# Google Sheets: Filter a Column by Multiple Values

Authored by

**Mohammed looti**

October 27, 2025

## Master the Art of Multi-Value Filtering in Google Sheets

Effective data management and analysis in **Google Sheets** often hinges on your ability to swiftly isolate specific data points. While the standard filtering tools are sufficient for dealing with simple, single **criteria**, real-world data tasks frequently demand the filtering of a **column** based on several **multiple values** simultaneously. Achieving this level of segmentation is paramount for deep-diving into your **dataset**, allowing you to focus only on the relevant subsets of information and significantly streamlining your data review processes.

Fortunately, **Google Sheets** provides a sophisticated, formula-based methodology to handle this complex requirement. The key lies in combining the robust **FILTER function** with the versatile **REGEXMATCH function**. This powerful pairing enables dynamic filtering based on **regular expressions**, offering far greater flexibility and superior efficiency compared to manually constructing long chains of `OR` logical operators, especially when processing large or constantly changing data sets.

This comprehensive guide will walk you through leveraging these functions to precisely filter a **column** by **multiple values**. We will break down the essential syntax, provide practical, step-by-step examples, and even demonstrate how to invert the logic to expertly exclude specific records. By the end of this tutorial, you will be equipped to perform high-level data manipulation with unparalleled ease and precision.

## Decoding the Core Formula: FILTER and REGEXMATCH

The foundation for filtering a **column** by **multiple values** in **Google Sheets** relies entirely on orchestrating the **FILTER function** alongside the **REGEXMATCH function**. Understanding the structure of this generic formula is essential for successful implementation:

**=FILTER(A1:C11, REGEXMATCH(A1:A11, "string1|string2|string3"))**

Let's dissect the components of this powerful construction. The **FILTER function** is designed to return a filtered version of a source **range**. It accepts two primary inputs: first, the **range** of data you wish to filter (e.g., A1:C11), and second, a boolean **condition** or set of **criteria** which dictates which **rows** should be included in the final output. In our specific case, the essential **condition** is dynamically generated by the nested **REGEXMATCH function**.

The true engine of multi-value matching is the **REGEXMATCH function**. This function evaluates whether a piece of text accurately matches a specified **regular expression** pattern. It requires two arguments: the **range** containing the values to be tested (e.g., A1:A11, representing the specific **column** you intend to filter), and the **regular expression pattern** enclosed in quotation marks

(e.g., "string1|string2|string3"). The vertical pipe character (`|`) is critical here, as it functions as the logical "OR" operator within the **regular expression** syntax. This structure ensures that if any of the specified **strings** ("string1", "string2", or "string3") are present in a cell of the monitored **column**, **REGEXMATCH function** returns `TRUE`, thereby instructing the **FILTER function** to include the entire corresponding **row**.

## Practical Example: Including Specific Records

To solidify your understanding, let's work through a practical scenario involving a sample **dataset**. Imagine you are tracking statistics for basketball players, including details such as their names, the teams they play for, and their scoring records.

This is the initial structure of our sample **dataset**:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | **Team** | **Position** | **Points** | |
| 2 | Mavs | Guard | 22 | |
| 3 | Mavs | Forward | 29 | |
| 4 | Celtics | Guard | 20 | |
| 5 | Heat | Guard | 34 | |
| 6 | Celtics | Center | 36 | |
| 7 | Mavs | Forward | 19 | |
| 8 | Warriors | Forward | 15 | |
| 9 | Heat | Center | 29 | |
| 10 | Celtics | Forward | 25 | |
| 11 | Nuggets | Guard | 20 | |
| 12 | | | | |
| 13 | | | | |
| 14 | | | | |
| 15 | | | | |
| 16 | | | | |
| 17 | | | | |
| 18 | | | | |
| 19 | | | | |
| 20 | | | | |
| 21 | | | | |

Our objective is to refine this data, displaying only the **rows** where the "Team" **column** explicitly contains the team name "Heat" or "Celtics." This task perfectly illustrates the need for efficient **multiple values** filtering.

The precise formula used to execute this selective inclusion is:

**=FILTER(A1:C11, REGEXMATCH(A1:A11, "Heat|Celtics"))**

In the context of this example, A1:C11 specifies the complete data **range** that we want the formula to analyze and return. The crucial **REGEXMATCH function** is focused on A1:A11, which is the specific **column** containing the "Team" names. The **regular expression pattern** "Heat|Celtics" acts as a direct instruction: check every cell in the "Team" **column** for the presence of "Heat" OR "Celtics." If a match is found, **REGEXMATCH function** yields `TRUE`, enabling the **FILTER function** to successfully display the corresponding **row**.

The outcome of applying this formula is visually represented below:

A14 | *fx* | `=FILTER(A1:C11, REGEXMATCH(A1:A11, "Heat|Celtics"))`

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | **Team** | **Position** | **Points** | | |
| 2 | Mavs | Guard | 22 | | |
| 3 | Mavs | Forward | 29 | | |
| 4 | Celtics | Guard | 20 | | |
| 5 | Heat | Guard | 34 | | |
| 6 | Celtics | Center | 36 | | |
| 7 | Mavs | Forward | 19 | | |
| 8 | Warriors | Forward | 15 | | |
| 9 | Heat | Center | 29 | | |
| 10 | Celtics | Forward | 25 | | |
| 11 | Nuggets | Guard | 20 | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | Celtics | Guard | 20 | | |
| 15 | Heat | Guard | 34 | | |
| 16 | Celtics | Center | 36 | | |
| 17 | Heat | Center | 29 | | |
| 18 | Celtics | Forward | 25 | | |
| 19 | | | | | |
| 20 | | | | | |
| 21 | | | | | |
| 22 | | | | | |
| 23 | | | | | |

As demonstrated in the resulting table, the filtered **dataset** is perfectly narrowed down, showing only those **rows** that meet our specified **criteria** of belonging to either the "Heat" or "Celtics" teams.

## Advanced Filtering: Excluding Multiple Values with NOT

Beyond merely including records, data analysis often requires the ability to exclude or filter out **rows** containing specific **values**. **Google Sheets** facilitates this inverse filtering by seamlessly integrating the logical **NOT function** into our existing formula structure.

The **NOT function** is designed to reverse a boolean result: if an expression yields `TRUE`, **NOT** will return `FALSE`, and vice-versa. By strategically wrapping our **REGEXMATCH function** condition within **NOT**, we invert the selection logic. This technique allows us to include all **rows** for which the specified pattern is **not** found.

If our goal is to filter for **rows** where the team name is **not equal** to "Heat" or "Celtics," we implement the following formula:

**=FILTER(A1:C11, NOT(REGEXMATCH(A1:A11, "Heat|Celtics")))**

The execution flow is straightforward: **REGEXMATCH function** first checks if a **row** contains "Heat" or "Celtics." If it matches (returns `TRUE`), the **NOT function** immediately converts that `TRUE` to `FALSE`, causing the **FILTER function** to reject the **row**. Conversely, if the team name is neither "Heat" nor "Celtics," **REGEXMATCH function** returns `FALSE`, which the **NOT function** converts to `TRUE`, thereby successfully including the record in the filtered output.

The following image clearly demonstrates the result of applying this exclusion logic:

A14 =FILTER(A1:C11, NOT(REGEXMATCH(A1:A11, "Heat|Celtics")))

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | **Team** | **Position** | **Points** | | |
| 2 | Mavs | Guard | 22 | | |
| 3 | Mavs | Forward | 29 | | |
| 4 | Celtics | Guard | 20 | | |
| 5 | Heat | Guard | 34 | | |
| 6 | Celtics | Center | 36 | | |
| 7 | Mavs | Forward | 19 | | |
| 8 | Warriors | Forward | 15 | | |
| 9 | Heat | Center | 29 | | |
| 10 | Celtics | Forward | 25 | | |
| 11 | Nuggets | Guard | 20 | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | Team | Position | Points | | |
| 15 | Mavs | Guard | 22 | | |
| 16 | Mavs | Forward | 29 | | |
| 17 | Mavs | Forward | 19 | | |
| 18 | Warriors | Forward | 15 | | |
| 19 | Nuggets | Guard | 20 | | |
| 20 | | | | | |
| 21 | | | | | |
| 22 | | | | | |
| 23 | | | | | |

This filtered view confirms that the **rows** containing "Heat" and "Celtics" have been entirely excluded, showcasing the flexibility of combining these powerful **NOT function** and **FILTER function** tools.

## Maximizing Efficiency: Case Sensitivity and Dynamic Criteria

While the **FILTER function** and **REGEXMATCH function** combination is robust, understanding specific nuances can dramatically expand your filtering capabilities in **Google Sheets**. One such critical nuance is **case sensitivity**. By default, the **REGEXMATCH function** performs a **case-sensitive** search, meaning "heat" will not match "Heat." If your data entry is inconsistent and you require a **case-insensitive** match, you must modify your **regular expression pattern** by prepending the modifier `(?i)`. For instance, the pattern would become "(?i)heat|celtics", ensuring that capitalization variations are successfully matched.

Another invaluable technique involves constructing dynamic filter **values**. Hardcoding the lookup

**strings** (like "string1|string2|string3") often limits flexibility. A superior approach is to reference a dedicated cell or a **range** of cells that hold your desired **values**. You can dynamically construct the required **regular expression pattern** using the `TEXTJOIN` function or by manually concatenating the cell references with the `&"|"&` operator. This method allows end-users to update the filter **values** instantly without needing to modify the complex core formula.

It is also important to remember that while this guide focuses on filtering a single **column** based on **multiple values**, the fundamental **FILTER function** supports multiple independent **conditions** simultaneously. You can easily extend the formula to filter by team (using **REGEXMATCH function**) AND by a minimum score (using a standard comparison operator like `>`). Each added **condition** simply becomes another argument within the **FILTER function**, separated by commas, allowing for incredibly precise and multi-layered data segmentation.

## Troubleshooting and Best Practices

Implementing complex array formulas can sometimes lead to unexpected errors. Being aware of common pitfalls will significantly speed up your debugging process. A primary source of error is incorrect **range** referencing. Always verify that the initial **range** argument within your **FILTER function** encompasses the entire area of data you intend to display (all relevant columns and rows). Furthermore, ensure the **range** supplied to the **REGEXMATCH function** accurately targets the specific **column** being filtered. Any mismatch in the number of rows between these two **ranges** will typically cause a formula error, often #VALUE!.

Another frequent challenge arises from mistakes within the **regular expression pattern** itself. Simple typographical errors or misapplication of **regular expression** metacharacters can result in either unintended matches or complete formula failure. It is essential to meticulously check your **pattern strings**, confirming correct placement of the `|` (OR) operator and ensuring proper quotation marks surround the pattern. For debugging highly complex **patterns**, external online **regex** testing tools can be invaluable for debugging.

Finally, always be mindful of data types. While **REGEXMATCH function** is fundamentally a text-matching function, if the **column** you are filtering contains numerical data, you must ensure that your **regular expression pattern** is designed to match numerical **strings** accurately. Note that an empty cell will inherently fail to match any **regular expression** pattern. If your analysis requires the inclusion or exclusion of blank **rows**, you will need to add an auxiliary logical **condition** using functions like `ISBLANK` or `NOT(ISBLANK)`.

## Conclusion: Streamlining Your Data Analysis

The technique of filtering a **column** by **multiple values** is a fundamental and indispensable skill for

anyone performing serious data work in **Google Sheets**. By mastering the synergy between the **FILTER function** and the **REGEXMATCH function**, you unlock advanced data control capabilities. This approach provides a highly flexible, scalable, and dramatically more efficient alternative to conventional manual filtering or formula construction, particularly when dealing with extensive lists of **criteria** or complex selection rules.

Whether your objective is to rapidly extract specific records, conduct detailed analysis on predefined subsets of your **data**, or systematically exclude irrelevant information, these combined functions offer a robust and reliable framework. We strongly encourage practicing the construction of **regular expressions** and thoroughly understanding the arguments of each function to fully harness their combined potential. This mastery will significantly boost your productivity and the analytical depth you can achieve within **Google Sheets**.

## Additional Resources

To further expand your proficiency in **Google Sheets** and related data manipulation techniques, consider exploring the following essential documentation and tutorials. These resources will help you build upon the foundational filtering knowledge gained in this guide, enabling you to tackle more complex spreadsheet challenges.

**Google Docs Editors Help: FILTER function**
**Google Docs Editors Help: REGEXMATCH function**
**Google Docs Editors Help: NOT function**
**Wikipedia: Regular expression**
Official **Google Sheets Help Center** for comprehensive guides.