How to Easily Order Facets in ggplot2 for Clear Data Visualization

Authored by

stats writer

November 27, 2025

The ggplot2 library, a fundamental component of the R ecosystem for data visualization, offers powerful tools for segmenting plots based on categorical variables, a process known as Faceting. Faceting allows users to split a visualization into multiple panels, with each panel representing a subset of the data defined by one or more grouping variables. While the default behavior of faceting functions like `facet_wrap()` or `facet_grid()` often arranges these panels alphabetically or based on the data's inherent order, data analysts frequently require a custom display order to emphasize specific comparisons or logical sequences. Achieving this custom order is not handled by simple arguments within the faceting function itself, but rather by explicitly defining the order of the underlying factor variable levels that determine the panel structure. This technique ensures that the resulting visualization communicates the intended narrative effectively.

The ability to control the presentation order is paramount for professional data storytelling. When working with variables that have an inherent sequence (such as size categories, severity levels, or chronological groups), relying on the default alphabetical sorting can obscure the true meaning of the plot. By leveraging R's capabilities to manage factor levels, we can dictate exactly how the facet panels should be arranged, transforming a standard visualization into a highly customized and insightful graphic. This detailed guide explores the mechanisms required to override the default sorting behavior and achieve precise control over the layout of your faceted plots in **ggplot2**.

Understanding Factor Levels and Facet Ordering

In the context of the R programming language, a factor is a data structure used to store categorical data. The crucial aspect of factors is their defined set of unique values, known as "levels." Unlike simple character strings, factors maintain an internal order corresponding to these levels. When **ggplot2** receives a categorical variable for faceting, it checks if that variable is a factor. If it is, **ggplot2** respects the internal level order defined by the factor; if it is a character vector, **ggplot2** typically converts it internally and applies default sorting (usually alphabetical).

To manipulate the display order of facets, the analyst must ensure the grouping variable is treated as a factor and that the order of its levels matches the desired arrangement of the panels. This is the cornerstone of customizing facet order, whether you are using `facet_wrap()` or facet_grid(). If the data grouping variable is already a factor, you may use functions like `relevel()` or `factor()` with the `levels` argument to redefine its sequence prior to plotting. If the variable is still stored as text, the preferred method is to convert it immediately to a factor while specifying the correct order.

The flexibility provided by this approach ensures that the visualization accurately reflects any inherent or desired logical hierarchy within the data. For instance, if plotting performance across different organizational levels (e.g., Junior, Senior, Manager), forcing an alphabetical sort (Manager, Junior, Senior) would be illogical. By setting the factor levels explicitly (Junior, Senior,

Manager), the faceted plot aligns perfectly with the organizational structure, enhancing readability and interpretability for the audience.

The Core Syntax for Custom Facet Ordering

The most robust and common method for establishing a custom facet order involves applying the `factor()` function directly within the faceting layer. This method avoids permanently altering the underlying data frame, instead applying the specific order only within the context of the plot generation. This technique is highly efficient for one-off visualizations or when experimenting with different panel arrangements without modifying the original data structure.

The general syntax employs the `facet_grid()` or `facet_wrap()` functions and utilizes the `levels` argument inside the `factor()` call. By listing the categorical values in the exact sequence they should appear in the visualization, we override the default internal sorting mechanism of **ggplot2**.

You can use the following basic syntax to specify the order of facets in **ggplot2**, typically used when faceting along the horizontal axis in `facet_grid`:

```
p +
facet_grid(~factor(my_variable, levels=c('val1', 'val2', 'val3', ...)))
```

In this structure, `my_variable` is the column name being used for faceting, and the comma-separated list of strings (`'val1'`, `'val2'`, etc.) defines the precise sequence of the panels. It is critical that every unique value present in `my_variable` that you wish to display is included in the `levels` argument, and they must be spelled exactly as they appear in the data. Omission of a level will result in that category not being displayed in the facet grid.

The following example demonstrates this powerful syntax in a practical application, showing how to transform a plot with default ordering into one that follows a custom, predefined sequence.

Setting Up the Example Data Frame in R

To illustrate the process of custom facet ordering, we will work with a simple data frame containing simulated statistics for different teams. This data structure includes a categorical variable (`team`) that we will use for faceting, and two numerical variables (`points` and `assists`) that will form the basis of our scatter plot. Defining this data frame in R is the essential first step before visualization.

The data frame, `df`, tracks performance metrics across four teams (A, B, C, D), with two observations per team:

```
#create data frame
```

```
df <- data.frame(team=c('A', 'A', 'B', 'B', 'C', 'C', 'D', 'D'),
points=c(8, 14, 20, 22, 25, 29, 30, 31),
assists=c(10, 5, 5, 3, 8, 6, 9, 12))

#view data frame
df
```

team points assists
1 A 8 10
2 A 14 5
3 B 20 5
4 B 22 3
5 C 25 8
6 C 29 6
7 D 30 9
8 D 31 12

In this initial state, the `team` variable is a character vector. When **ggplot2** processes this data, it will likely sort the facets alphabetically (A, B, C, D) or based on the order of first appearance in the dataset (A, B, C, D, in this specific case, which happens to align with the alphabetical order).

Visualizing the Default Facet Order

Before implementing the custom ordering, it is instructive to observe the default behavior of the faceting function. We will use facet_grid() to create individual scatter plots for each team, mapping `assists` to the x-axis and `points` to the y-axis. This visualization serves as our baseline against which the customized result will be compared.

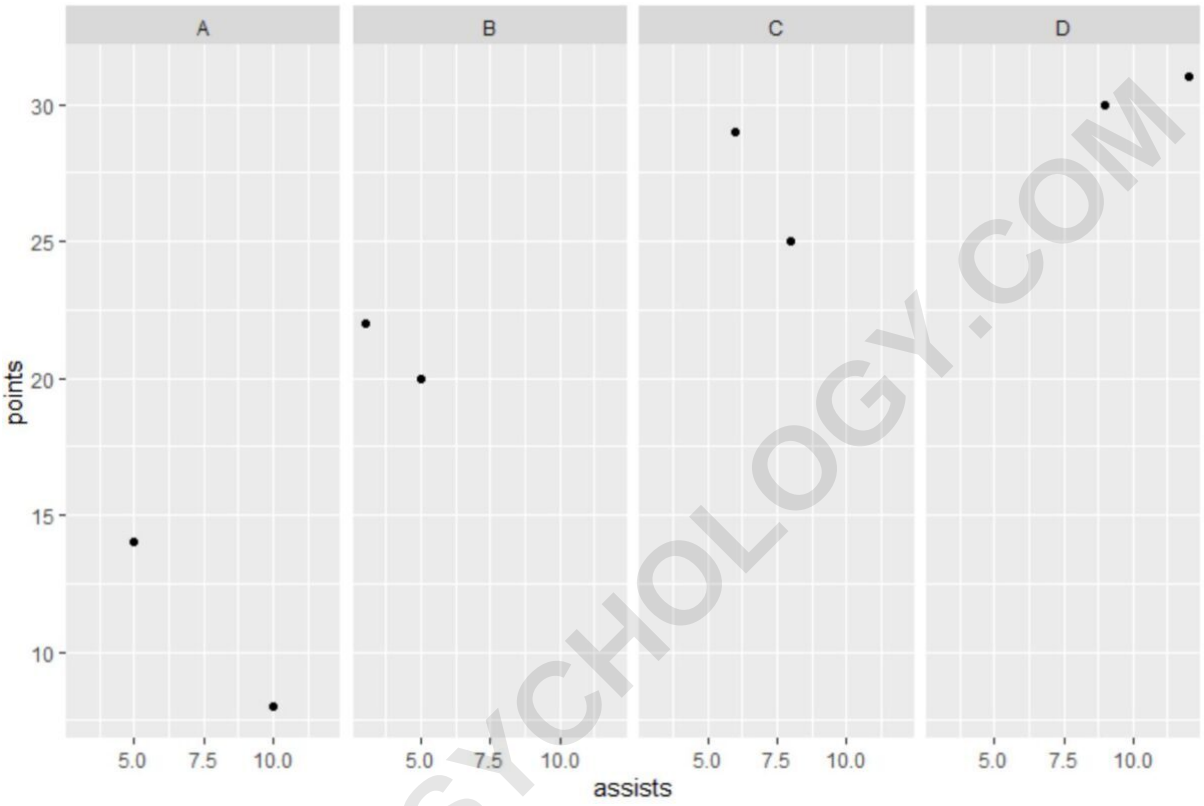
The following code utilizes the `facet_grid()` function, applying the `team` variable without any explicit factor level manipulation:

```
library(ggplot2)

#create multiple scatter plots using facet_grid
ggplot(df, aes(assists, points)) +
geom_point() +
facet_grid(.~team)
```

The syntax `facet_grid(.~team)` specifies that the panels should be arranged horizontally based on the values in the `team` column. As anticipated, the output graphic organizes the panels

according to the default sorting rules applied by **ggplot2** to the character vector, resulting in the sequence A, B, C, D from left to right. This default arrangement, derived from the variable's intrinsic character or appearance order, may not always be the most effective for data presentation.



By default, **ggplot2** places the scatter plots in order based on which values appear first in the `team` variable in the data frame, which in this case corresponds to the alphabetical sequence. While this is straightforward, it lacks the necessary control when a specific, non-alphabetical or non-chronological order is mandated by the analysis context.

Implementing Custom Order Using Factor Levels

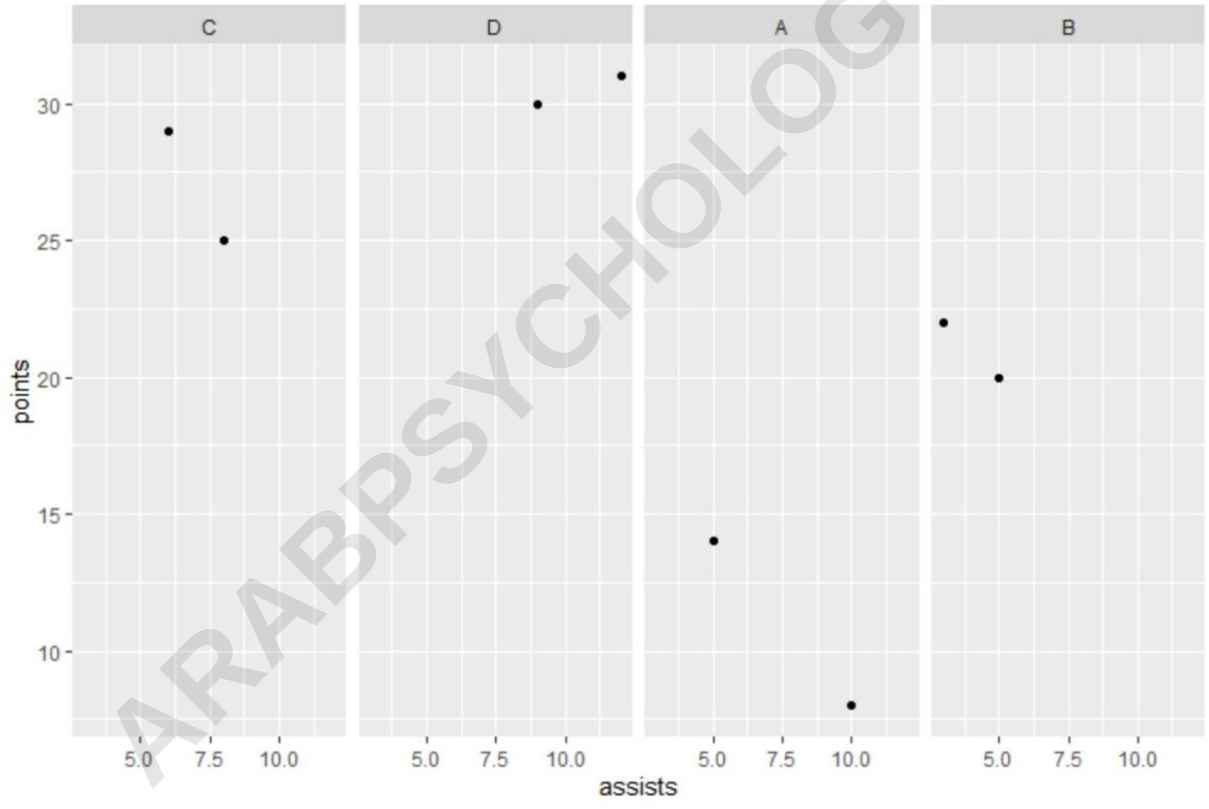
To deviate from the default sorting, we introduce the crucial step of converting the `team` variable into a factor variable directly within the `facet_grid()` call. This is achieved using the `factor()` function, where we explicitly define the `levels` argument. The sequence of values provided to `levels` dictates the precise order of the resulting facet panels.

Suppose our analytical goal is to present the data in the order C, D, A, B--perhaps prioritizing the teams with the highest observed average points, or following an external comparison standard. We simply list these teams in the desired sequence within the `levels` argument:

library(ggplot2)

```
#create multiple scatter plots using facet_grid with specific order
ggplot(df, aes(assists, points)) +
geom_point() +
facet_grid(~factor(team, levels=c('C', 'D', 'A', 'B')))
```

This single line of code, nested within the faceting function, executes the temporary conversion and ordering. The `facet_grid()` function then uses this newly ordered factor to arrange the plot panels. This methodology is highly efficient because it is contained entirely within the visualization step, minimizing the risk of modifying the source data or requiring intermediate data manipulation steps.



Upon reviewing the generated visualization, we can clearly observe that the scatter plots are now arranged precisely according to the custom order specified within the `levels` argument: C, followed by D, then A, and finally B. This confirmation demonstrates the successful application of factor level manipulation for controlling facet presentation.

Advantages and Best Practices of In-Function Ordering

The primary advantage of using the `factor(..., levels=c(...))` approach directly within the `facet_grid()` or `facet_wrap()` function is that we do not actually modify the underlying source data frame. The manipulation of levels is temporary and exists only within the scope of the plotting command. This ensures data integrity is maintained, adhering to best practices in reproducible data analysis where raw data should ideally remain untouched.

Instead of modifying the original column, we only change the levels within the faceting function's scope. This is particularly useful in workflows involving many plots or iterative analysis, where permanently changing the factor levels of a variable might interfere with subsequent calculations or summaries that rely on the default alphabetical or numeric order.

For more complex scenarios, such as ordering based on a numerical summary statistic (e.g., ordering teams by average points), an analyst would typically first calculate the summary statistic, then use helper functions from packages like `forcats` (part of the Tidyverse) to redefine the factor levels based on that calculated order, and finally pass the newly ordered factor variable to **ggplot2**. However, for manual, predefined orders, the direct `factor(..., levels)` method remains the cleanest and most explicit approach.

Extending Control to Facet_Wrap() and Axis Scaling

While our example focuses on `facet_grid()`, the exact same principle of managing factor levels applies to `facet_wrap()`. The only difference is the layout geometry: `facet_wrap()` wraps panels into a 2D layout, whereas `facet_grid()` enforces a strict grid based on two variables (rows and columns).

The original introductory text mentioned the `scales` argument within `facet_wrap()`. While `scales` does not control order, it is an important related concept when customizing facet appearance. The `scales` argument allows the user to specify whether the x and y axes should be the same across all panels (`scales = "fixed"`, the default) or vary (`scales = "free"`, `"free_x"`, or `"free_y"`). Using `scales = "free"` can dramatically improve the visual comparison when the data ranges vastly differ between facets, but it is entirely independent of the panel ordering mechanism defined by factor levels.

In conclusion, mastery over faceting in **ggplot2** requires a solid understanding of how R handles categorical data through factor levels. By taking explicit control over these levels, the data scientist ensures that visualizations are not only aesthetically pleasing but also logically structured to support compelling data narratives, moving beyond simple default sorting mechanisms.