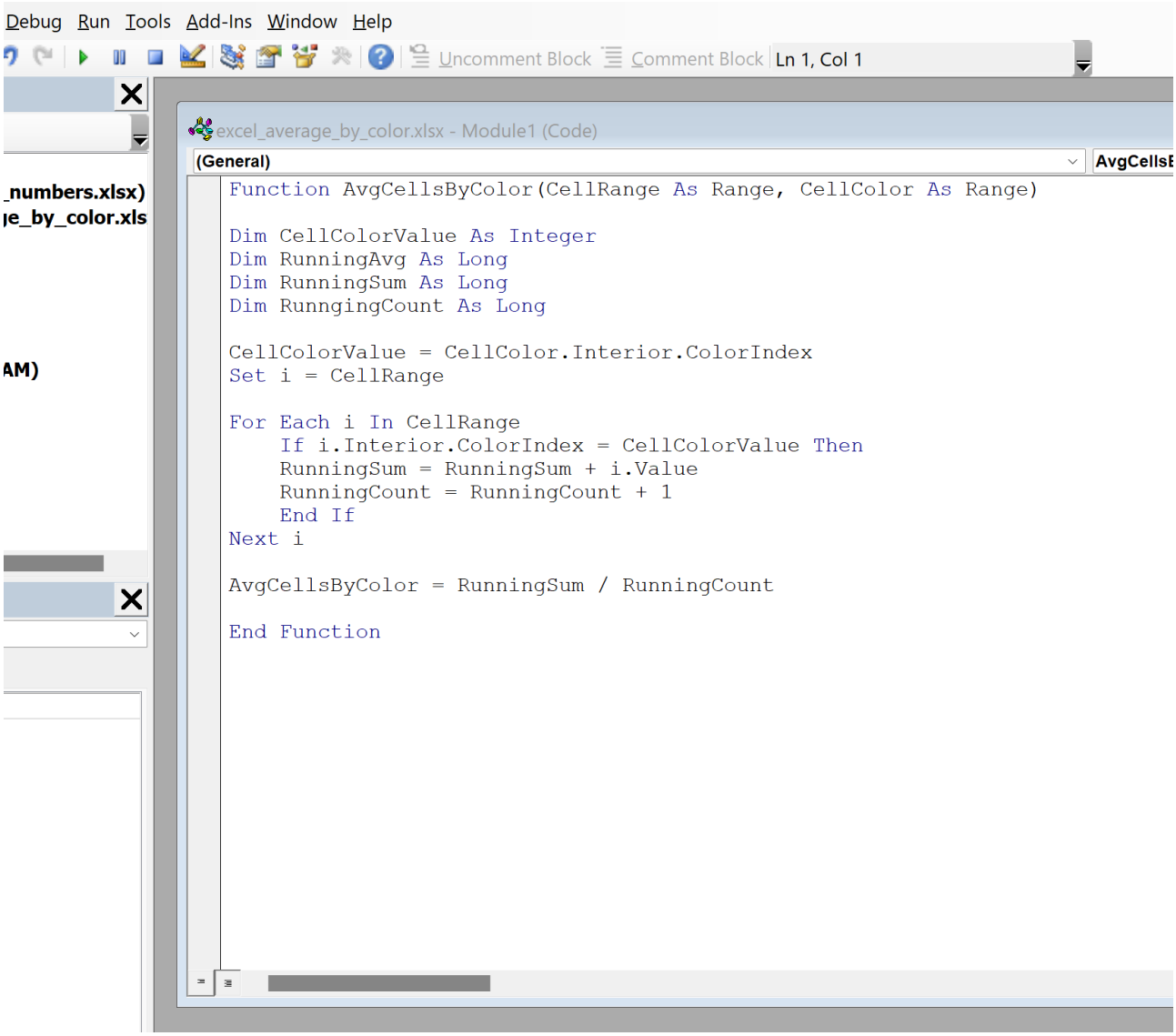
CellColorValue = CellColor.Interior.ColorIndex
Set i = CellRange

For Each i In CellRange
If i.Interior.ColorIndex = CellColorValue Then
RunningSum = RunningSum + i.Value
RunningCount = RunningCount + 1
End If
Next i

AvgCellsByColor = RunningSum / RunningCount

End Function

The successful implementation of the code within the VBE is illustrated in the screenshot below. Note that syntax highlighting is automatically applied by the editor to improve readability.



Once the code has been pasted, there is no need to save the module separately, as **VBA** code is saved automatically when you save the Excel workbook. Simply close the **Visual Basic Editor** window to return to your main worksheet.

Step 4: Applying the Custom Function in Your Worksheet

The final step is to leverage the newly created Function, `AvgCellsByColor`, directly within your Excel worksheet, just as you would use any native function like `SUM` or `AVERAGE`.

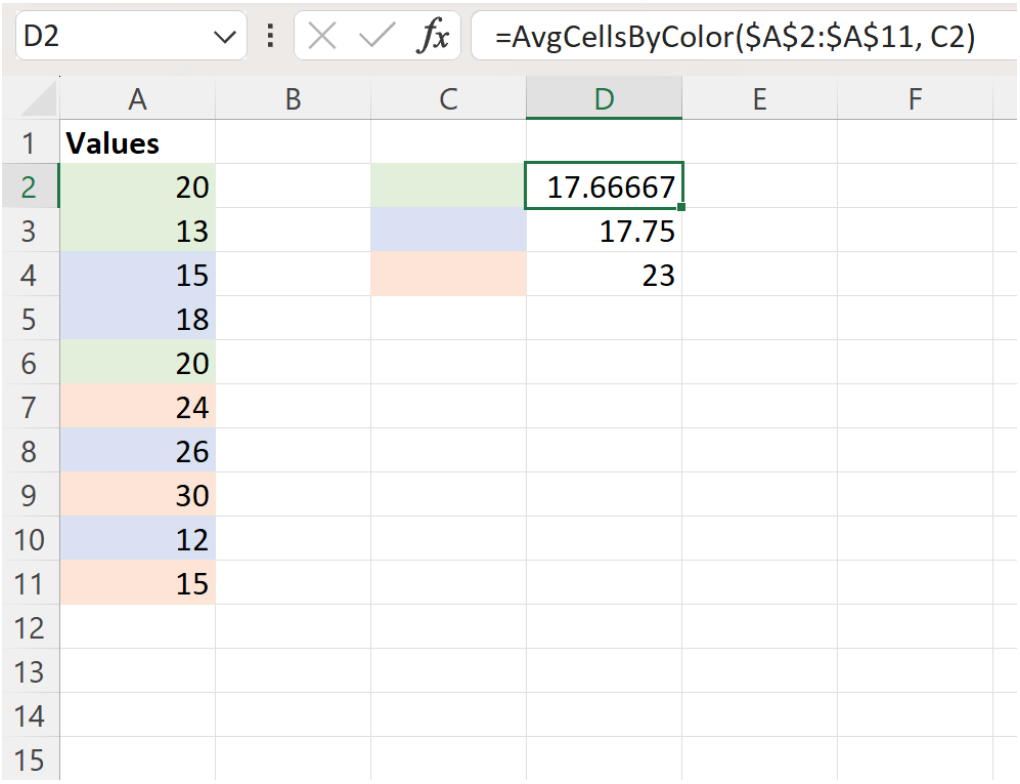
First, dedicate a separate column (e.g., Column C) to hold the reference colors. Fill cells **C2:C4** with the specific background colors for which you intend to calculate the average. These cells serve as the color criteria for the formula.

Next, in the adjacent column (e.g., cell **D2**), enter the custom formula. The formula structure

requires the fixed range of values (the data set in column A) and the relative cell containing the color criterion (the reference cell in column C). Type the following formula into cell **D2**:

=AvgCellsByColor($A$2:$A$11, C2)

The absolute referencing (using the dollar signs, **$A$2:$A$11**) ensures that the data range remains fixed when the formula is copied down. The relative reference (**C2**) ensures that the color criterion updates for each row. Drag the fill handle of cell **D2** down to the remaining cells in column D. The formula will automatically calculate the average value corresponding to each unique background color referenced in column C.



Conclusion: Validating the Results and Next Steps

The results presented in Column D demonstrate the power of using a custom Function written in VBA to perform conditional calculations based on cell formatting. For example, the calculated average for cells with a light green background is determined to be **17.67**.

We can easily confirm this result by manually calculating the average of the values associated with the light green color:

Values in light green cells: 20, 13, and 20.

Manual calculation: (20 + 13 + 20) / 3 = 53 / 3 = **17.67**.

This verification confirms that the custom **VBA** Macro successfully and accurately performed the required color-based averaging calculation. This technique is indispensable when dealing with data that relies on visual formatting as a key categorization metric. Remember that since this solution uses a Macro, the workbook must be saved as an **Excel Macro-Enabled Workbook (.xlsm)** to preserve the custom code.

Additional Resources

The following tutorials explain how to perform other common operations in Excel, expanding on the concepts of conditional calculations and advanced data handling: