# Excel: Use Wildcard in FILTER Function

Authored by

**Mohammed looti**

October 27, 2025

## Leveraging Complex Logic for Wildcard Filtering in Excel

The capacity to filter large and complex datasets efficiently is perhaps the most fundamental skill required for data analysis in Excel. With the introduction of the dynamic array capabilities in modern Excel versions, the `FILTER` function has become an indispensable tool for extracting specific subsets of data based on defined criteria. While `FILTER` excels at simple comparisons and exact matches, many real-world scenarios demand flexibility--specifically, the ability to filter rows based on a **partial string match**, regardless of where that string appears within the cell content. This is precisely the domain where the powerful concept of wildcard characters traditionally reigns supreme.

However, a common hurdle encountered by users is that the `FILTER` function does not natively interpret traditional Excel wildcard symbols like the asterisk (`*`) or question mark (`?`) when defining its criteria for partial text searches. Attempting to use these symbols directly within the `include` argument of `FILTER` will only result in unexpected output or errors. To successfully overcome this architectural limitation and enable robust, substring-based filtering, we must pivot from traditional wildcards and employ a sophisticated yet elegant combination of three core functions: the logical test `ISNUMBER` and the text locator `SEARCH`.

This advanced technique allows us to generate the precise boolean array (an array of TRUE/FALSE values) that the `FILTER` function requires for accurate row selection. By mastering this method, you can transform your approach to dynamic data analysis, moving beyond rigid exact matching to flexible, substring-based criteria. This comprehensive guide will walk you through the exact syntax, provide clear, practical examples, and discuss essential considerations, such as managing case sensitivity and handling empty results gracefully, ensuring you can confidently implement wildcard-like filtering across all your datasets in Excel.

## Deconstructing the Formula Structure for Partial Matching

To successfully integrate wildcard functionality within the `FILTER` function, we must construct a specific formula that utilizes the output of a text function (`SEARCH` or `FIND`) and validates it using a logical function (`ISNUMBER`). The general syntax for this powerful and versatile combination is structured as follows, forming the backbone of your partial string filtering solution:

**=FILTER(array, ISNUMBER(SEARCH("some_string", lookup_range)), "None")**

This formula is meticulously designed to process every cell in the designated `lookup_range` and determine whether it contains the specified partial string, represented here by `"some_string"`. If a match is found, the corresponding row from the main `array` is returned. The `"None"` argument, serving as the optional third parameter, is critical for formula robustness; it ensures that if the

search criteria fail to find any matches across the entire `lookup_range`, the formula returns a user-friendly text message instead of a cryptic `#CALC!` error, significantly improving user experience and output stability.

Understanding the role of each argument is essential for effective deployment. The `array` argument specifies the entire range of data--potentially multiple columns and rows--that you intend to filter and ultimately return as the dynamic result set. The `lookup_range` must be a single column or equivalent range from the `array` where the actual text search will take place. The most complex, and arguably the most important, part of this formula is the `include` argument: `ISNUMBER(SEARCH("some_string", lookup_range))`. This internal component generates a boolean array, which acts as the filter mask. For example, if the `lookup_range` has 10 rows, this segment will return an array of 10 TRUE or FALSE values, telling `FILTER` exactly which rows meet the criteria and must be included in the final output.

It is the genius of this construction that allows us to bypass the `FILTER` function's limitation regarding traditional wildcard characters. Instead of trying to force a wildcard symbol into the criterion, we use the `SEARCH` function to effectively determine the presence of a substring, which is fundamentally what a wildcard search achieves. The resulting numerical position or error is then converted into the binary logic (TRUE/FALSE) necessary for the `FILTER` function to execute its dynamic row selection mechanism successfully.

## The Essential Collaboration of ISNUMBER and SEARCH

The true innovation behind enabling wildcard-like behavior within the `FILTER` function stems from the powerful and logical interplay between the `ISNUMBER function` and the `SEARCH function`. To fully appreciate this technique, one must first grasp the distinct output characteristics of each function when applied across a range of cells, and then understand how the outer function manipulates the inner function's results. This conversion process is the linchpin of the entire filtering mechanism.

The `SEARCH function` is primarily a text utility designed to locate the starting position of a specific text string (the `find_text`) within a larger text string (the `within_text`). When `SEARCH` successfully finds the partial string, it returns the numerical index (the character position) where the match begins. However, if the specified text is absent from the target cell, `SEARCH` does not return a zero or a blank; instead, it returns the standard Excel error value: `#VALUE!`. A critical feature of `SEARCH` that makes it ideal for flexible filtering is its inherent case-insensitive nature, meaning it treats uppercase and lowercase letters identically for matching purposes, providing broader results that are often preferred in general data filtering tasks.

The role of the `ISNUMBER function` is to act as the logical translator. It is a simple diagnostic

function that checks if its argument is a numerical value, returning `TRUE` if it is and `FALSE` if it is not. By nesting the `SEARCH` function inside `ISNUMBER`, we effectively convert the results of the `SEARCH` array into the required boolean array. If `SEARCH` finds the string, it returns a number (position), which `ISNUMBER` converts to `TRUE`, signaling `FILTER` to include the row. Conversely, if `SEARCH` fails, it returns the `#VALUE!` error (which is not a number), and `ISNUMBER` converts this error into `FALSE`, signaling `FILTER` to exclude the row. This seamless conversion from numerical position/error state to a binary TRUE/FALSE array is the conceptual core of this powerful filtering technique.

## Practical Application: Filtering a Basketball Dataset

To solidify the understanding of this technique, let us apply the combined formula to a tangible scenario involving a simple dataset. Imagine you are managing a spreadsheet in Excel that lists various professional basketball teams and their respective conferences. Your task is to dynamically extract all rows for teams whose names contain a specific sequence of letters, such as the substring "ets," regardless of capitalization.

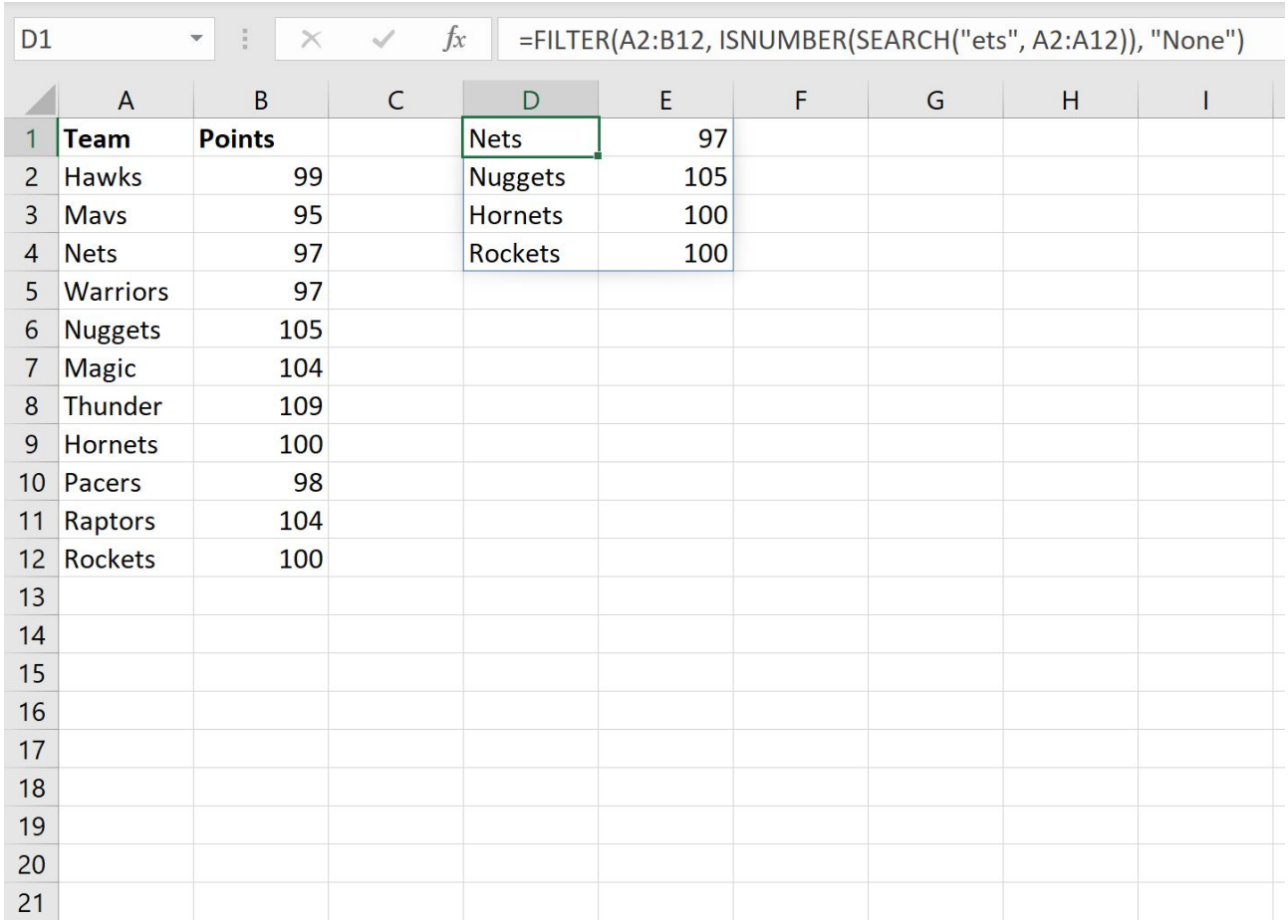
The dataset is structured across columns A and B, where column A contains the "Team" names and column B lists the "Conference" details. We define the range **A2:A12** as the specific column we wish to search for the partial string.

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | **Team** | **Points** | | | | |
| 2 | Hawks | 99 | | | | |
| 3 | Mavs | 95 | | | | |
| 4 | Nets | 97 | | | | |
| 5 | Warriors | 97 | | | | |
| 6 | Nuggets | 105 | | | | |
| 7 | Magic | 104 | | | | |
| 8 | Thunder | 109 | | | | |
| 9 | Hornets | 100 | | | | |
| 10 | Pacers | 98 | | | | |
| 11 | Raptors | 104 | | | | |
| 12 | Rockets | 100 | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |
| 16 | | | | | | |
| 17 | | | | | | |
| 18 | | | | | | |
| 19 | | | | | | |
| 20 | | | | | | |
| 21 | | | | | | |
| 22 | | | | | | |

Our objective is to use the wildcard logic to find all teams containing "ets" and return both the team name and the conference. To achieve this dynamic extraction, we enter the following formula into an empty cell (e.g., cell D2) where we want the filtered list to automatically cascade:

**=FILTER(A2:B12, ISNUMBER(SEARCH("ets", A2:A12)), "None")**

In the context of this specific example, the formula elements function as follows: `A2:B12` is the full data `array` to be returned; `"ets"` is the partial string (`find_text`) being searched for; `A2:A12` is the `lookup_range` where the search is executed; and `"None"` ensures a clean output if no teams match the criteria. Upon execution, the `SEARCH` function generates an array of numbers and `#VALUE!` errors, which `ISNUMBER` instantly converts into an array of TRUE and FALSE values, directing `FILTER` precisely which rows to display. The screenshot below illustrates the correct application and the resulting dynamic array output:




| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | **Team** | **Points** | | Nets | 97 | | | | |
| 2 | Hawks | 99 | | Nuggets | 105 | | | | |
| 3 | Mavs | 95 | | Hornets | 100 | | | | |
| 4 | Nets | 97 | | Rockets | 100 | | | | |
| 5 | Warriors | 97 | | | | | | | |
| 6 | Nuggets | 105 | | | | | | | |
| 7 | Magic | 104 | | | | | | | |
| 8 | Thunder | 109 | | | | | | | |
| 9 | Hornets | 100 | | | | | | | |
| 10 | Pacers | 98 | | | | | | | |
| 11 | Raptors | 104 | | | | | | | |
| 12 | Rockets | 100 | | | | | | | |
| 13 | | | | | | | | | |
| 14 | | | | | | | | | |
| 15 | | | | | | | | | |
| 16 | | | | | | | | | |
| 17 | | | | | | | | | |
| 18 | | | | | | | | | |
| 19 | | | | | | | | | |
| 20 | | | | | | | | | |
| 21 | | | | | | | | | |

As demonstrated by the results, the `FILTER` function successfully identified all teams containing the substring "ets" at any position within their name. The resulting list includes "Nets" (match at the beginning), "Nuggets" (match embedded), "Hornets" (match embedded), and "Rockets" (match embedded). This practical exercise confirms the formula's effectiveness in performing dynamic wildcard-like searches using the combined `ISNUMBER` and `SEARCH` method, providing a powerful, flexible alternative to traditional filtering methods in Excel.

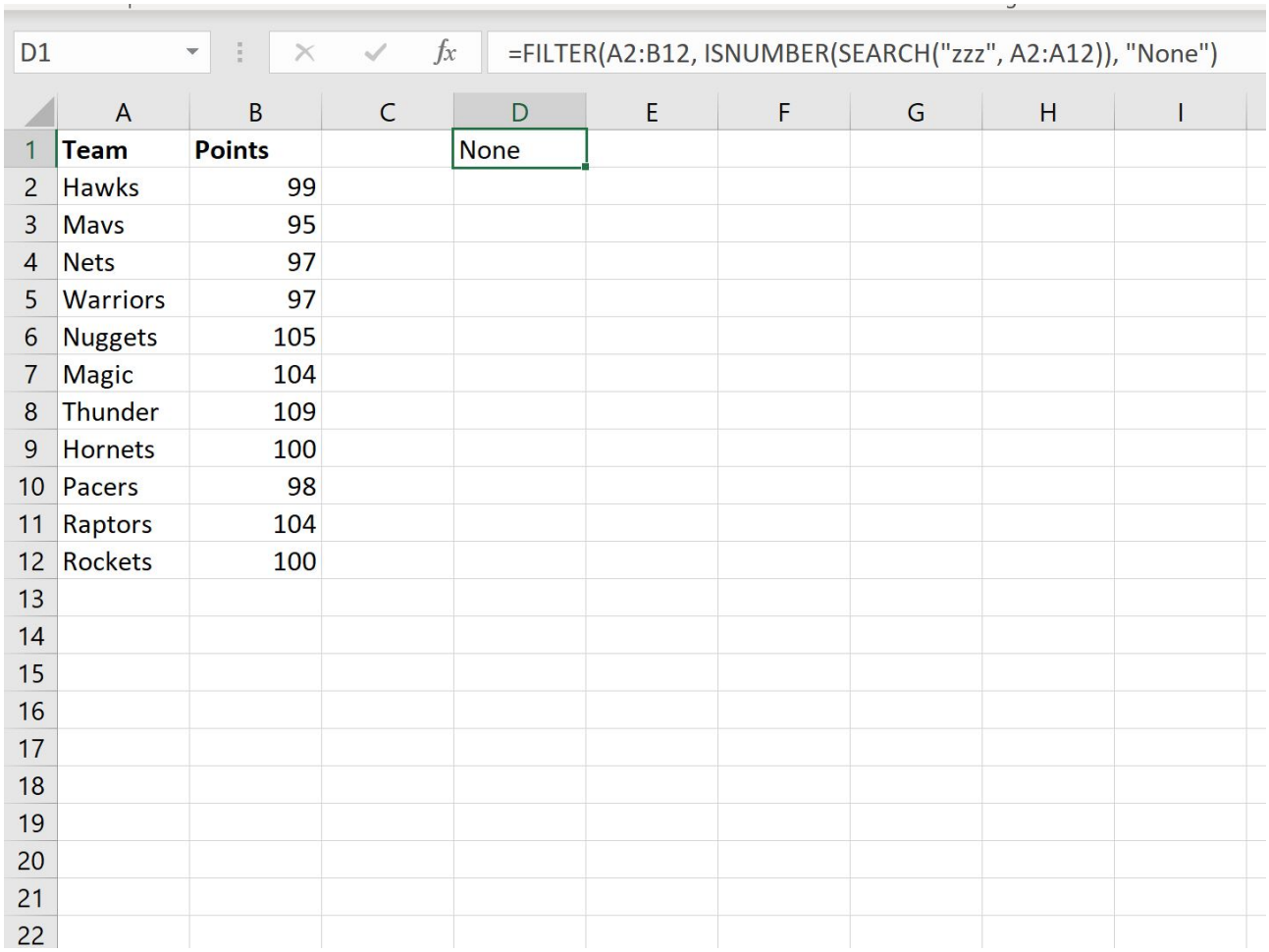
## Ensuring Robustness: Utilizing the 'if_empty' Argument

When constructing dynamic array formulas, especially those involving text matching and filtering across potentially large datasets, it is vital to account for scenarios where the criteria yield no results. The `FILTER` function addresses this need with its optional third argument, . This argument is critical for maintaining the robustness and user-friendliness of your Excel application. If the filtering criteria, derived from our complex `ISNUMBER(SEARCH(...))` logic, result in an array composed entirely of `FALSE` values, `FILTER` will automatically revert to the value specified in . Failing to specify this argument, however, causes the formula to return the default `#CALC!` error, which can be disruptive to dashboards and subsequent calculations.

