How to Hide a Legend in ggplot2: A Quick Guide

Authored by

stats writer

December 5, 2025

The **ggplot2** package, based on the grammar of graphics, is the industry standard for creating elegant and informative visualizations in R. By default, when a visual aesthetic (like color, shape, or size) is mapped to a variable in the data, **ggplot2** automatically generates a **legend** to ensure interpretability. However, there are common scenarios--such as generating plots for a presentation, simplifying complex dashboards, or preparing figures for publication where space is premium--where the legend needs to be suppressed or completely removed.

Fortunately, **ggplot2** provides two primary, efficient methods for achieving this goal. The choice between these methods depends on whether you wish to hide all legends globally across the entire plot, or if you only need to suppress the legend associated with a single geometric object (a specific `geom` layer). Understanding these nuances is key to producing clean, finalized visualizations.

The first method involves setting the argument **`show.legend = FALSE`** within a specific geometry function, such as **`geom_point()`** or `geom_bar()`. This is useful for selectively hiding a legend when multiple aesthetics are mapped. The second, more comprehensive approach involves modifying the plot's overall theme using **`theme()`** and setting the `legend.position` argument to `"none"`. This latter method is generally preferred for complete, universal removal.

Core Syntax for Global Legend Removal

For most use cases requiring the complete disappearance of all explanatory legends, applying a modification to the plot's theme structure is the most reliable approach. This leverages the **`theme()`** function, which controls all non-data visual aspects of the plot, including axis titles, panel background, and, critically, the placement of the legend.

When you use the `theme()` function to set `legend.position`, you are making a global styling choice that affects the entire visual output, regardless of how many different legends might be generated by the various geometric layers used in the visualization. This ensures a clean slate, freeing up valuable space around the visualization area.

You can use the following syntax to remove a legend from a plot in **ggplot2**:

```
ggplot(df, aes(x=x, y=y, color=z)) +
geom_point() +
theme(legend.position="none")
```

By specifying **`legend.position="none"`** within the `theme()` layer, you are instructing **ggplot2** to override the default placement (usually "right" or "bottom") and suppress the rendering of all legends associated with the plot's aesthetics. This is a global setting that affects every legend generated by the preceding geometry layers. We will demonstrate this universal approach using a

practical **scatterplot** example below.

Setting Up the Environment and Data

Before diving into the visualization itself, it is necessary to establish a working **data frame** that maps multiple variables, ensuring that at least one categorical variable is used to define an aesthetic, thereby forcing **ggplot2** to generate a legend by default. This example uses simulated sports statistics to showcase different player positions (Guard, Forward, Center).

The process begins by loading the necessary R libraries and creating the foundational data structure. We use the `data.frame()` function to construct a tidy dataset where each row represents an observation and columns represent variables such as `assists`, `points`, and `position`. This structure is essential for the seamless functioning of the **ggplot2** package, where variables are explicitly mapped to visual properties.

The variable `position` is the categorical factor we will map to the `color` aesthetic, ensuring that the initial plot requires a legend for clear differentiation of the data points. If this categorical mapping were absent, no legend would be generated in the first place, making the subsequent removal steps redundant.

Step 1: Create the Data Frame

First, let's create a data frame in R:

#create data frame
df <- data.frame(assists=c(3, 4, 4, 3, 1, 5, 6, 7, 9),
points=c(14, 8, 8, 16, 3, 7, 17, 22, 26),
position=rep(c('Guard', 'Forward', 'Center'), times=3))

#view data frame
df

assists points position
1 3 14 Guard
2 4 8 Forward
3 4 8 Center
4 3 16 Guard
5 1 3 Forward
6 5 7 Center
7 6 17 Guard

8 7 22 Forward
9 9 26 Center

Step 2: Create a Plot Using ggplot2 (Default Legend Included)

Once the data is prepared, the next logical step is to construct the visualization using the **ggplot2** workflow. We map the continuous variables `assists` and `points` to the X and Y axes, respectively, within the primary `aes()` call. Crucially, we map the categorical variable `position` to the `color` aesthetic, which is the action that triggers the automatic generation of a legend by **ggplot2**.

