

How to Sum Matches with XLOOKUP in Excel

Authored by
stats writer

February 3, 2026

RECOMMENDED CITATION

stats writer (2026). *How to Sum Matches with XLOOKUP in Excel*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=129269>

Excel: Using Dynamic Arrays to Sum All Matches (Beyond XLOOKUP)

The Excel environment demands efficient tools for managing and querying large datasets. Among these tools, the XLOOKUP function has emerged as a cornerstone replacement for older lookup methods like **VLOOKUP** and **HLOOKUP**. It provides superior flexibility, allowing users to search for specific values (the **lookup value**) within a defined range (the **lookup array**) and return corresponding data from another specified range (the **return array**). While XLOOKUP is highly versatile for single returns, users often encounter a challenge when attempting to aggregate or sum values associated with **multiple matches** of a single criterion.

A common requirement in advanced data analysis is the ability to find every instance of a specific item--such as a product code, a team name, or an employee ID--and then calculate the total sum of an associated numerical column, like sales figures or points scored. If you attempt to rely solely on the standard XLOOKUP function for this task, you will find that it is fundamentally designed to stop execution upon identifying the **first match**, thereby failing to capture the necessary aggregation across all relevant rows. This limitation necessitates the use of a dynamic array formula combination to achieve the desired result: summing all corresponding values.

This comprehensive guide will detail why traditional lookup functions are unsuitable for multiple match aggregation and, more importantly, provide the robust solution utilizing the powerful combination of the SUM function and the FILTER function. By employing this technique, you can dynamically retrieve, filter, and sum values based on specific **criteria**, ensuring accurate and comprehensive calculations for your complex Excel models. Understanding this dynamic approach is critical for moving beyond basic spreadsheet operations.

Understanding the Limitation of XLOOKUP for Aggregation

While the **XLOOKUP** function is an incredibly versatile tool for most lookup needs, its core purpose is to retrieve a single corresponding item based on a specific search criterion. When searching through a column that contains duplicate entries--for example, multiple sales transactions linked to the same customer ID--**XLOOKUP** is optimized for speed and efficiency by immediately returning the value associated with the **first occurrence** it finds. This behavior is by design, making it unsuitable when the goal is to process every match and calculate an aggregated result, such as a total sum.

Many users initially attempt to coerce XLOOKUP into summing all results, perhaps by nesting it within other aggregation functions or hoping for a hidden optional argument that handles array returns for summing. Unfortunately, this nesting often fails because the output of a standard

XLOOKUP is a single scalar value, not an array of values corresponding to all matches. Furthermore, there is no built-in "sum all matches" mode within the function's standard syntax. This is why relying solely on **XLOOKUP** when you have duplicate lookup values will result in an inaccurate calculation, as you will only capture data from the very first relevant row.

Therefore, when the requirement is to look up some value in a source range and then precisely sum the corresponding numerical values in a target column for *all matches*, we must look beyond single-lookup functions. The solution must involve a function capable of creating a temporary, filtered array of data based on the defined criteria, which can then be passed into an aggregation function. This approach leverages the modern capabilities of Excel's dynamic array engine.

The Necessity of Dynamic Array Functions for Aggregation

To overcome the inherent limitations of **XLOOKUP** and other traditional lookup functions in handling multiple matches, we must utilize FILTER function, a dynamic array function introduced in recent versions of Excel. Dynamic array functions are unique because they can return multiple values to adjacent cells simultaneously, creating a spill range. In our scenario, the **FILTER** function is ideal because it allows us to define specific conditions and return an array containing only the values that meet those conditions.

Specifically, the **FILTER** function evaluates a range (the **include array**) against a logical test. When this logical test is met (i.e., the lookup value is found), **FILTER** returns the corresponding data from the specified data array. Unlike traditional array formulas that required manual entry using **Ctrl+Shift+Enter**, dynamic array functions handle the array processing automatically, resulting in simpler and cleaner formulas. This ability to isolate and return a precise subset of data is the key prerequisite for accurate aggregation.

Once the **FILTER** function successfully generates an array containing only the numerical values that correspond to all matching criteria, we need a method to calculate the total. This is where the powerful and straightforward SUM function comes into play. By nesting the **FILTER** output directly inside the SUM function, we effectively instruct Excel to sum every element within that newly generated, filtered array. This combined approach is the most efficient and contemporary method for performing multi-match lookups and summation in modern Excel versions.

Deconstructing the SUM and FILTER Solution

The solution to summing all matches involves the formula structure: `=SUM(FILTER(return_array, lookup_array = lookup_value))`. This formula combines two distinct operations into a single, cohesive calculation. First, the **FILTER** function performs the lookup and filtering operation, and then the SUM function performs the final calculation on the resulting data set. Understanding the arguments within the **FILTER** function is paramount to implementing this solution correctly.

The **FILTER** function requires at least two arguments: the `array` and the `include` argument. The `array` argument specifies the column or range containing the values you ultimately want to return and sum--in our case, the column containing the numerical data (e.g., points or sales). The `include` argument is where the lookup happens. Here, you define the logical test by comparing the entire lookup column (e.g., the Team column) against the specific **lookup value** (e.g., "Mavs"). This comparison creates an array of **TRUE** and **FALSE** values, where **TRUE** indicates a match.

The **FILTER** function then uses this TRUE/FALSE array as a mask: it returns only those values from the primary `array` argument that correspond to a **TRUE** result in the `include` logical test. This resulting output is a dynamic array containing only the numerical values associated with your search criterion. Finally, the outer **SUM** function receives this array of filtered numbers and calculates their total. This nested approach elegantly bypasses the inherent limitation of **XLOOKUP** by performing the filtering before the aggregation, guaranteeing that every match is included in the final sum.

Practical Example: Setting Up Your Dataset

To demonstrate this powerful technique, let us consider a common scenario in data analysis involving sports statistics. Suppose we maintain a dataset in Excel tracking the performance of basketball players across different teams. Our goal is to calculate the total points scored by a specific team, requiring us to look up the team name multiple times and sum all associated point totals.

Our dataset is structured with two key columns: a **Team** identifier column and a **Points** column. We also need a separate cell where we can input the specific team name we wish to query (the **lookup value**). This setup is crucial for making the formula flexible, allowing us to easily change the criterion without modifying the formula structure itself. The goal is to use the combination of **SUM** and **FILTER** to dynamically find and total the points for a single team name that appears repeatedly throughout the list.

The following illustration shows the structure of our sample data. In this example, the teams are listed in column A (A2:A11), and the corresponding points scored are listed in column C (C2:C11). The cell E2 is reserved for inputting the specific team name we want to analyze (our lookup value).

	A	B	C	D	E
1	Team	Points			
2	Mavs	20			
3	Rockets	14			
4	Nets	17			
5	Spurs	13			
6	Mavs	39			
7	Rockets	28			
8	Spurs	25			
9	Mavs	12			
10	Kings	15			
11	Nets	22			
12					
13					
14					
15					
16					

The data clearly shows that the team "Mavs" appears multiple times, each with a different points value. A simple XLOOKUP function would only return the points from the first row containing "Mavs" (20 points), completely ignoring the subsequent entries (39 and 12 points). Our robust solution must ensure that all three instances are identified and their corresponding points summed correctly.

Step-by-Step Implementation of the SUM(FILTER()) Formula

We are now ready to construct the formula that will accurately sum all matches based on the criteria provided in cell E2. We are looking up the value in E2 within the team names (A2:A11) and summing the corresponding points (C2:C11).

The syntax we use is as follows:

=SUM(FILTER(C2:C11, E2=A2:A11))

Let's break down how this formula executes the task internally:

Defining the Return Array (c2:c11): This is the first argument of the FILTER function. It tells Excel, "These are the numerical values I want to return if the condition is met."

Defining the Logical Test (E2=A2:A11): This is the `include` argument. Excel compares the lookup value in E2 ("Mavs") against every cell in the team column A2:A11. This comparison results in a dynamic array of TRUE/FALSE values (e.g., {TRUE; FALSE; TRUE; FALSE; ...}).

Filtering the Data: The `FILTER` function uses the TRUE/FALSE array to pull only the numerical values from C2:C11 that correspond to a TRUE result. In our example, it returns the array {20; 39; 12}.

Aggregating the Result: The outer `SUM` function then takes this filtered array {20; 39; 12} and calculates the total, yielding the final aggregated result.

This method ensures that every instance of the specified team name contributes accurately to the final total, fulfilling the requirement to sum all matches within the designated ranges. This dynamic array approach is generally superior to older, resource-intensive techniques involving `SUMIFS` for pure conditional summation tasks, especially when dealing with large datasets.

Analyzing the Practical Example: Summing Team Scores

Applying the formula to our basketball dataset allows us to immediately see the benefit of using the `SUM(FILTER())` combination. Suppose we would like to look up the value "Mavs" specified in cell E2 in the **Team** column (A2:A11) and sum the corresponding values in the **Points** column (C2:C11) for all instances.

We use the exact syntax derived above:

=SUM(FILTER(C2:C11, E2=A2:A11))

The following screenshot demonstrates the application of this formula in an Excel environment, where the result is displayed in a designated output cell:

	A	B	C	D	E	F
1	Team	Points		Team	Mavs	
2	Mavs	20		Sum of Points	71	
3	Rockets	14				
4	Nets	17				
5	Spurs	13				
6	Mavs	39				
7	Rockets	28				
8	Spurs	25				
9	Mavs	12				
10	Kings	15				
11	Nets	22				
12						
13						
14						

As illustrated by the output, the formula successfully returns the value **71**. This result represents the total sum of all scores found in the **Points** column whenever the corresponding row in the **Team** column matched the criterion "Mavs". This outcome confirms the efficacy of using the combined dynamic array method for multi-match aggregation, providing a fast and accurate alternative to attempting complex workarounds with **XLOOKUP** or relying on manual calculation for large data ranges.

Final Verification and Alternative Approaches

It is always good practice to verify the results when employing new or complex formulas. We can manually check the data to ensure that 71 is indeed the correct sum for all "Mavs" entries. By filtering the dataset visually or performing a manual calculation based on the rows where the team is "Mavs," we confirm the following:

Row 2: Mavs, 20 Points

Row 5: Mavs, 39 Points

Row 10: Mavs, 12 Points

Sum of Mavs Points: $20 + 39 + 12 = 71$. The result precisely matches the output generated by the `=SUM(FILTER(C2:C11, E2=A2:A11))` formula, validating our approach.

While the **SUM(FILTER())** combination is the most modern and efficient method for this specific

task, there are alternative approaches worth noting, particularly if you are using an older version of Excel that does not support dynamic arrays. One robust alternative is the **SUMIFS** function. The **SUMIFS** function is specifically designed to sum values in a range based on one or more criteria and does not suffer from the same first-match limitation as XLOOKUP. The syntax for **SUMIFS** would be `=SUMIFS(sum_range, criteria_range1, criteria1)`, which achieves an identical aggregation result in this single-criterion scenario, often providing backward compatibility.

Another powerful tool frequently utilized for summarizing and aggregating data, especially when multiple criteria or dimensional views are required, is the **Pivot Table**. Pivot tables provide a highly flexible graphical interface for summarizing data, eliminating the need to write complex formulas entirely. However, for a quick, one-off, or embedded calculation within a larger spreadsheet model, the **SUM(FILTER())** formula offers the best blend of clarity, modernity, and performance.

The following tutorials explain how to perform other common operations in Excel: