How to Automatically Record the Last Saved Date and Time in Excel

Authored by

stats writer

January 3, 2026

The Importance of Tracking Document Metadata in Excel

In data management and financial modeling, maintaining a reliable audit trail is paramount. Tracking the precise moment an Excel workbook was last saved provides crucial metadata for version control, collaborative accountability, and adherence to regulatory standards. While modern cloud services often manage version history, embedding the 'Last Saved Date' directly into the spreadsheet ensures that anyone viewing or printing the document has immediate visibility into its freshness and revision status.

Relying solely on external file system properties (like Windows Explorer's file date) can be cumbersome and unreliable, especially when files are moved or copied across different environments. Therefore, integrating this information directly into a designated cell within the worksheet is a common requirement for professional spreadsheets. This technique ensures that the date and time reflect the true status of the workbook content, offering immediate assurance that the displayed data is current.

However, achieving this functionality requires more than just standard spreadsheet formulas. Functions like **NOW()** or **TODAY()** are volatile; they display the current date and time whenever the spreadsheet recalculates, not the specific historical moment of the last save. To capture and permanently display the document's official save time, we must leverage the power of **Visual Basic for Applications (VBA)**, which allows us to access and retrieve the document's inherent properties.

Methods for Date Insertion: Built-in Functions vs. VBA

Many users initially attempt to use Excel's native functions for date insertion. It is important to understand the distinctions between these functions and why VBA is necessary for achieving the specific goal of displaying the historical 'Last Saved Date'.

The NOW() Function: This function inserts the current system date and time. It is highly volatile, meaning it updates every time the worksheet is opened or recalculated, making it unsuitable for tracking a static last saved moment.
The TODAY() Function: Similar to **NOW()**, this inserts only the current date and also recalculates frequently, failing to meet the requirement for a permanent timestamp linked to a file save event.
The DATEVALUE() Function: This is used to convert a date stored as text into a proper numeric date format that Excel can use for calculations. While useful for data manipulation, it does not retrieve file system metadata.

The only robust method to retrieve the true last saved time--which is stored internally within the workbook's properties--is through a custom function written in VBA. This custom function will access the **BuiltinDocumentProperties** object, specifically targeting the property that records the

time of the last successful save operation. This approach guarantees that the date displayed remains static until the file is saved again, providing the accurate audit trail required.

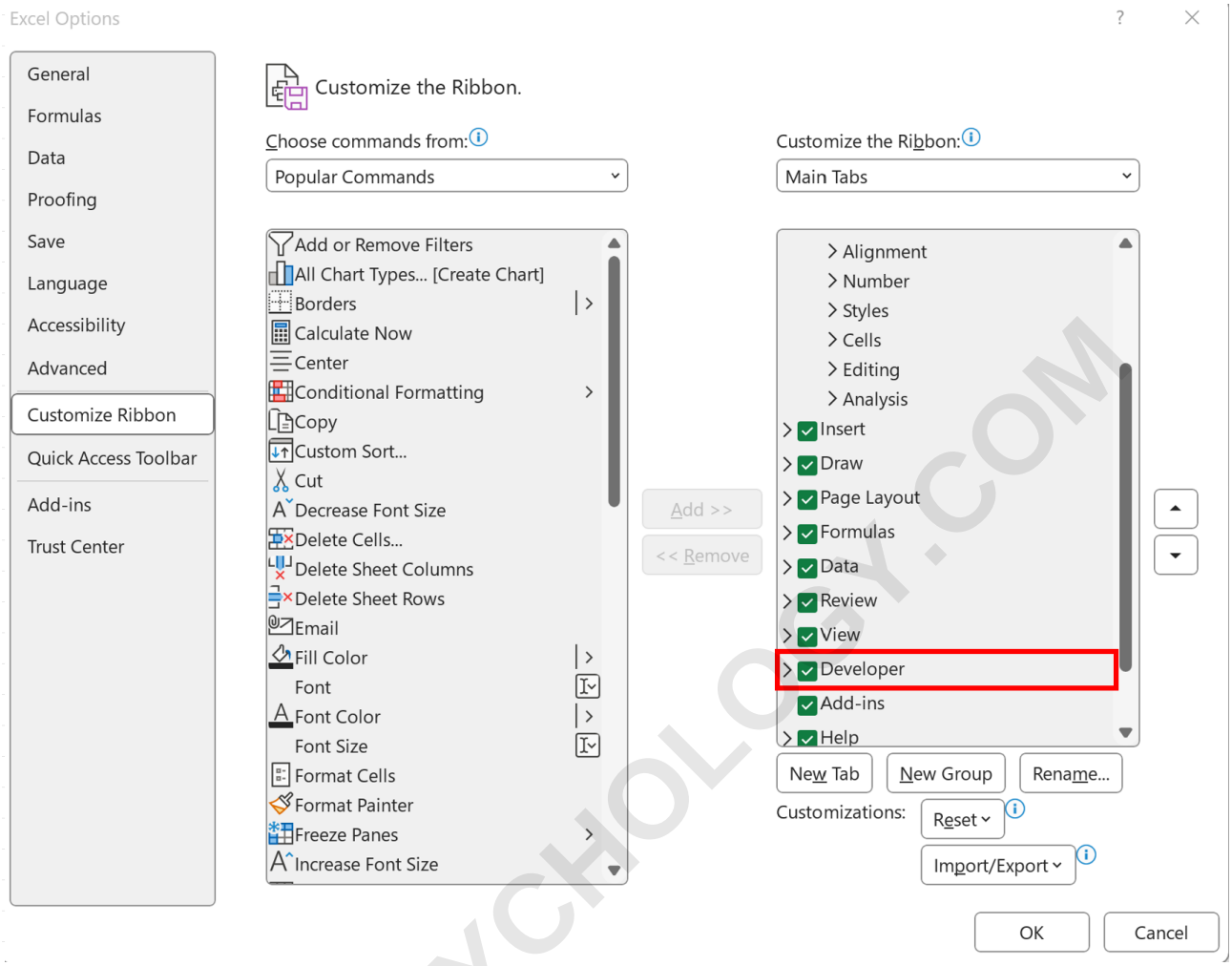
Prerequisite Setup: Enabling the Developer Tab

Before writing any custom code, the **Developer** tab must be visible in the Excel ribbon. This tab provides access to crucial tools like the VBA editor and macro controls. By default, this tab is often hidden to simplify the user interface for basic users. Enabling it is the essential first step in utilizing advanced features like Developer Tab.

The process for displaying the **Developer** tab is universal across recent versions of Microsoft Excel:

Click the **File** tab located in the upper-left corner of the Excel window.
Select **Options** from the menu to open the Excel Options dialog box.
In the left pane of the dialog box, click **Customize Ribbon**.
Under the section labeled **Main Tabs** on the right-hand side, locate and check the box next to **Developer**.
Click **OK** to close the dialog box and apply the changes.

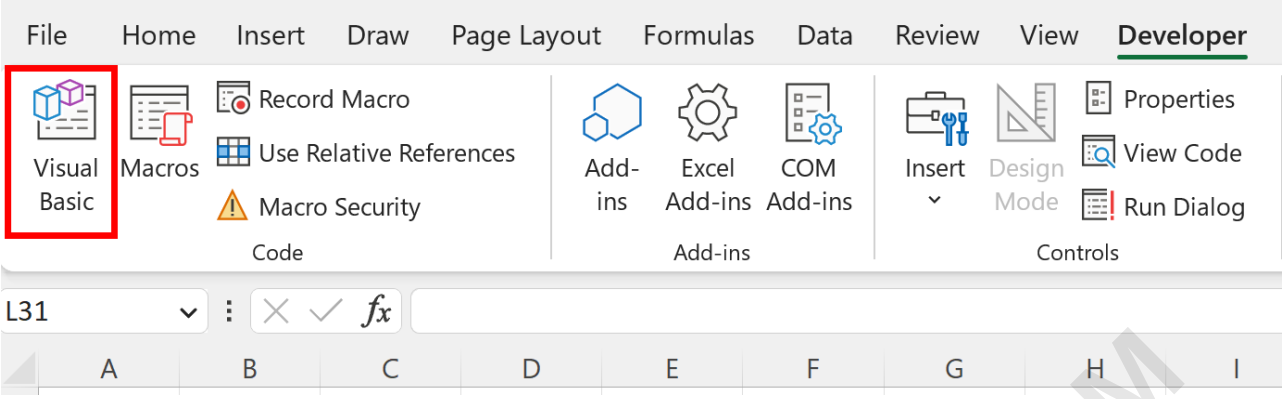
Once completed, the **Developer** tab will appear in your primary Excel ribbon, giving you access to the **Visual Basic** editor necessary for the next step.



Developing the Macro Function

The core functionality relies on creating a custom function, often referred to as a User-Defined Function (UDF) or a macro, within the VBA environment. This function will serve as a permanent formula available for use within any cell of the current workbook. The function's sole purpose is to retrieve a specific piece of information--the last save time--from the workbook's internal structure.

We begin by accessing the **Visual Basic Editor (VBE)**, the dedicated environment for writing VBA code. From the main Excel window, navigate to the newly visible **Developer** tab and click the **Visual Basic** icon.

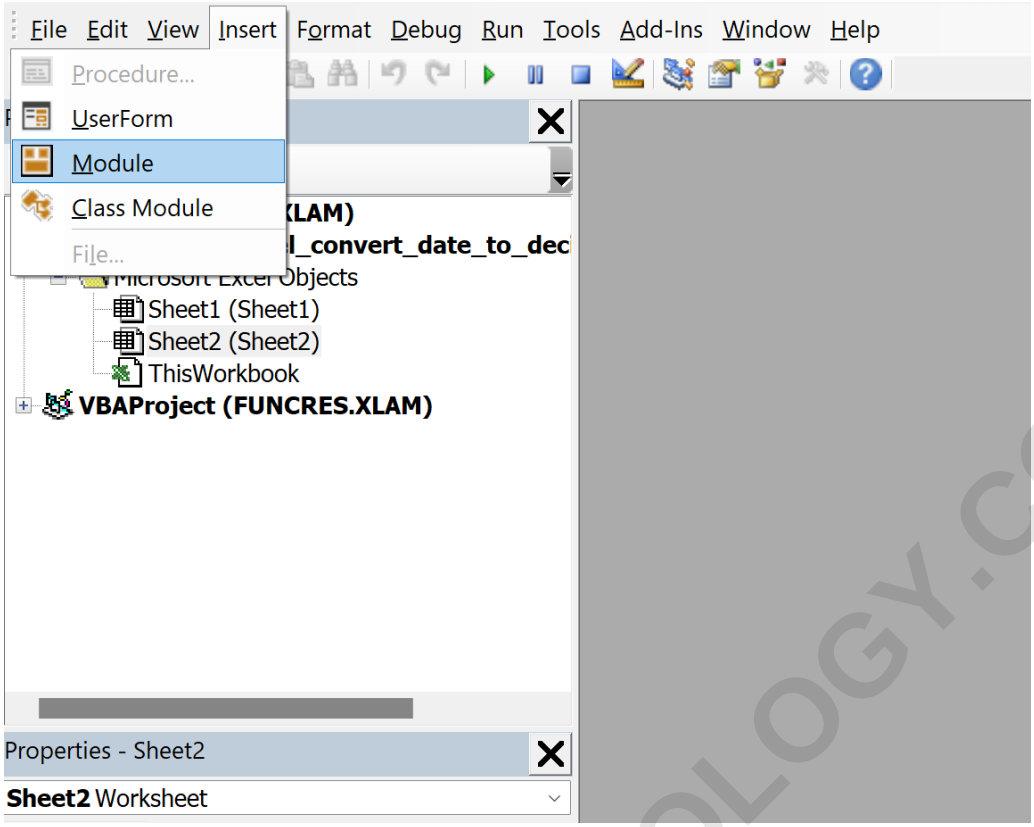


The code needed is brief but powerful, utilizing the **ActiveWorkbook.BuiltinDocumentProperties** collection to extract the specific timestamp required. This collection contains metadata properties automatically maintained by Excel, such as the creation date, author, and, critically, the 'Last Save Time'.

Implementing the Macro in the VBA Module

The custom function must reside within a standard Module, ensuring it is globally accessible to all sheets within the current workbook. Follow these steps within the Visual Basic Editor:

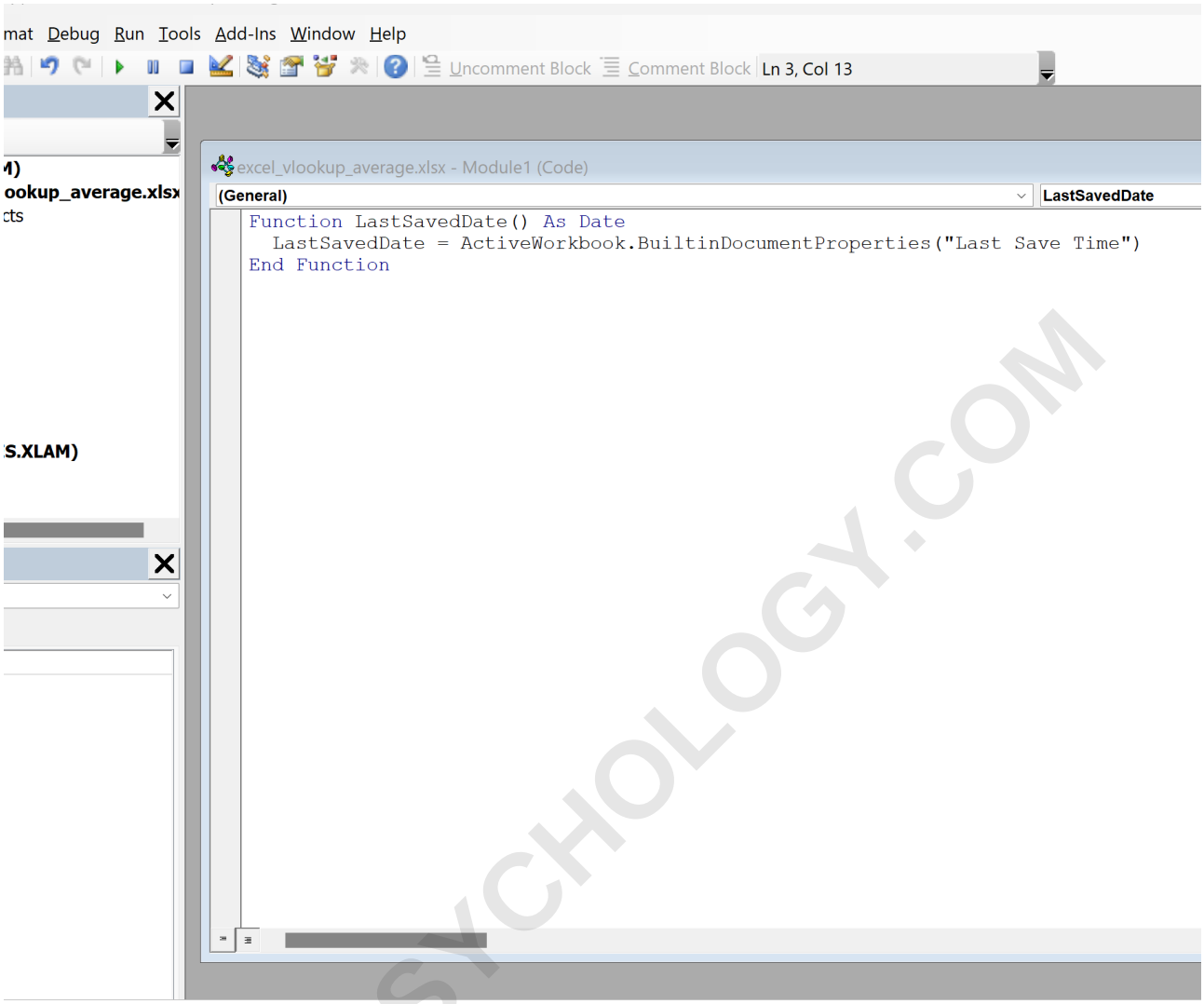
In the VBE window, click the **Insert** tab on the menu bar.
Select **Module** from the dropdown list. This opens a new, blank code editor window where we will paste the function.



Next, carefully paste the following VBA code into the blank module. This code defines the custom function named **LastSavedDate()** and sets its return value to the document's official last save time:

```
Function LastSavedDate() As Date
LastSavedDate = ActiveWorkbook.BuiltinDocumentProperties("Last Save Time")
End Function
```

The line `Function LastSavedDate() As Date` declares the function and specifies that its output should be handled as a Date/Time value. The critical instruction, `LastSavedDate = ActiveWorkbook.BuiltinDocumentProperties("Last Save Time")`, assigns the value of the internal document property to the function itself, allowing it to be returned to the worksheet. After pasting the code, ensure you save the module (File > Save) and then close the Visual Basic Editor.



Utilizing the Custom Function in the Spreadsheet

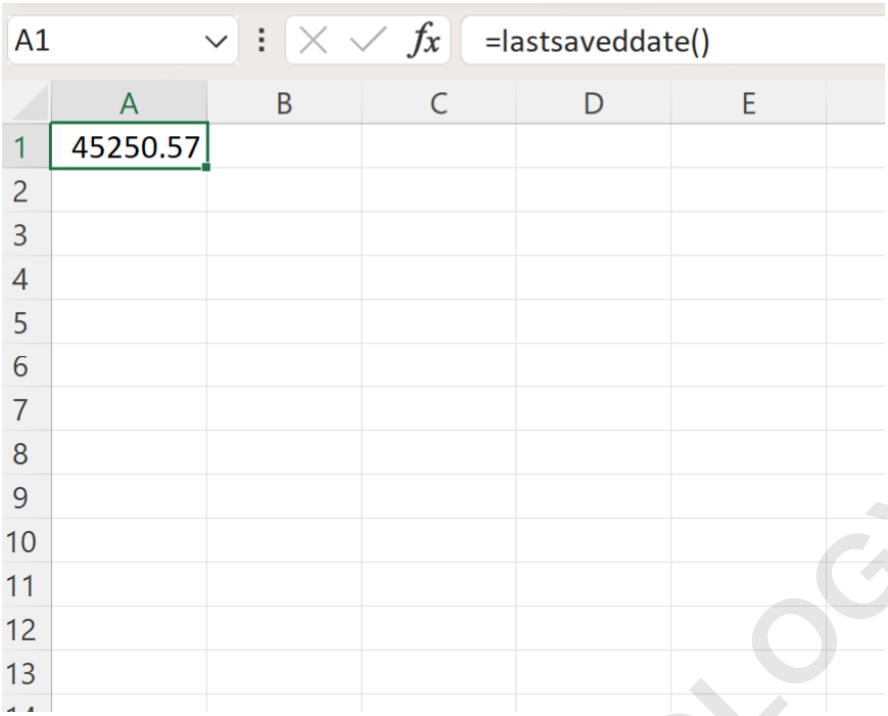
With the macro successfully implemented within the module, we can now call it directly from any cell within the workbook, just as we would use a native Excel function like **SUM()** or **AVERAGE()**. This custom function, **=LastSavedDate()**, retrieves the time stored by the macro.

