Keeping Your Code Readable

When writing R code (or any other programming language), it is important to use a clear and consistent style that is free from errors. This helps make your code easier to read and understand. In this reading, you will learn some best practices to follow when writing R code. You will also go through some tips for identifying and fixing errors in R code, also known as debugging.

Style

Using a clear and consistent coding style generally makes your code easier for others to read. There's no official coding style guide that is mandatory for all R users. But over the years, the wider community of R users has developed a coding style based on shared conventions and preferences. You can think of these conventions as the unwritten rules of R style.

There are two main reasons for using a consistent coding style:

- If you are working with collaborators or teammates, using a consistent style is important so that everyone can easily read, share, edit, and work on each other's code.
- If you are working alone, using a consistent style is important because it makes it much easier and faster to review your code later on and fix errors or make revisions.

Let's go over a few of the most widely accepted stylistic conventions for writing R code.

Naming

	Guidance	**Examples of best practice**	**Examples to avoid**

Files	File names should be meaningful and end in `.R`. Avoid using special characters in file	`# Good`
		`explore_penguins.R`
		`annual_sales.R`
			`# Bad`
			`Untitled.r`
			`stuff.r`

	names—stick with numbers, letters, dashes, and underscores.		

Object names	Variable and function names should be lowercase. Use an underscore `_` to separate words within a name. Try to create names that are clear, concise, and meaningful.
	
	Generally, variable names should be nouns.
		`# Good`
		`day_one`
			`# Bad`
			`DayOne`

	Function names should be verbs.	`# Good`
		`add ()`
			`# Bad`
			`addition ()`

Syntax

	Guidance	**Examples of best practice**	**Examples to avoid**

Spacing	Most operators (`==`, `+`, `-` , `<-`, etc.) should be surrounded by spaces.	`# Good`
		`x == y`
		`a <- 3 * 2`
			`# Bad`
			`x==y`
			`a<-3*2`

	Always put a space *after* a comma (never before).	`# Good`
		`y[, 2]`
			`# Bad`
			`y[,2]`
			`y[ ,2]`

	Do not place spaces around code in parentheses or square brackets (unless there’s a comma, in which case see above).	`# Good` `if (debug) do(x)` `species[“dolphin”, ]`	`# Bad` `if ( debug ) do(x)` `species[ “dolphin” ,]`
	Place a space before left parentheses, except in a function call.	`# Good` `sum(1:5)` `plot(x, y)`	`# Bad` `sum (1:5)` `plot (x, y)`
Curly braces	An opening curly brace should never go on its own line and should always be followed by a new line. A closing curly brace should always go on its own line (unless it’s followed by an `else` statement). Always indent the code inside curly braces.	`# Good` `x <- 7` `if (x > 0) {` ` print("x is a positive number")` `} else {` ` print ("x is either a negative number or zero")` `}`	`# Bad` `x <- 7` `if (x > 0)` `{` ` print("x is a positive number")` `}` `else {` ` print ("x is either a negative number or zero")` `}`
Indentation	When indenting your code, use two spaces. Do not use tabs or mix tabs and spaces.	`-`	`-`
Line length	Try to limit your code to 80 characters per line. This fits nicely on a printed page with a reasonably sized font.	`-`	`-`

	Note that many style guides mention to never let a line go past 80 (or 120) characters. If you're using RStudio, there's a helpful setting for this. Go to Tools -> Global Options -> Code -> Display, and select the option Show margin, and set margin column to 80 (or 120).		

Assignment	Use `<-` , not `=` , for assignment.	`# Good`
		`z <- 4`
			`# Bad`
			`Z = 4`

Organization

	Guidance	**Examples of best practice**	**Examples to avoid**

Commenting	Entire commented lines should begin with the comment symbol and a single space: `#`.	`# Good`
		`# Load data`
			`# Bad`
			`Loaddata`

Resources

- Check out this tidyverse style guide to get a more comprehensive breakdown of the most important stylistic conventions for writing R code (and working with the tidyverse).

- The styler package is an automatic styling tool that follows the tidyverse formatting rules. Check out the styler webpage to learn more about the basic features of this tool.

Debugging

Successfully debugging any R code begins with correctly diagnosing the problem. The first step in diagnosing the problem in your code is to understand what you expected to occur. Then, you can identify what actually occurred, and how it differed from your expectations.

For example, imagine you want to run the **glimpse()** function to get a summary view of the *penguins* dataset. You write the following code:

```
Glimpse(penguins)
```

When you run the function, you get the following result:

```
Error in Glimpse(penguins) : could not find function "Glimpse"
```

You were expecting a display of the dataset. Instead, you got an error message. What went wrong? In this case, the problem can be diagnosed as a stylistic error: you wrote `Glimpse` with a capital “G,” but the code is case sensitive and requires a lowercase “g.” If you run the code `glimpse(penguins)` you’ll get the result you expected.

When diagnosing the problem, it is more likely that you–and anyone else who might help debug your code–will understand the problem if you ask the following questions:

- What was your input?
- What were you expecting?
- What did you get?
- How does the result differ from your original expectations?
- Were your expectations correct in the first place?

Some bugs are difficult to discover, and finding the cause of the problem can be challenging. If you come across error messages or you need help with a bug, start by searching online for information about it. You might discover that it's actually a common error with a quick solution.

Resources

- For more information on the technical aspects of debugging R code, check out Debugging with RStudio on the RStudio Support website. RStudio Support is a great place to find answers to your questions about RStudio. This article will take you through the R debugging tools built into RStudio, and show you how to use them to help debug R code.

- To learn more about problem-solving strategies for debugging R code, check out the chapter on Debugging in Advanced R. Advanced R is a great resource if you want to explore the finer details of an R topic and take your knowledge to the next level.

Spreadsheet Errors and Fixes

When you are new to data analytics—and sometimes even when you aren't—spreadsheet struggles are real. It never feels good when you type in what you are sure is a perfect formula or function, only to get an error message. Understanding errors and how to fix them is a big part of keeping your data clean, so it's important to know how to deal with issues as they come up, and more importantly, not to get discouraged.

Remember, even the most advanced spreadsheet users come across problems from time to time. In this reading, you will learn about common errors and how to fix them.

But first, here are a few best practices and helpful tips. These strategies will help you avoid spreadsheet errors to begin with, making your life in analytics a whole lot less stressful:

1. Filter data to make your spreadsheet less complex and busy.
2. Use and freeze headers so you know what is in each column, even when scrolling.
3. When multiplying numbers, use an asterisk (*) not an X.
4. Start every formula and function with an equal sign (=).
5. Whenever you use an open parenthesis, make sure there is a closed parenthesis on the other end to match.
6. Change the font to something easy to read.
7. Set the border colors to white so that you are working in a blank sheet.
8. Create a tab with just the raw data, and a separate tab with just the data you need.

Now that you have learned some basic ways to avoid errors, you can focus on what to do when that dreaded pop-up does appear. The following table lists common spreadsheet errors and examples of each. Knowing what the errors mean takes some of the fear out of getting them.

Error	Description	Example
#DIV/0!	A formula is trying to divide a value in a cell by 0 (or an empty cell with no value)	=B2/B3, when the cell B3 contains the value 0
#ERROR!	(Google Sheets only) Something can’t be interpreted as it has been input. This is also known as a parsing error.	=COUNT(B1:D1 C1:C10) is invalid because the cell ranges aren't separated by a comma
#N/A	A formula can't find the data	The cell being referenced can't be found
#NAME?	The name of a formula or function used isn't recognized	The name of a function is misspelled
#NUM!	The spreadsheet can't perform a formula calculation because a cell has an invalid numeric value	=DATEDIF(A4, B4, "M") is unable to calculate the number of months between two dates because the date in cell A4 falls after the date in cell B4
#REF!	A formula is referencing a cell that isn't valid	A cell used in a formula was in a column that was deleted

#VALUE!	A general error indicating a problem with a formula or with referenced cells	There could be problems with spaces or text, or with referenced cells in a formula; you may have additional work to find the source of the problem.

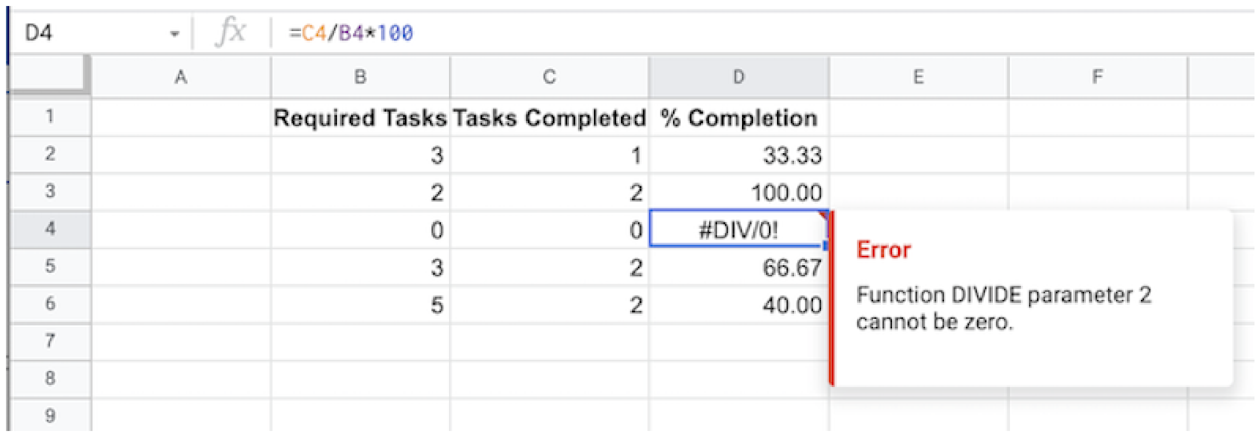
The next sections provide examples of these errors and possible solutions. There is also provide a pro tip at the end of this reading about how you can spot errors quickly in your spreadsheet by using conditional formatting.

Tip: If you are new to spreadsheets, focus on the descriptions of the errors in the previous table. Spend some time working with spreadsheets and then come back to read the examples below.

#DIV/0!

A #DIV/0! error means that your formula is trying to divide a value in a cell by 0, or by an empty cell (with no value). In math, dividing by zero doesn't make sense. Dividing by zero doesn't make sense in spreadsheets either.

Assume you are trying to calculate the percentage of required tasks that have been completed in a project. Column B has the number of required tasks and Column C has the number of completed tasks. You enter the formula **=C2/B2*100** in cell D2 to calculate the percentage of completion. You copy the formula to the rest of the cells in Column D, but there is a #DIV/0! error in cell D4. No tasks are required for that particular line item so the formula is trying to divide by 0 in cell B4.


D4 =C4/B4*100

	A	B	C	D	E	F
1		Required Tasks	Tasks Completed	% Completion		
2		3	1	33.33		
3		2	2	100.00		
4		0	0	#DIV/0!		
5		3	2	66.67		
6		5	2	40.00		
7						
8						
9						

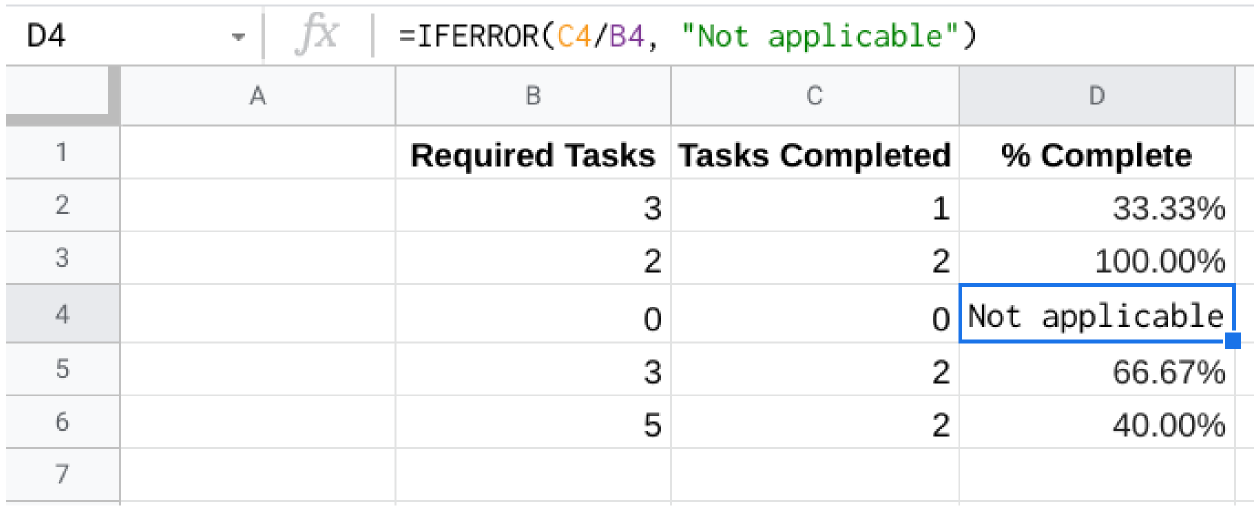
Error

Function DIVIDE parameter 2 cannot be zero.

B1: Required tasks B2: 3 B3: 2 B4: 0 B5: 3 B6: 5 C1: Tasks completed C2: 1 C3: 2 C4: 0 C5: 2 C6: 2 D1: % completion D2: 33.33 D3: 100.0 D4: #DIV/0! (Error: function DIVIDE parameter 2 cannot be zero) D5: 66.67 D6: 40.0

You could delete row 4, but if things change and tasks are required for that line item in the future, you would have to insert that row back into the spreadsheet.

A better solution is to have the spreadsheet enter "Not applicable" whenever a cell in the B column contains a 0 and causes the divide by zero error.


D4 =IFERROR(C4/B4, "Not applicable")

	A	B	C	D
1		Required Tasks	Tasks Completed	% Complete
2		3	1	33.33%
3		2	2	100.00%
4		0	0	Not applicable
5		3	2	66.67%
6		5	2	40.00%
7				

Fixing the error

Change the formula in the D column cells so the formula in cell D4 changes from =C4/B4*100 to **=IFERROR(C4/B4*100, "Not applicable")**. The results are still the same for all other cells in the D column, but the #DIV/0! error no longer appears in cell D4.

The **IFERROR** function returns the first argument (calculation) if it is not an error value, or returns the second argument when there is an error. In the example, C4/B4*100 is the first argument and "Not applicable" is the second argument.

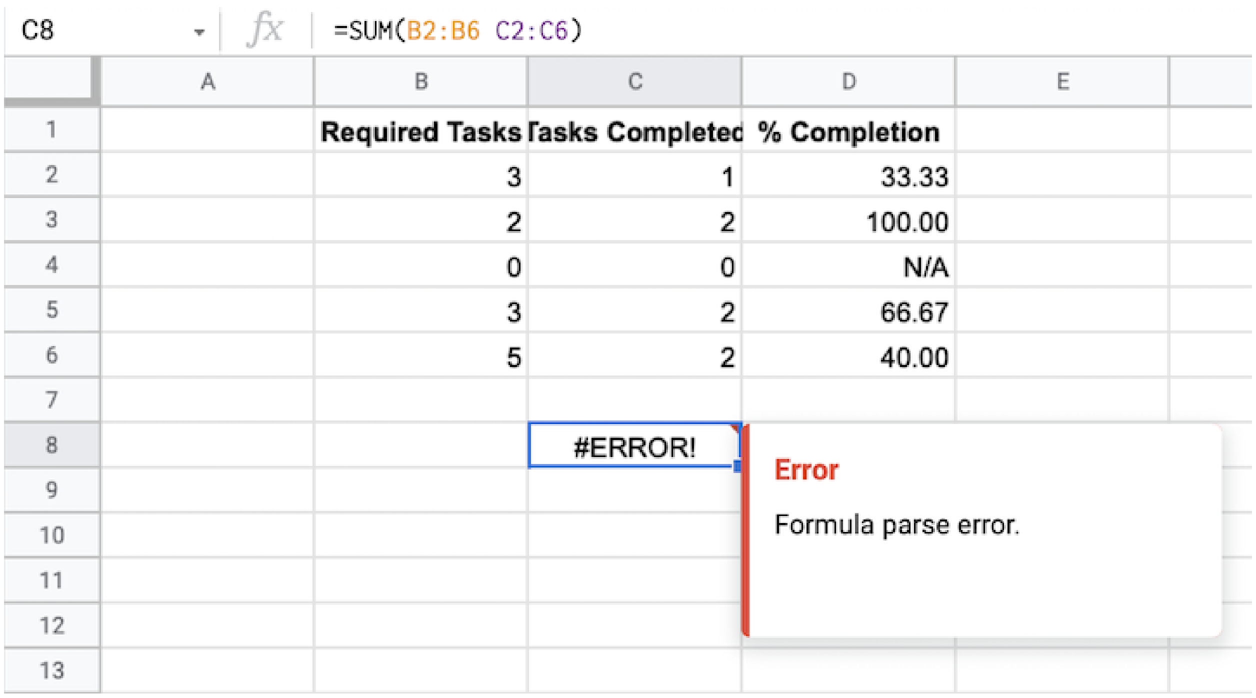
#ERROR!

A #ERROR! error can occur if you have specified two or more cell ranges without a comma as the delimiter to separate them. The spreadsheet can't figure out what the true cell ranges are.

The following ranges written without a comma cause #ERROR! to appear:

=COUNT(B1:D1 C1:C10)

=SUM(B2:B6 C2:C6)



C8 =SUM(B2:B6 C2:C6)

	A	B	C	D	E
1		Required Tasks	Tasks Completed	% Completion	
2		3	1	33.33	
3		2	2	100.00	
4		0	0	N/A	
5		3	2	66.67	
6		5	2	40.00	
7					
8			#ERROR!		
9					
10					
11					
12					
13					

Error

Formula parse error.

B1: Required tasks B2: 3 B3: 2 B4: 0 B5: 3 B6: 5 C1: Tasks completed C2: 1 C3: 2 C4: 0 C5: 2 C6: 2 D1: % completion D2: 33.33 D3: 100.0 D4: N/A D5: 66.67 D6: 40.0 C8: #ERROR! (formula parse error)

Fixing the error

You can fix these errors by replacing the space between the cell ranges with a comma.

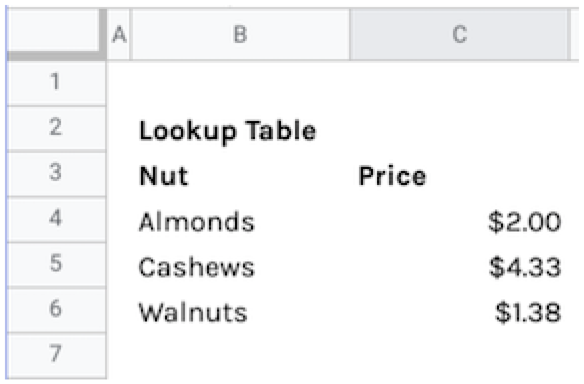
=COUNT(B1:D1,C1:C10) instructs the spreadsheet to count the number of values from cell B1 through cell D1 and from C1 through C10.

=SUM(B2:B6,C2:C6) instructs the spreadsheet to add the values in cell B2 through cell B6 and from cell C2 through cell C6.

Refer to the support pages for COUNT and SUM for additional information. These functions will be covered later in the program.

#N/A

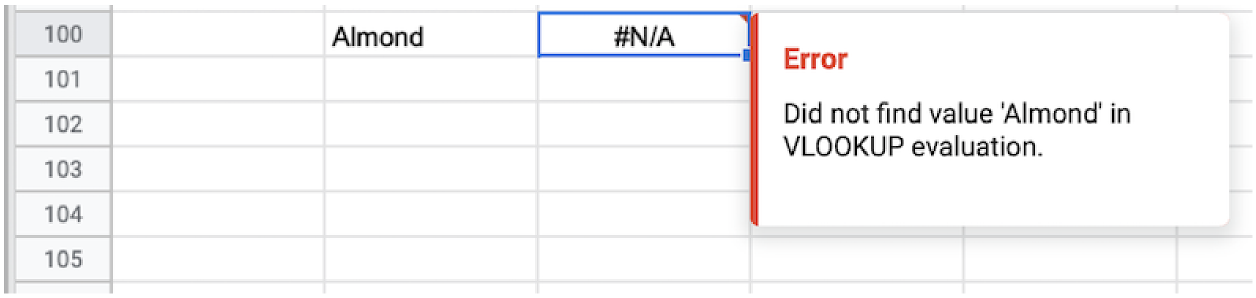
A #N/A error tells you that the data in your formula or function can't be found by the spreadsheet. Generally, this means the data doesn't exist. This error most often occurs when you are using functions like **VLOOKUP** to look up a value in a spreadsheet based on matching criteria. For example, if the following lookup table is at the top of a large spreadsheet, the spreadsheet could automatically look up and fill in the price of almonds anywhere in the spreadsheet that you have inserted a VLOOKUP function.



	A	B	C
1			
2		**Lookup Table**	
3		**Nut**	**Price**
4		Almonds	$2.00
5		Cashews	$4.33
6		Walnuts	$1.38
7			

B2: Lookup Table B3: Nut B4: Almonds B5: Cashews B6: Walnuts C3: Price C4: $2.00 C5: $4.33 C6: $1.38

Assume cell C100 in the spreadsheet contains the formula: **=VLOOKUP(B100, $B$4:$C$6, 2, 0)**. This formula should compare the text in cell B100 with Almonds, Cashews, or Walnuts in the lookup table and insert the price for the matching nut in cell C100. But there is a #N/A error in cell C100 because the price for **Almond** doesn't exist in the lookup table. The plural, **Almonds**, is in the lookup table.



100		Almond	#N/A
101			
102			
103			
104			
105			

Error

Did not find value 'Almond' in VLOOKUP evaluation.

Refer to the VLOOKUP support page for the syntax and usage of this function. VLOOKUP will be covered in more detail later in the program.

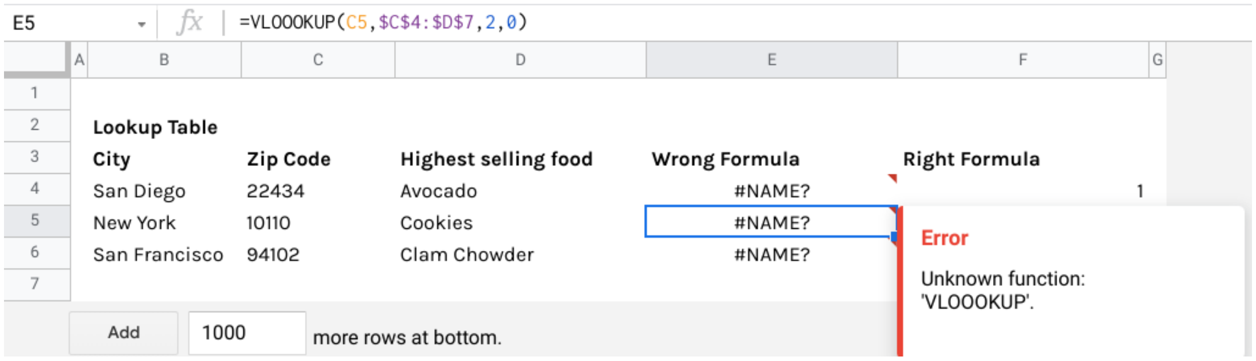
Fixing the error

To fix the error, change **Almond** in cell B100 to **Almonds**. This will enable the spreadsheet to automatically find and insert the price for almonds from the lookup table, $2.00 in cell C4, into cell C100.

#NAME?

A #NAME? error means that your spreadsheet needs help understanding your formula or function. To solve #NAME? errors, the first step is to check your spelling. Then, be sure to use the full name for any formulas or functions. Spreadsheet applications will suggest formulas and functions for you so it is a good idea to make use of this feature.

Here is an example of a #NAME? error resulting from an extra O in the VLOOKUP function. The spreadsheet is trying to use **VLOOOKUP** which doesn't exist.



```
E5   fx   =VLOOOKUP(C5,$C$4:$D$7,2,0)
```

	B	C	D	E	F
1					
2	Lookup Table				
3	City	Zip Code	Highest selling food	Wrong Formula	Right Formula
4	San Diego	22434	Avocado	#NAME?	1
5	New York	10110	Cookies	#NAME?	
6	San Francisco	94102	Clam Chowder	#NAME?	
7					

Error

Unknown function: 'VLOOOKUP'.

Add 1000 more rows at bottom.

Same spreadsheet as the previous example. Cell E5 in the Wrong Formula has been selected. The error message reads “Unknown function: 'VLOOOKUP'.” The formula reads “=VLOOOKUP(C5,$C$4:$D$7,2,0)” with an extra O in VLOOKUP.

Fixing the error

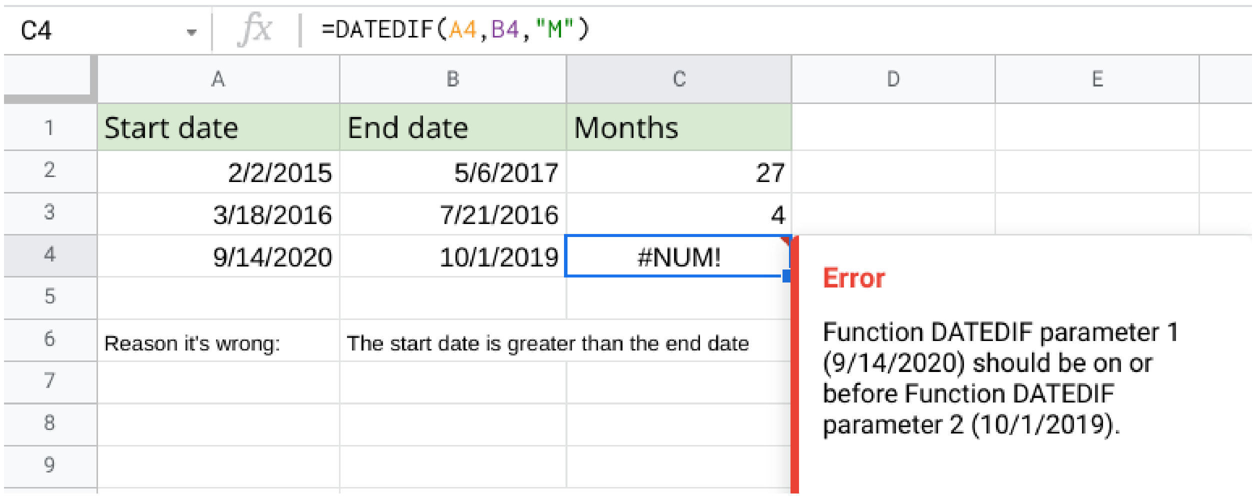
To fix the error, change **VLOOOKUP** to **VLOOKUP** in the formula in cell E5.

#NUM!

A #NUM! error means that the spreadsheet can't perform a calculation as specified. This can happen for a few reasons. The numbers might be too big or small for the spreadsheet to process, the calculation might be impossible, or there is something wrong with the variables that have been input. To fix a #NUM! error, your best bet is to just return to your formula and double-check it.

In the example below, the spreadsheet can't execute the DATEDIF formula because the Start Date is after the End Date; this needs to be corrected before the formula will work.

Refer to the DATEDIF support page for more information about the syntax and usage.



C4 =DATEDIF(A4,B4,"M")

	A	B	C	D	E
1	Start date	End date	Months		
2	2/2/2015	5/6/2017	27		
3	3/18/2016	7/21/2016	4		
4	9/14/2020	10/1/2019	#NUM!		
5					
6	Reason it's wrong:	The start date is greater than the end date			
7					
8					
9					

Error

Function DATEDIF parameter 1 (9/14/2020) should be on or before Function DATEDIF parameter 2 (10/1/2019).

Screenshot of a spreadsheet. There are three columns: Start date, End date, and Months. In the Start date column, it lists 2/2/2015, 3/18/2016, and 9/14/2020. In the End date column, it lists 5/6/2017, 7/21/2016, and 10/1/2019. In the Months column, it lists 27, 4, and a #NUM! Error. The cell with the error, C4, has been selected; The formula reads "=DATEDIF(A4,B4,"M")." The error message reads "Function DATEDIF parameter 1 (9/14/2020) should be on or before Function DATEDIF parameter 2 (10/1/2019)." Underneath the table, there is text: "Reason it's wrong: the start date is greater than the end date."

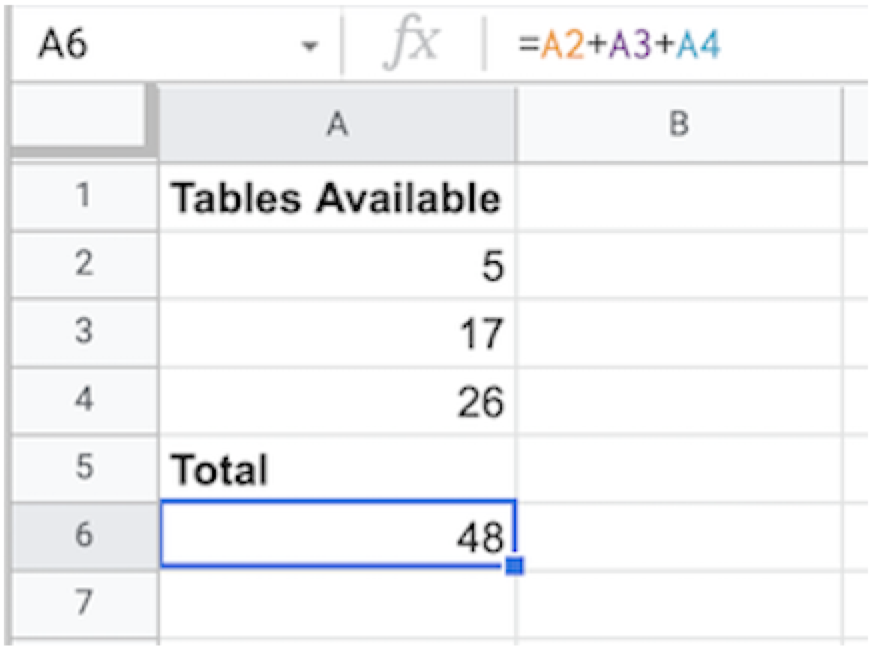
Fixing the error

Change the date in cell A4 from **9/14/2020** to **10/1/2019** and the date in cell B4 from **10/1/2019** to **9/14/2020**. The dates will be in the correct order for the formula to work, and the error will no longer appear.

#REF!

A #REF! error tells you that your formula or function is referencing a cell that is no longer valid. A cell (or range of cells) may be missing because it was deleted.

In the example below, a simple formula is adding the values in cells A2, A3, and A4.

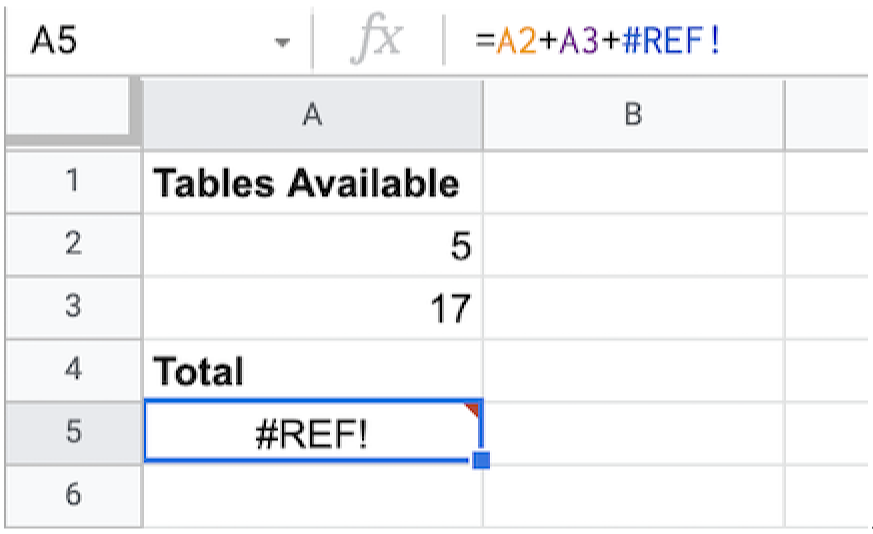


A6 =A2+A3+A4

	A	B
1	Tables Available	
2	5	
3	17	
4	26	
5	Total	
6	48	
7		

A1: Tables available A2: 5 A3: 17 A4: 26 A5: Total A6: 48

If you delete row 4 (and the value 26 in cell A4), the #REF! error appears and the spreadsheet can no longer calculate the total.



	A	B
1	Tables Available	
2	5	
3	17	
4	Total	
5	#REF!	
6		

A1: Tables available A2: 5 A3: 17 A4: Total A5: #REF!

