How to Easily Create Multi-Row Legends in ggplot2

Authored by

stats writer

November 28, 2025

Creating a legend in ggplot2 with multiple rows is a common requirement when visualizing datasets with many categories. This advanced technique enhances readability by preventing the legend from overwhelming the plot area, especially when placed horizontally. The successful implementation of a multi-row legend relies primarily on the utilization of the `guides function`, specifically designed for controlling the appearance and layout of graphical elements derived from aesthetic mappings.

The process involves two critical steps: first, specifying the aesthetic mapping that drives the legend (such as color or shape); and second, invoking `guide_legend()` within `guides()` to define parameters like the number of rows (`nrow`) and the direction of item population (`byrow`). Beyond structure, visual refinement is achieved using the `theme function`, allowing precise adjustments to characteristics such as font size, background, and overall legend position, ensuring the final visualization is both informative and aesthetically pleasing.

The Necessity of Multi-Row Legends

When working with complex statistical visualizations in R, especially those generated by the powerful ggplot2 package, the default legend layout often presents challenges. If a categorical variable used for mapping aesthetics (like color or shape) contains many distinct levels--perhaps six, eight, or more--the resulting legend, by default, organizes these items into a single, lengthy column. This vertical stretch can significantly shrink the effective plot area, leading to an imbalance in the visualization composition, particularly when using standard aspect ratios.

The need for multi-row legends emerges as a solution to this spatial constraint. By collapsing a long vertical legend into a more compact, horizontal block spread across several rows, we reclaim valuable plotting real estate. This strategic reorganization ensures that the viewer can easily reference the mapping key without having their attention diverted by a disproportionately large legend box. Furthermore, in contexts where the legend is positioned at the top or bottom of the plot, controlling the number of rows is crucial for maintaining content flow and overall document readability.

A well-structured legend is integral to data storytelling. It must be clear, concise, and unobtrusive. Employing multi-row layouts is not merely a technical fix; it is a fundamental design decision that supports clarity. It provides a methodical way for content creators to manage visual density, ensuring that even data-rich plots remain accessible. To implement this structure, we turn to the specialized functions within the ggplot2 ecosystem that govern the behavior of guides.

Core Components: The guides() function and guide_legend()

To explicitly control how legends are rendered in ggplot2, one must utilize the `guides function`. The `guides()` function serves as the central hub for modifying guides (legends or axes) that are

automatically generated when aesthetic mappings are applied within the main `ggplot()` call. When dealing specifically with legend appearance, we pipe the output of `guides()` into `guide_legend()`, which accepts several arguments tailored for legend manipulation.

The essential structure for defining a multi-row legend involves specifying which aesthetic mapping is being modified and then applying the desired guide type. For instance, if the legend is driven by the `color` aesthetic, the syntax will look like `guides(color = guide_legend(...))`. Within `guide_legend()`, two arguments are paramount for achieving the desired layout: `nrow` and `byrow`. The `nrow` argument dictates the exact number of rows the legend should occupy, effectively condensing the list of items. Setting `nrow=2`, for example, forces all legend items into two rows.

Equally important is the `byrow` argument, which controls the direction in which the legend keys are populated. If `byrow` is set to `TRUE` (the typical setting for horizontal legends), the items will fill the legend row by row, moving left to right. Conversely, if `byrow` is set to `FALSE`, the items will fill column by column, which is usually the default behavior when the legend is intended to be vertical. Understanding the interaction between these two parameters is key to achieving precise control over the final presentation of the visualization key.

Here is the fundamental syntax used to create a legend with a specified number of rows:

```
ggplot(df, aes(x=x_var, y=y_var, color=group_var)) +
geom_point() +
guides(color=guide_legend(nrow=2, byrow=TRUE))
```

In this structure, the value assigned to the **`nrow`** argument explicitly determines the vertical complexity of the legend, while **`byrow = TRUE`** ensures that the items are ordered horizontally across those rows, optimizing space utilization.

Setting Up the Example Data Frame in R

To illustrate the practical application of multi-row legend creation, we will utilize a sample data frame in R. This dataset simulates performance statistics for six professional basketball teams, allowing us to map the categorical variable (team name) to the color aesthetic, thereby generating a complex legend that requires strategic arrangement. The inclusion of six unique categories necessitates a multi-row layout to maintain visual clarity when the legend is positioned horizontally.