To display the last saved date and time, simply navigate to the desired cell (for example, A1) and enter the following formula:

=LastSavedDate()

Upon pressing Enter, the cell will immediately display the value retrieved from the workbook's internal save time property. However, it is important to note that Excel initially stores dates and times as serial numeric values, representing the count of days and fractions of a day since January

1, 1900. Therefore, the initial result may appear as a large number, which needs proper formatting.

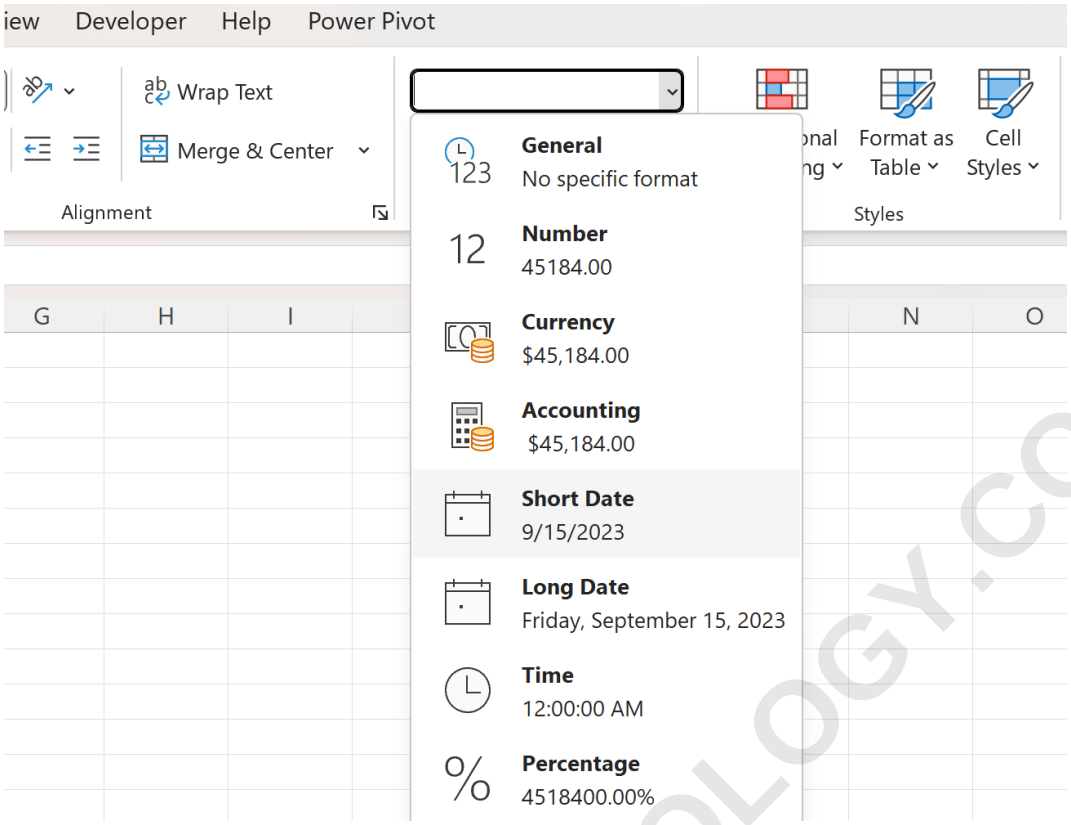


Formatting the Output: Converting Numeric Value to Date/Time

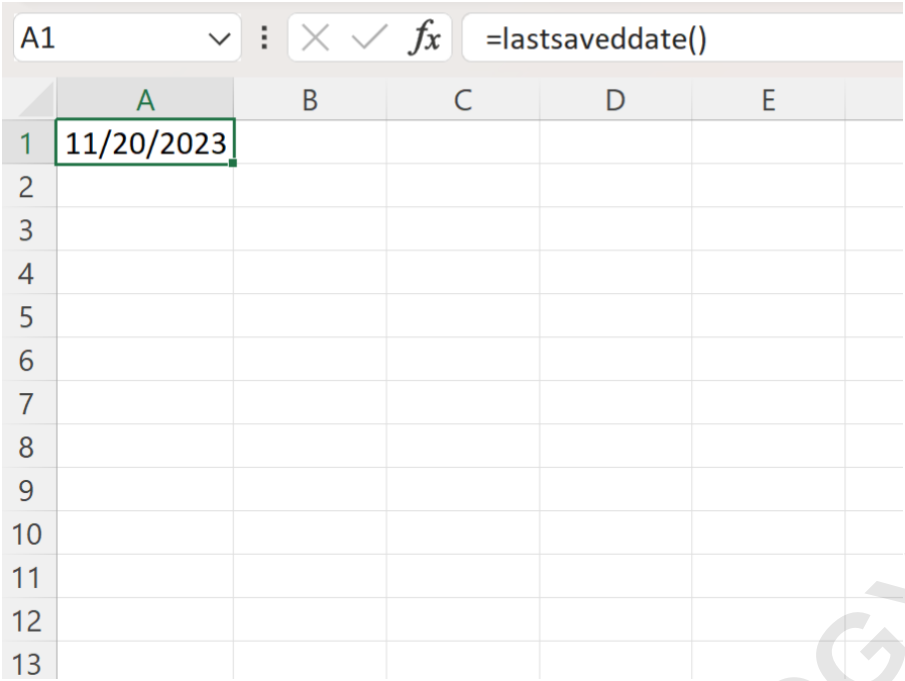
Since the custom macro returns a numeric value, the final step involves applying Excel's number formatting to convert this serial number into a readable date and time format. This process does not change the underlying value, only how it is displayed to the user.

To convert the serial number to an understandable date:

Click the cell containing the **=LastSavedDate()** formula (e.g., Cell A1).
Navigate to the **Home** tab on the Excel ribbon.
Locate the **Number** group.
Click the **Number Format** dropdown menu (usually showing "General" initially).
Select **Short Date** or **Long Date**, depending on the desired level of detail. Selecting **Short Date** is often sufficient for tracking the day the file was saved.



After applying the formatting, the numeric value will transform into a recognizable date. If the **LastSavedDate()** function returns both date and time (which it does, as the 'Last Save Time' property includes both), you may use a custom format (e.g., `mm/dd/yyyy hh:mm:ss`) to display the time component as well, offering even greater precision for auditing purposes.



For instance, if the value in cell A1 now displays **11/20/2023**, this confirms that the specific Excel workbook instance was last saved at that exact time, providing the necessary audit stamp directly within the document.

Best Practices and Considerations

When implementing macros in Excel for data tracking, several best practices should be followed to ensure security and reliability:

Saving as Macro-Enabled Workbook: Since you have implemented VBA code, the workbook must be saved as an **Excel Macro-Enabled Workbook (.xlsm)**. If saved in the standard .xlsx format, the macro will be stripped out, and the custom function will cease to work.
Security Settings: Users opening the file must have their Excel security settings configured to allow the execution of macros. If macros are disabled, the formula will return an error (e.g., #NAME?).
Automatic Update: The **LastSavedDate()** function is not volatile and will only update its displayed value when the file is explicitly saved. This non-volatility is precisely what makes it useful for tracking version history, as it maintains the historic save point until the next save event occurs.
Scope of the Function: The function created in the module is available only within the workbook where it was created. To use this functionality across multiple projects, the VBA code must be copied into a new module in each workbook, or saved to the Personal Macro Workbook (Personal.xlsb) for global availability across all Excel sessions.

By following these detailed steps, you can effectively integrate static, reliable document metadata into your spreadsheets, enhancing accountability and streamlining audit processes.