Fixing the error

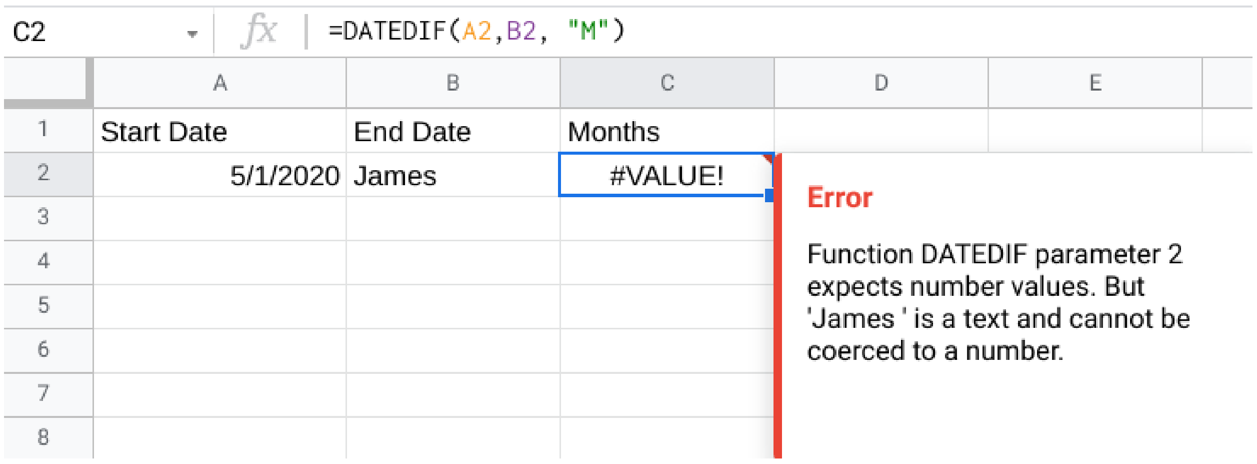
You can fix the error by updating the formula in cell A5 to add the values from cell A2 and cell A3 only. **=A2+A3**

#VALUE!

A #VALUE! error is a general error that could indicate a problem with a function or referenced cells. It might not be clear right away what the problem is, so this error could require a little more effort to fix.

If you are working with Microsoft Excel, there is an interactive page, How to correct a #VALUE! error, that can help you narrow down the cause of this error. You can select a specific function from a drop-down list to display a link to tips to fix the error when using that function.

In the example below, a text string "James" is in End Date column instead of a date. The spreadsheet can't perform the =DATEDIF(A2,B2, "M") calculation.


C2 =DATEDIF(A2,B2, "M")

	A	B	C	D	E
1	Start Date	End Date	Months		
2	5/1/2020	James	#VALUE!		
3					
4					
5					
6					
7					
8					

Error

Function DATEDIF parameter 2 expects number values. But 'James ' is a text and cannot be coerced to a number.

Screenshot of a spreadsheet. There are three columns: Start Date, End Date, and Months. Under Start Date, it contains 5/1/2020. Under End Date, it contains the name James. The formula reads “=DATEDIF(A2,B2, "M").”There is a #VALUE! Error under the Months column; the message reads “Function DATEDIF parameter 2 expects number values. But 'James ' is a text and cannot be coerced to a number.”

Fixing the error

Replace "James" in cell B2 with an end date in the right format, and the error will no longer appear.

Pro tip: spotting errors in spreadsheets with conditional formatting

Conditional formatting can be used to highlight cells a different color based on their contents. This feature can be extremely helpful when you want to locate all errors in a large spreadsheet. For example, using conditional formatting, you can highlight in yellow all cells that contain an error, and then work to fix them.

Conditional formatting in Microsoft Excel

To set up conditional formatting in Microsoft Excel to highlight all cells in a spreadsheet that contain errors, do the following:

1. Click the green triangle above row number 1 and to the left of Column A to select all cells in the spreadsheet.

2. From the main menu, click **Home**, and then click **Conditional Formatting** to select **Highlight Cell Rules > More Rules**.
3. For Select a Rule Type, choose **Use a formula to determine which cells to format**.
4. For Format values where this formula is true, enter **=ISERROR(A1)**.
5. Click the **Format** button, select the Fill tab, select yellow (or any other color), and then click **OK**.
6. Click **OK** to close the format rule window.

To remove conditional formatting, click Home and select Conditional Formatting, and then click Manage Rules. Locate the format rule in the list, click Delete Rule, and then click OK.

Conditional formatting in Google Sheets

To set up conditional formatting in Google Sheets to highlight all cells in a spreadsheet that contain errors, do the following:

1. Click the empty rectangle above row number 1 and to the left of Column A to select all cells in the spreadsheet. In the Step-by-step in spreadsheets video, this was called the Select All button.
2. From the main menu, click **Format** and select **Conditional Formatting** to open the Conditional format rules pane on the right.
3. While in the Single Color tab, under Format rules, use the drop-down to select **Custom formula is,** enter **=ISERROR(A1)**, select yellow (or any other color) for the formatting style, and then click **Done**.

To remove conditional formatting, click Format and select Conditional Formatting, and then click the Trash icon for the format rule.

Spreadsheet error resources

To learn more and read about additional examples of errors and solutions, explore these resources:

- **Microsoft Formulas and Functions:** This resource describes how to avoid broken formulas and how to correct errors in Microsoft Excel. This is a useful reference to have saved in case you run into a specific error and need to find solutions quickly while working in Excel.

- **When Your Formula Doesn’t Work: Formula Parse Errors in Google Sheets**: This resource is a guide to finding and fixing some common errors in Google Sheets. If you are working with Google Sheets, you can use this as a quick reference for solving problems you might encounter working on your own.

With some practice and investigative determination, you will become much more comfortable handling errors in spreadsheets. Each error you catch and fix will make your data clearer, cleaner, and more useful.