In all our examples, we have employed `"None"` as the `if_empty` argument. This choice provides clear, immediate feedback to the user, indicating that the search criteria, such as the wildcard string, did not locate any corresponding data within the specified range. For instance, consider a situation where we attempt to filter the basketball dataset for a highly unusual or non-existent substring, like "zzz". The formula structure remains identical, but the outcome changes:

**=FILTER(A2:B12, ISNUMBER(SEARCH("zzz", A2:A12)), "None")**

Since the `SEARCH` function will fail to locate "zzz" in any cell from A2 to A12, the `include` argument array will consist solely of `FALSE` values. Consequently, the `FILTER` function executes its safety protocol, returning the customized text "None" specified in the third argument, as illustrated in the output below:




| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | **Team** | **Points** | | None | | | | | |
| 2 | Hawks | 99 | | | | | | | |
| 3 | Mavs | 95 | | | | | | | |
| 4 | Nets | 97 | | | | | | | |
| 5 | Warriors | 97 | | | | | | | |
| 6 | Nuggets | 105 | | | | | | | |
| 7 | Magic | 104 | | | | | | | |
| 8 | Thunder | 109 | | | | | | | |
| 9 | Hornets | 100 | | | | | | | |
| 10 | Pacers | 98 | | | | | | | |
| 11 | Raptors | 104 | | | | | | | |
| 12 | Rockets | 100 | | | | | | | |
| 13 | | | | | | | | | |
| 14 | | | | | | | | | |
| 15 | | | | | | | | | |
| 16 | | | | | | | | | |
| 17 | | | | | | | | | |
| 18 | | | | | | | | | |
| 19 | | | | | | | | | |
| 20 | | | | | | | | | |
| 21 | | | | | | | | | |
| 22 | | | | | | | | | |

The flexibility of the `if_empty` argument allows for sophisticated output customization beyond simple text. You can return a custom phrase such as `"No results found for this query"`, an empty string (`""`) if you prefer the cell to remain visually blank, or even a numerical value (e.g., `0`) if the output feeds into other numerical calculations. By diligently utilizing this argument, you ensure

that your dynamic array formulas are not only powerful in filtering but are also resilient and user-friendly, maintaining data integrity even when no matches are present.

## Controlling Precision: Case Sensitivity with FIND vs. SEARCH

A critical consideration when performing text-based filtering in Excel is the distinction between case-insensitive and case-sensitive matching. As established, the default choice for wildcard-like partial filtering--the `SEARCH function`--is fundamentally case-insensitive. This means a search for "nets" will successfully match "Nets," "NETS," or "nets," offering broad matching capability suitable for most general data queries.

However, there are specialized analytical requirements where capitalization is meaningful and exact matching is paramount. For instance, if you are filtering a list of product codes, chemical abbreviations, or proper nouns where "ID" must be distinct from "id," you require a strictly case-sensitive search. In these specific scenarios, the solution is to substitute the `SEARCH` function with the `FIND function` within the `ISNUMBER` wrapper.

The `FIND function` operates identically to `SEARCH` in terms of finding the starting position of a substring and returning a numerical index or a `#VALUE!` error if the text is not located. The crucial difference is its strict adherence to the case of the characters provided in the `find_text` argument. Using `FIND` ensures that if you search for "Nets," it will only match cells containing "Nets" with that precise capitalization and will ignore instances of "nets" or "NETS." The formula structure for a case-sensitive partial string match is therefore modified as follows:

**=FILTER(A2:B12, ISNUMBER(FIND("some_string", A2:A12)), "None")**

This inherent flexibility in choosing between `SEARCH` and `FIND` allows analysts to tailor the filtering logic to the exact requirements of the data precision needed. For general text matching, `SEARCH` provides flexibility; for mission-critical precision where case matters, `FIND` is the superior choice, ensuring your dynamic filtering results are precisely aligned with your analytical needs.

## Conclusion

The mastery of sophisticated data manipulation techniques is paramount for maximizing efficiency in Excel. While direct use of wildcard characters within the `FILTER` function is unsupported, the robust combination of the `FILTER` function with the `ISNUMBER` and `SEARCH` functions provides a dynamic and highly effective workaround for performing partial string matching. This technique is not merely a trick; it is an essential component of advanced dynamic array formulation.

By understanding how `SEARCH` or `FIND` converts text presence into a numerical index or an error,

and how `ISNUMBER` transforms this output into the critical boolean array (TRUE/FALSE) required by `FILTER`, you gain a powerful capability for flexible data extraction. This method empowers you to handle extensive lists and complex reporting needs with unparalleled speed and precision. Always remember to incorporate the `if_empty` argument to ensure your formulas are robust, professional, and provide clear feedback when no records meet the specified criteria, thereby solidifying this valuable technique in your Excel toolkit.

## Additional Resources

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