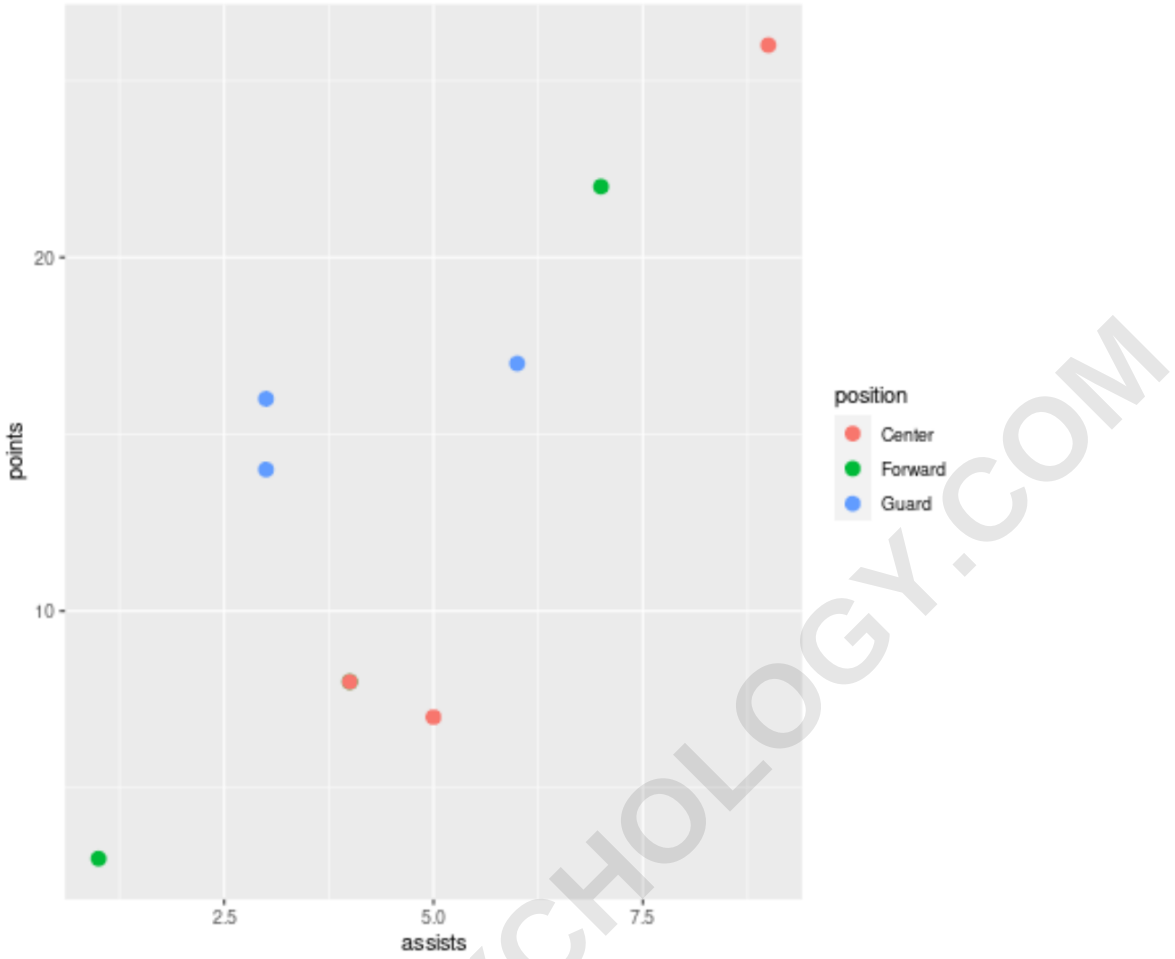
We use the **`geom_point()`** function to create a **scatterplot**, which is an excellent visualization for examining the relationship between two continuous variables while simultaneously differentiating subgroups based on color. Notice that in the code block below, no theme modifications are applied, resulting in the default display behavior, which includes a legend.

This initial plot serves as our baseline, confirming that the aesthetic mapping is correctly implemented and that the package successfully generates the required explanatory element before we proceed with its removal.

Next, let's use **ggplot2** to create a simple **scatterplot**:

```
library(ggplot2)

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



As illustrated in the resulting graphic, **ggplot2** automatically includes a legend on the right side of the plot. This legend clearly maps the colors assigned to each of the three player positions, making it easy for the viewer to interpret the distinct clusters of data points. This default behavior prioritizes clarity and comprehensive data display, but for production use, it often needs to be suppressed.

Step 3: Implementing Global Legend Removal using theme()

To eliminate this legend entirely, we introduce the `theme()` layer and specify the `legend.position = "none"` argument. This critical command must be added after all aesthetic mappings and geometry layers have been defined, as it acts as a final styling override for the entire visualization structure. This modification successfully removes the explanatory keys and labels, minimizing visual clutter without altering the core data representation.

When applying `legend.position="none"`, you should confirm that the removal of the legend does not hinder the audience's ability to understand the plot. Legends are essential when colors or shapes represent meaningful data categories. If you are removing the legend, ensure that the groups are either clearly labeled directly on the plot using annotation functions like `geom_text()` or

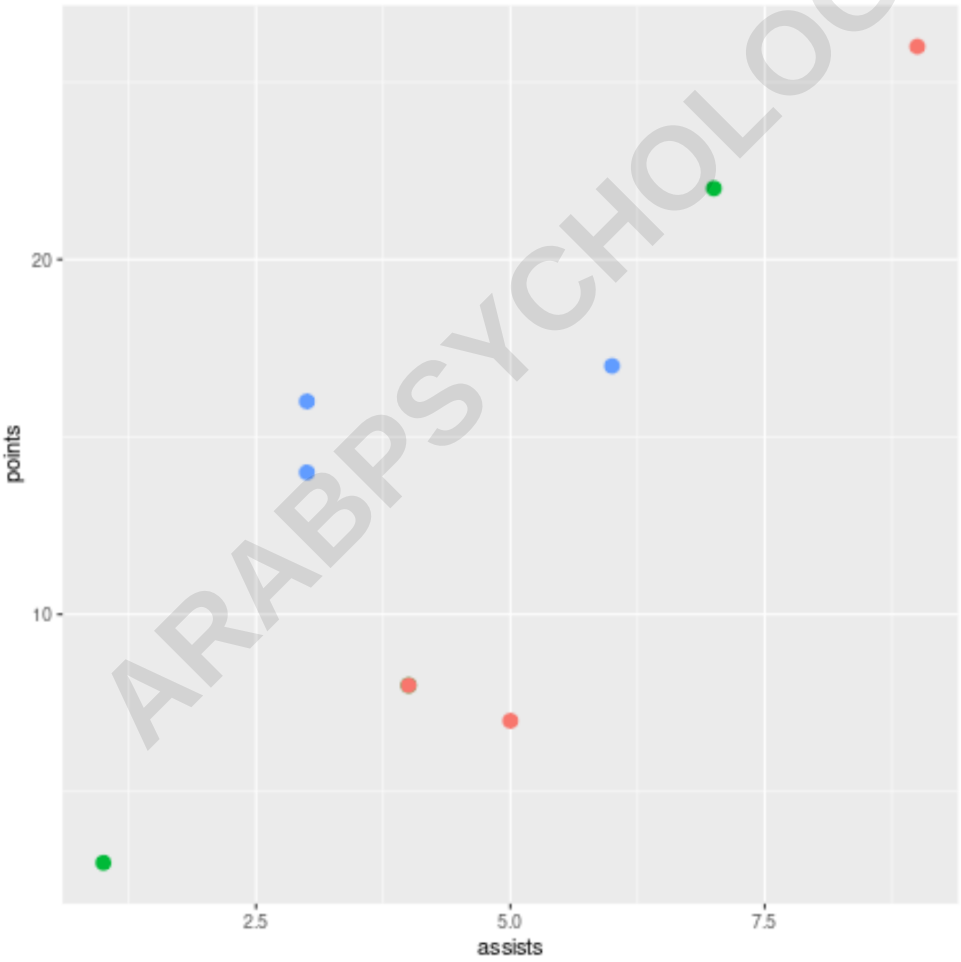
that the context is explicitly provided elsewhere in the accompanying text or presentation slides.

This method is highly scalable. Even if we had added additional geometric layers with their own aesthetic mappings (e.g., different shapes or line types), setting this single theme parameter would override and hide all resulting legends simultaneously.

Next, let's use `legend.position="none"` to remove the legend from the plot:

library(ggplot2)

#create scatterplot with no legend
ggplot(df, aes(x=assists, y=points, color=position)) +
geom_point(size=3) +
theme(legend.position="none")



The legend has been completely removed from the plot.

Method B: Selective Legend Removal using show.legend = FALSE

While `legend.position = "none"` is effective for global removal, there are times when a plot might feature multiple geometric layers, and only the legend associated with a specific layer needs to be hidden. This is where the argument **`show.legend = FALSE`** becomes indispensable. Unlike the theme modification, this argument is placed directly within the geometry function itself (e.g., **`geom_point()`**, `geom_line()`, or `geom_bar()`).

This method is particularly useful when combining layers. For instance, imagine plotting individual data points color-coded by group (which requires a legend) and then adding a reference mean line colored black (which should not generate its own legend entry). By adding `show.legend = FALSE` to the mean line geometry, we prevent **ggplot2** from creating a redundant entry for the non-variable mapping.

This selective approach offers granular control, ensuring that only necessary explanatory elements are displayed. If a plot contains only one aesthetic mapping that generates a legend, using `show.legend = FALSE` within that single `geom` function will achieve the same visual result as setting `legend.position = "none"`, but the latter is generally considered syntactically cleaner for universal removal.

 # Example of selective legend removal
ggplot(df, aes(x=assists, y=points, color=position)) +
geom_point(size=3) + # Legend for 'position' (color) is shown
geom_smooth(method="lm", se=FALSE, color="black", show.legend = FALSE) # Prevents legend for geom_smooth

Advanced Legend Positioning and Styling Alternatives

While complete removal is often the goal, it is worth noting that the `legend.position` argument in `theme()` offers a powerful intermediate solution for space optimization. Instead of eliminating the legend, you might choose to reposition it to a less intrusive area, such as inside the plotting panel itself, or simply move it from the default right side to the bottom.

The `legend.position` argument accepts several standard character strings: `"right"` (default), `"left"`, `"top"`, and `"bottom"`. For fine-tuned placement, particularly when integrating the legend into the data visualization area, it can also accept a numeric vector of length two, defining the exact X and Y coordinates relative to the plot area (ranging from 0 to 1). For example, `theme(legend.position = c(0.8, 0.2))` places the legend inside the plot area near the bottom-right corner.

Furthermore, if you need to retain the legend but reduce its visual footprint, the `theme()` function allows for detailed control over every appearance setting, including text size and key dimensions. Using commands like `theme(legend.title = element_text(size=8), legend.text = element_text(size=7), legend.key.size = unit(0.5, "cm"))` can significantly compress the legend, making full removal less necessary and maintaining necessary context for the audience.

Conclusion: Generating Finalized, Clean Visualizations

The ability to precisely control the visual components of a plot is essential for effective data storytelling. In **ggplot2**, the two primary mechanisms for legend suppression--the global `theme(legend.position = "none")` and the layer-specific `show.legend = FALSE`--provide the flexibility needed to finalize graphics for any medium.

Always prioritize clarity. If the plot is self-explanatory or if contextual information is provided through other means (like direct annotation or accompanying text), removing the legend streamlines the visualization and focuses the viewer's attention on the data patterns. By integrating these specific commands into your standard workflow, you can ensure your final plots are both aesthetically pleasing and maximally informative.

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