The data frame, named `df`, contains three variables: `team` (categorical), `points` (numeric response variable), and `assists` (numeric predictor variable). We generate this structure using R's `data.frame()` function, ensuring that the variable types are correctly defined for subsequent use in ggplot2 aesthetic mappings. This setup is crucial because the number of unique entries in the

`team` column directly drives the number of items that will appear in our generated legend.

The code below demonstrates the creation and inspection of our sample dataset:

```
#create data frame
df <- data.frame(team=c('Mavs', 'Heat', 'Nets', 'Lakers', 'Suns', 'Cavs'),
points=c(24, 20, 34, 39, 28, 29),
assists=c(5, 7, 6, 9, 12, 13))

#view data frame
df

team points assists
1 Mavs 24 5
2 Heat 20 7
3 Nets 34 6
4 Lakers 39 9
5 Suns 28 12
6 Cavs 29 13
```

This dataset provides a robust foundation for our visualization exercise. By mapping the `team` variable to the color aesthetic, we can proceed to observe how ggplot2 handles the legend generation by default and then apply the necessary modifications to optimize its structure using the `guides function`.

Visualizing the Default Legend Behavior

Visualizing the Default Legend Behavior

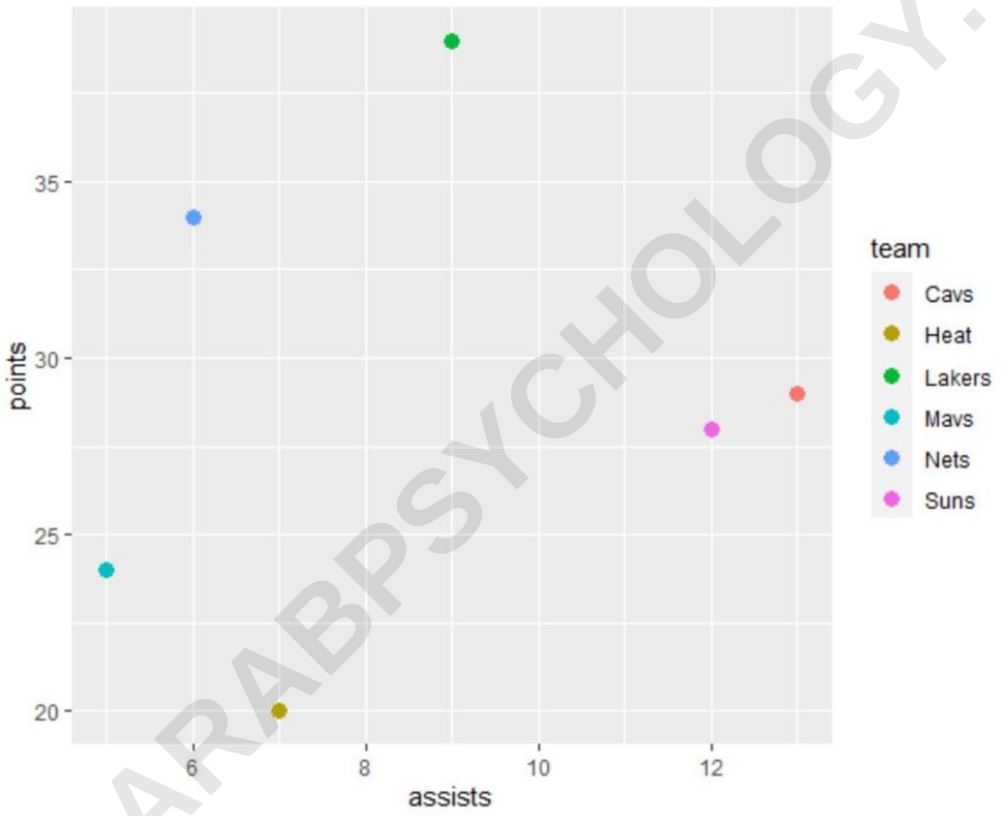
Before implementing any customizations, it is essential to understand the default behavior of the ggplot2 legend. When a scatter plot is generated using the defined dataset, mapping the `team` variable to the `color` aesthetic, ggplot2 automatically creates a legend. By default, ggplot2 attempts to maximize the vertical arrangement of legend items, resulting in a single, long column if space allows, or sometimes wrapping if the plot area is too constrained, but without explicit control over the number of rows.

In the following code block, we generate a basic scatter plot using `geom_point()`, mapping `assists` to the x-axis and `points` to the y-axis, differentiated by `team` color. Notice that we intentionally omit any `guides()` call or legend-specific `theme()` adjustments. This allows us to observe the baseline rendering where all six legend items are stacked vertically.

library(ggplot2)

#create default scatterplot
ggplot(df, aes(x=assists, y=points, color=team)) +
geom_point()(size=3)

As demonstrated in the resulting visualization below, the default layout places one label per line, creating a tall legend box positioned on the right side of the plot. While functional, this configuration is visually inefficient, consuming significant vertical space and potentially distracting from the primary data visualization. This inefficiency highlights why manual control over legend dimensions is often necessary in professional data presentation contexts.



Implementing the Multi-Row Legend (nrow and byrow explained)

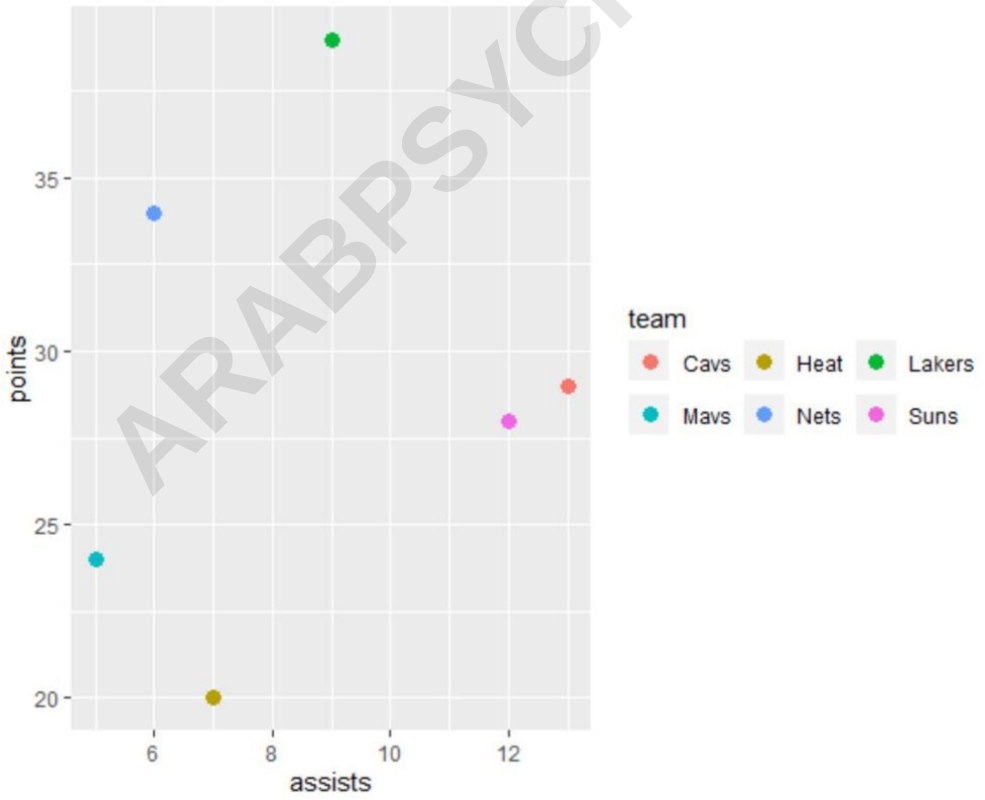
To transform the lengthy vertical legend into a more compact, horizontal multi-row format, we must introduce the `guides function` and its crucial arguments. The primary goal is to tell ggplot2 to allocate the six unique team categories across a defined number of rows, rather than relying on the default one-category-per-row standard. In our case, setting `nrow = 2` is ideal, as six items divided into two rows results in three balanced columns.

The function call `guides(color = guide_legend(nrow = 2, byrow = TRUE))` is the mechanism for this transformation. Here, we specify that the guide associated with the `color` aesthetic should be manipulated. The `nrow = 2` argument mandates that the legend must be constrained to two rows. Furthermore, `byrow = TRUE` ensures that the items fill the rows horizontally first, which is critical for creating a clean, condensed rectangular block. If `byrow` were set to `FALSE`, the items would be filled vertically down the columns first (i.e., item 1 and 2 in column 1, items 3 and 4 in column 2, etc.), which is generally less intuitive for horizontal legend placement.

By integrating this specific call into the scatter plot creation, we achieve immediate and precise control over the legend structure. This technique is highly scalable; for a dataset with 10 categories, setting `nrow = 3` would distribute the items across three rows, resulting in four columns in the first two rows and two items in the last row, maintaining proportional density.

library(ggplot2)

#create scatterplot with two rows in legend
ggplot(df, aes(x=assists, y=points, color=team)) +
geom_point()(size=3) +
guides(color=guide_legend(nrow=2, byrow=TRUE))



Observation of the resulting plot clearly shows the effectiveness of the `nrow` argument. The legend, which previously occupied a long, narrow vertical space, is now successfully condensed into a two-row format, significantly improving the visual balance of the overall graphic.

Advanced Customization: Adjusting Legend Position using **theme()**

While restructuring the legend using `guides()` is crucial for controlling internal layout, often the position of the legend relative to the plot area must also be adjusted for optimal presentation. Placing a condensed legend at the bottom of the plot is a conventional and highly effective strategy, particularly when the legend uses a multi-row horizontal format, as this arrangement maximizes the data visualization area itself.

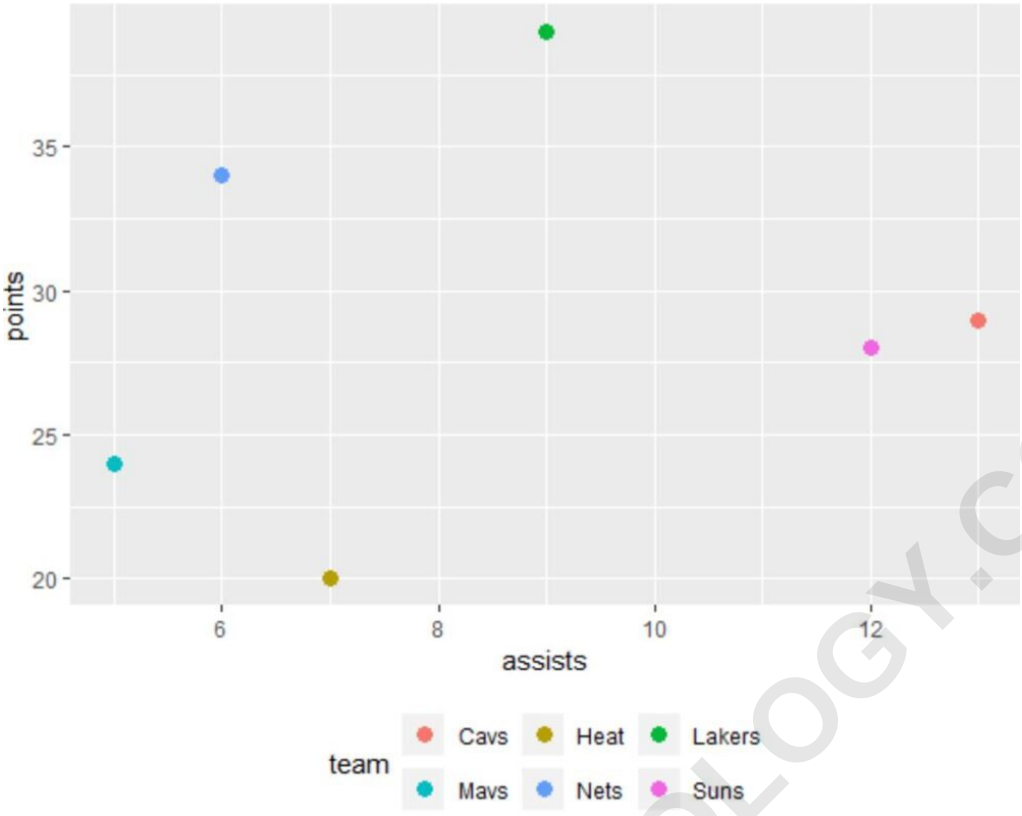
Legend positioning is managed through the `theme function`, a core component of ggplot2 responsible for non-data plot elements. Specifically, the `legend.position` argument within `theme()` allows the user to specify where the legend should be drawn. Common acceptable values include `'right'` (the default), `'left'`, `'top'`, and `'bottom'`. For our multi-row legend, moving it to the bottom is the logical step for a professional publication aesthetic.

We integrate the `theme(legend.position='bottom')` command into our existing plot syntax, ensuring that the `guides()` call remains intact to maintain the two-row structure. Note that `theme()` and `guides()` operate independently but contribute synergistically to the final output. The order of these functions in the plot chain does not generally matter for their effect, but placing `theme()` before `guides()` is conventional.

library(ggplot2)

#create scatterplot with two rows in legend and bottom position
ggplot(df, aes(x=assists, y=points, color=team)) +
geom_point()(size=3) +
theme(legend.position='bottom') +
guides(color=guide_legend(nrow=2, byrow=TRUE))

By relocating the condensed legend to the bottom, the entire width of the plot can now be dedicated to the data points, resulting in a cleaner and more focused visualization, as shown in the output below.



Further Legend Customization Options

While `nrow`, `byrow`, and `legend.position` address the primary structural and spatial concerns, ggplot2 offers a vast array of additional options for fine-tuning the appearance of the legend to match specific corporate or publication styling guides. These customizations fall primarily under the `theme function`, allowing meticulous control over every aesthetic element.

One common refinement is adjusting the appearance of the legend text and title. Arguments such as `legend.title` and `legend.text` accept `element_text()` calls, enabling control over font size, color, and face (bold, italic). For example, to make the legend text smaller and the title bold, you might add `theme(legend.text = element_text(size=8), legend.title = element_text(face="bold"))` to your plot code. This precision ensures that the legend integrates seamlessly with the overall design without visually competing with the axes or plot title.

Furthermore, the background and key appearance can be modified. `legend.background` controls the background fill color and border of the entire legend box, while `legend.key` controls the appearance of the background behind the individual symbols (the 'keys') within the legend. For advanced control over spacing, `legend.spacing.x` and `legend.spacing.y` can be used to manually adjust the horizontal and vertical gaps between legend items, which is often crucial when working with condensed multi-row layouts to prevent clutter.

A list of powerful legend customization elements includes:

`legend.title`: Controls the text style of the legend title.
`legend.text`: Governs the font style and size of the legend labels.
`legend.background`: Defines the background color and outline of the legend box.
`legend.key.size`: Adjusts the size of the symbol keys.
`legend.direction`: Can be set to `"horizontal"` or `"vertical"`, though this is often overridden by `nrow`/`byrow` when using `guide_legend()`.

Summary of Best Practices for Legend Management

Effective legend management in ggplot2 transitions a simple visualization into a professional-grade graphic. The key takeaway is that relying solely on ggplot2 defaults is often suboptimal when dealing with more than five categorical levels. Proactive intervention using specialized guide functions is necessary to optimize layout and spatial efficiency.

Best practices dictate a methodical approach:

Identify the Aesthetic Mapping: Determine which aesthetic (e.g., color, shape, size) generates the legend that needs modification.
Invoke `guides()`: Use `guides(aesthetic = guide_legend(...))` to gain control over that specific guide.
Define Structure: Set the `nrow` argument to condense the legend into the optimal number of rows, calculating the necessary value based on the total number of categories. Simultaneously, set `byrow = TRUE` for standard horizontal filling.
Optimize Position: Use `theme(legend.position = 'bottom')` or `'top'` in conjunction with the multi-row layout to reclaim the primary plot area.
Refine Appearance: Apply additional `theme()` arguments (like `legend.text` or `legend.key.size`) to ensure the legend's visual properties enhance, rather than detract from, the data visualization.

Mastering these steps ensures that any visualization created using ggplot2 maintains high standards of clarity, professionalism, and effective communication, regardless of the complexity of the underlying categorical data.

The following tutorials explain how to perform other common operations in ggplot2: