How to Summarize Data with `stat_summary()` in ggplot2

Authored by

stats writer

January 15, 2026

RECOMMENDED CITATION

stats writer (2026). *How to Summarize Data with `stat_summary()` in ggplot2*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=126178

The `stat_summary()` function within the renowned ggplot2 package is an indispensable tool for data visualization in R. It is specifically designed to efficiently calculate summary statistics for subsets of data and integrate those metrics directly into a plot. Instead of manually grouping and summarizing your data frame, `stat_summary()` handles this heavy lifting internally, producing visualizations that represent central tendencies, distributions, or extreme values with minimal code. This function allows users to quickly generate informative plots--such as points, lines, or bars--that display summary measures like the **mean**, **median**, or **standard deviation**, significantly accelerating the exploratory data analysis process and providing immediate insight into overall data trends and group comparisons. The convenience of customizing summary measures and adding visual cues like error bars makes `stat_summary()` a flexible and powerful component of the ggplot2 ecosystem.

The Purpose of stat_summary()

Generating visualizations often requires summarizing raw data first, especially when dealing with categorical variables or time series where raw data points might clutter the plot. The **`stat_summary()`** function is the direct implementation of the "statistics" layer in the Grammar of Graphics, enabling transformation of data before mapping it to aesthetics. By default, this function groups the data based on the categorical variable assigned to the x-axis, calculates the specified summary statistic for the continuous variable (y-axis) within each group, and then plots the result using the defined geometric object. Understanding this workflow is key to leveraging `stat_summary()` effectively for high-level data interpretation.

This powerful function streamlines the process of visualizing aggregate metrics. You can use the **`stat_summary()`** function in ggplot2 to create visualizations that display summary metrics of specific variables in a data frame, allowing for direct comparison of group performance or characteristics. This approach is highly efficient because it removes the intermediate step of creating a separate summary table before plotting.

The flexibility of **`stat_summary()`** is derived from its ability to accept various statistical functions and geometric elements. Whether you need to highlight the arithmetic average, the median, or a specific quantile, `stat_summary()` adapts to the required analysis and maps the output seamlessly onto the visualization canvas, maintaining code cleanliness and clarity.

Setting Up the Sample Data

To demonstrate the practical application of **`stat_summary()`**, we will utilize a simple, yet representative, data frame in R. This data frame contains performance metrics (points) across several distinct teams. This structure allows us to easily visualize how various summary statistics differ across these categorical groups. We rely on the **ggplot2** library for visualization and often

pair it with **dplyr** for pre-processing steps, although `stat_summary()` often negates the need for explicit summarizing steps using `dplyr::summarise()`.

The following R code establishes the sample data frame named `df`. This structure includes two variables: `team`, which is a categorical identifier, and `points`, which is the numerical variable we intend to summarize and visualize. Reviewing the raw data output below helps set the context for the subsequent visualizations, showing the initial distribution of scores before summarization occurs.

#create data frame
df = data.frame(team=rep(c('A', 'B', 'C'), each=4),
points=c(8, 12, 4, 6, 26, 21, 25, 20, 9, 18, 14, 14))

#view data frame
df

team points
1 A 8
2 A 12
3 A 4
4 A 6
5 B 26
6 B 21
7 B 25
8 B 20
9 C 9
10 C 18
11 C 14
12 C 14

Key Arguments: fun and geom

Before diving into specific examples, it is critical to understand the primary parameters governing the behavior of **`stat_summary()`**: **`fun`** and **`geom`**. The **`fun`** argument (short for function) dictates which statistical operation should be applied to the data subsets. This argument accepts a string specifying a standard R summary function (like `'mean'`, `'median'`, `'min'`, or `'max'`), or a custom R function defined by the user. The function specified here is what is applied individually to the `points` vector for each level of the `team` variable.

The **`geom`** argument, conversely, specifies the geometric object used to render the calculated

summary statistic on the plot. This is fundamental to the visual style of the output. Common choices include `'bar'` (for bar plots), `'point'` (for marking the summary location), `'line'` (for visualizing trends), or `'errorbar'` (when paired with specialized statistical functions). It is essential to choose a **`geom`** that appropriately represents the calculated statistic; for instance, bars are excellent for comparing aggregate magnitudes, while points are better for overlaying metrics onto raw data distributions.

If you need to summarize multiple statistics or include complex error calculations (which require plotting y, ymin, and ymax), the related argument **`fun.data`** is used instead of `fun`. This argument requires a function that returns a data frame containing the necessary output columns, providing the infrastructure needed for robust error visualization using geometries like `'errorbar'` or `'pointrange'`. The combination of the chosen statistical function and the geometric representation determines the final output visualization.

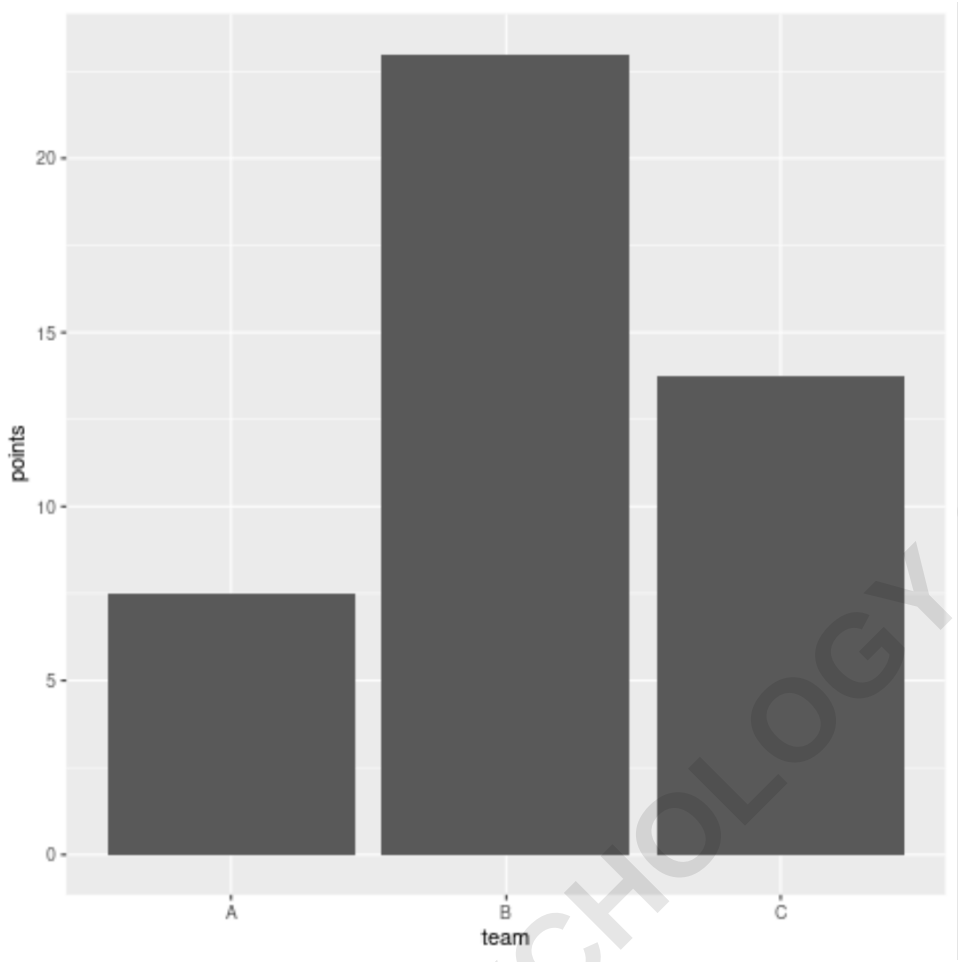
Example 1: Visualizing Mean Values with a Bar Plot

A bar plot is a classic and effective method for comparing a single summary statistic--such as the mean--across different categorical groups. In this first practical demonstration, we use **`stat_summary()`** to calculate and display the average `points` scored by each `team`. This method provides an immediate visual comparison of central tendency across the three defined teams (A, B, and C).

We specifically set `fun='mean'` to calculate the arithmetic average of the points for each team subset, and `geom='bar'` to ensure the resulting statistic is mapped as the height of a bar. Notice how the input data stream uses the pipe operator (`%>%`) to feed the `df` data frame directly into the ggplot2 initialization. This code efficiently bypasses the need for manual data aggregation, allowing the visualization package itself to handle the statistical calculation within the visualization layer.

```
library(ggplot2)
library(dplyr)

#create bar plot to visualize mean points by team
df %>%
ggplot(aes(x=team, y=points)) +
stat_summary(fun='mean', geom='bar')
```



As illustrated by the resulting visualization, the height of each bar corresponds exactly to the mean `points` value achieved by that specific team. Team B clearly shows the highest average score (23 points), while Team A has the lowest (7.5 points), and Team C sits in the middle (13.75 points). This example highlights the clarity provided by mapping the summary statistic to a recognizable geometric form like a bar, making group differences highly visible and easy to interpret.

Example 2: Visualizing Mean Values with a Scatter Plot

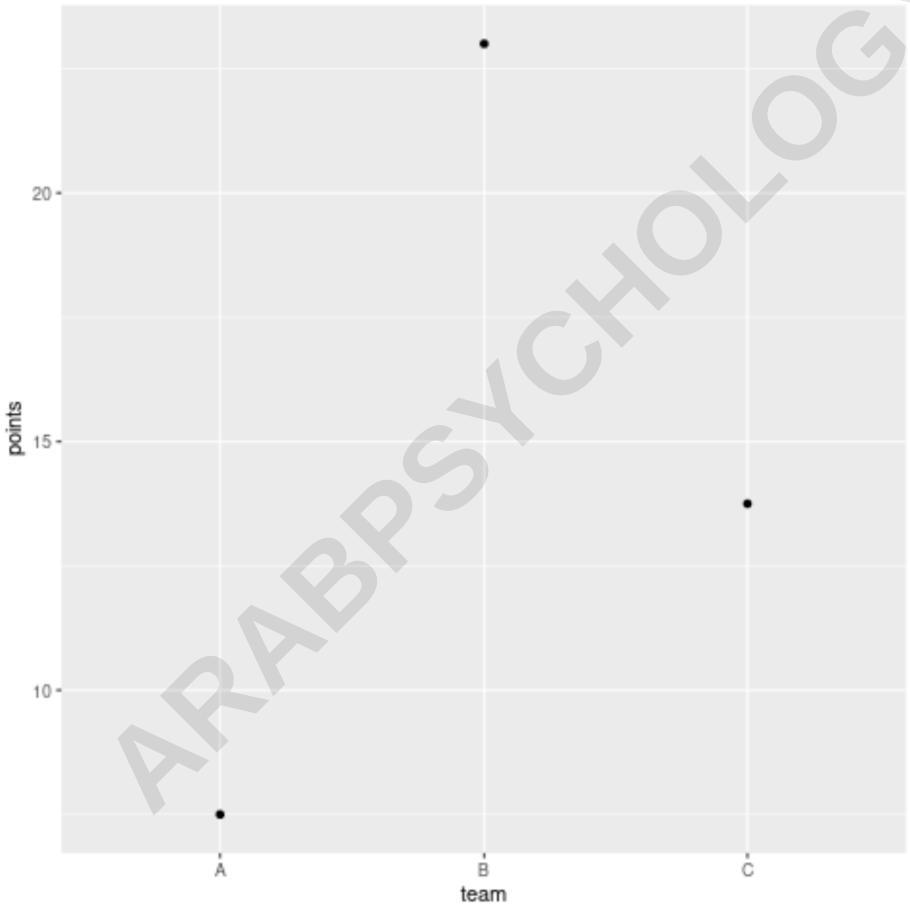
While bar plots are ideal for comparing magnitudes, sometimes a simpler point representation is preferred, particularly if you plan to layer multiple summary statistics or incorporate other geometric elements onto the same plot. In this second scenario, we maintain the goal of visualizing the mean `points` per team, but we change the geometric representation to points using `geom='points'`. This results in a cleaner plot where the summary statistic is represented simply by a single dot corresponding to the calculated average value on the y-axis for each category on the x-axis.

The structure of the code remains nearly identical to the previous example, emphasizing the interchangeability of the **`geom`** argument within **`stat_summary()`**. By setting `geom='points'`, we

instruct ggplot2 to plot the summary result (the mean) as individual data markers rather than aggregated bars. This choice is often beneficial when the focus is less on cumulative magnitude and more on the precise location of the calculated center, or when preparing to overlay the summary on top of the original raw data points.

```
library(ggplot2)
library(dplyr)

#create plot with points to visualize mean points by team
df %>%
ggplot(aes(x=team, y=points)) +
stat_summary(fun='mean', geom='points')
```



The resulting plot confirms that the calculated mean values are identical to those in Example 1, demonstrating that changing the `geom` only affects the visualization style, not the underlying statistical calculation performed by the `fun` argument. Using points is particularly useful if you intend to overlay these summary statistics onto a jittered plot of the raw data, allowing viewers to see both the distribution of individual scores and the calculated central tendency simultaneously.

We explicitly used the **`geom`** argument within the **`stat_summary()`** function to specify the desired geometric shape for the plot output.

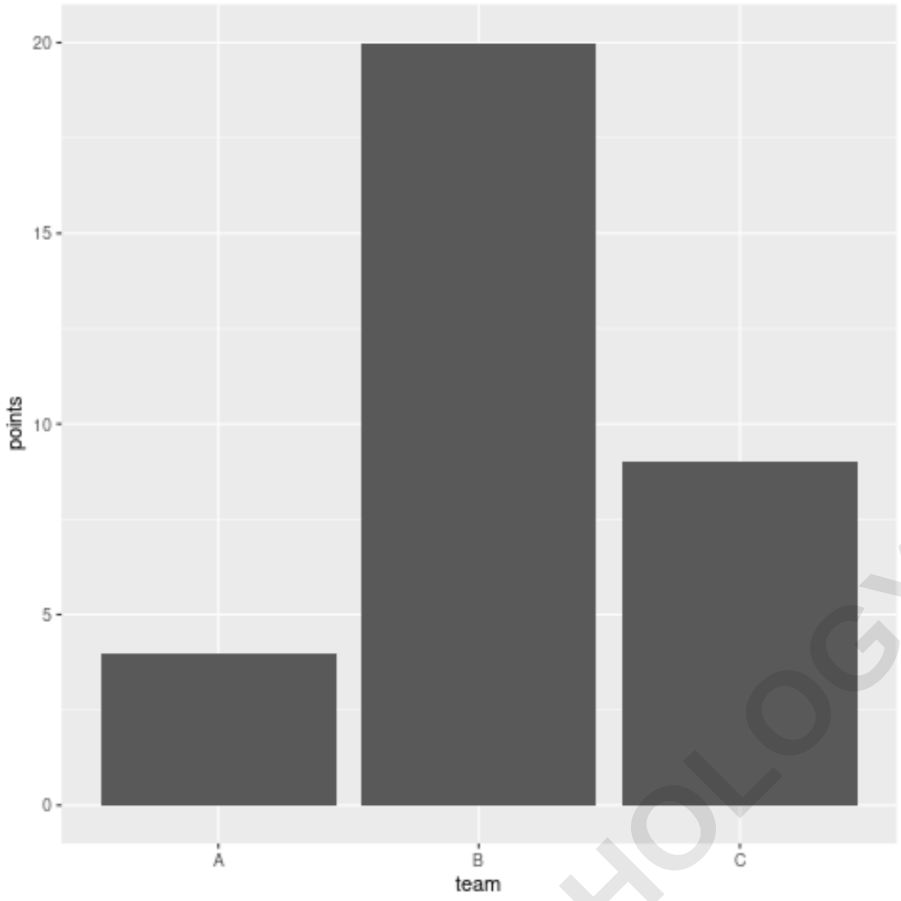
Example 3: Calculating Extreme Values (Minimum)

The flexibility of **`stat_summary()`** extends far beyond calculating measures of central tendency like the mean. This function can easily calculate and visualize measures of dispersion or extreme values, such as the minimum or maximum score. In this example, we shift our focus to identifying the lowest score, or **minimum value**, achieved by each team. Understanding minimums is crucial in performance analysis to identify baseline results or potential outliers that might drag down overall performance.

To implement this, we simply adjust the **`fun`** argument within **`stat_summary()`** to `'min'`. We return to using `geom='bar'` to visualize these minimum scores, providing a clear visual representation of the floor performance for each team. This variation underscores how simple it is to switch the statistical focus using the **`fun`** parameter, making complex statistical comparisons straightforward while keeping the visualization style consistent with our previous bar plots.

```
library(ggplot2)
library(dplyr)

#create bar plot to visualize minimum points by team
df %>%
ggplot(aes(x=team, y=points)) +
stat_summary(fun='min', geom='bar')
```



The resulting bar plot displays the lowest score recorded for Team A (4 points), Team B (20 points), and Team C (9 points). By changing **`fun`** to `'min'`, we have effectively transformed the visualization to highlight the weakest performance in each category, confirming Team B's overall strength even when considering their lowest score. Similarly, setting `fun='max'` would allow us to visualize the highest scores achieved. We explicitly used the **`fun`** argument within the **`stat_summary()`** function to specify that we required the minimum as the summary statistic.

Example 4: Incorporating Measures of Uncertainty (Error Bars)

In rigorous statistical visualization, it is rarely sufficient to display only the central tendency; measures of uncertainty, variability, or error are often required to communicate the reliability of the calculated summary. **`stat_summary()`** is highly capable of generating visualizations that include error bars, typically by utilizing the **`fun.data`** argument instead of `fun`.

The **`fun.data`** argument requires a function that returns three specific output columns: `y` (the center point), `ymin` (the lower bound), and `ymax` (the upper bound). ggplot2 provides several built-in functions optimized for this, such as `mean_se` (for mean plus/minus standard error) or `mean_sdl` (for mean plus/minus standard deviation). When using `fun.data`, we commonly set

`geom='errorbar'` to visualize the uncertainty range, and often pair it with `geom='point'` or `geom='line'` (using a separate `stat_summary()` layer) to plot the center point itself.

For instance, to visualize the mean points along with the standard error of the mean for each team, we would use the following code structure. Notice we use two separate `stat_summary()` layers: one for the points and one for the error bars, ensuring a complete and statistically sound visualization where the error bars indicate the expected range of the true mean based on the sample data.

library(ggplot2)
library(dplyr)

Plotting Mean (Point) and Standard Error (Errorbar)
df %>%
ggplot(aes(x=team, y=points)) +
Layer 1: Plot the mean as a point
stat_summary(fun='mean', geom='point', size=3) +
# Layer 2: Plot the standard error as error bars
stat_summary(fun.data='mean_se', geom='errorbar', width=0.2)

Example 5: Using Custom Summary Functions

One of the most powerful features of ggplot2 is its extensibility. If the built-in summary functions (like mean, median, min) do not meet specific analytical requirements, **`stat_summary()`** permits the use of custom R functions. This allows data scientists to apply specialized metrics, such as trimmed means, robust estimators, or non-standard quantiles, directly within the visualization pipeline without altering the base data frame.

To use a custom function with the **`fun`** argument, the function must be defined beforehand and must accept a numeric vector as input and return a single numeric value as output. For example, if we wanted to calculate a trimmed mean (excluding the top and bottom 10% of scores) for each team, we would define a custom function and pass its name (without quotes) to the `fun` argument. This capability ensures that `stat_summary()` remains relevant even for highly specialized statistical tasks, providing a highly tailored view of the data.

 # Define a custom function for 10% trimmed mean
trimmed_mean <- function(x) {
mean(x, trim = 0.1)
}

Use the custom function in stat_summary()

```
df %>%
ggplot(aes(x=team, y=points)) +
stat_summary(fun=trimmed_mean, geom='point', size=4, color='blue')
```

Conclusion: Streamlining Visualization with stat_summary()

The `stat_summary()` function is far more than a simple plotting tool; it is a core statistical layer that dramatically streamlines the visualization workflow. By integrating data aggregation and visualization into a single layer, it allows users to rapidly explore summary metrics across categorical variables without the need for manual pre-processing steps. Whether you need to compare means using bar plots, highlight extreme values using points, or communicate variability using error bars derived from custom or standard functions, `stat_summary()` provides the required flexibility and efficiency essential for modern data analysis.

Mastering the use of the **`fun`** (or **`fun.data`**) and **`geom`** arguments allows for precise control over both the statistical calculation and the resulting aesthetic representation. This level of control ensures that visualizations are not only visually appealing but also statistically accurate and contextually relevant. For anyone relying on ggplot2 for data analysis, integrating `stat_summary()` into their workflow is essential for generating clean, statistically accurate, and highly informative graphics quickly and efficiently.

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