Learning to Visualize Data: A Step-by-Step Guide to Creating Relative Frequency Histograms in R

Authored by

Mohammed looti

November 8, 2025

The **relative frequency histogram** stands as a cornerstone graphical tool in statistical analysis, providing an intuitive visual representation of how observations are distributed across a numerical range. Crucially, it displays the proportion or percentage of a **data set** that falls within specific, contiguous intervals, commonly known as bins. Unlike traditional frequency histograms, which plot raw counts, the relative frequency approach normalizes the data, making it exceptionally valuable for comparing distributions from samples of vastly different sizes, as the vertical axis consistently represents normalized density or percentages.

This comprehensive tutorial is dedicated to mastering the construction of publication-quality relative frequency **histograms** within the **R programming environment**. We will leverage the capabilities of the robust **lattice package**, which specializes in generating sophisticated, multi-panel statistical graphics. The core functionality we will explore is provided by the versatile **histogram()** function. Understanding its primary syntax and arguments is the first step toward effective data visualization in R.

The general syntax for invoking the function is straightforward:

histogram(x, type)

The function relies on two principal arguments to generate the output:

x: This argument mandates the provision of the numerical vector containing the raw, continuous **data** that requires statistical summarization and visualization.
type: This optional, yet critical, argument determines the specific method of relative frequency representation displayed on the y-axis. Options include **percent** (the default setting), **count** (which reverts to standard, absolute frequency), and **density** (which calculates normalized probability density).

Loading the Lattice Package and Generating Default Output

Before leveraging the sophisticated charting features of the **lattice** package, it is mandatory to load it into the active R session. If you are running this code for the first time, ensure the package is installed using the standard R command, `install.packages("lattice")`. Loading the package makes the **histogram()** function accessible and ready for use in the **R programming environment**.

library(lattice)

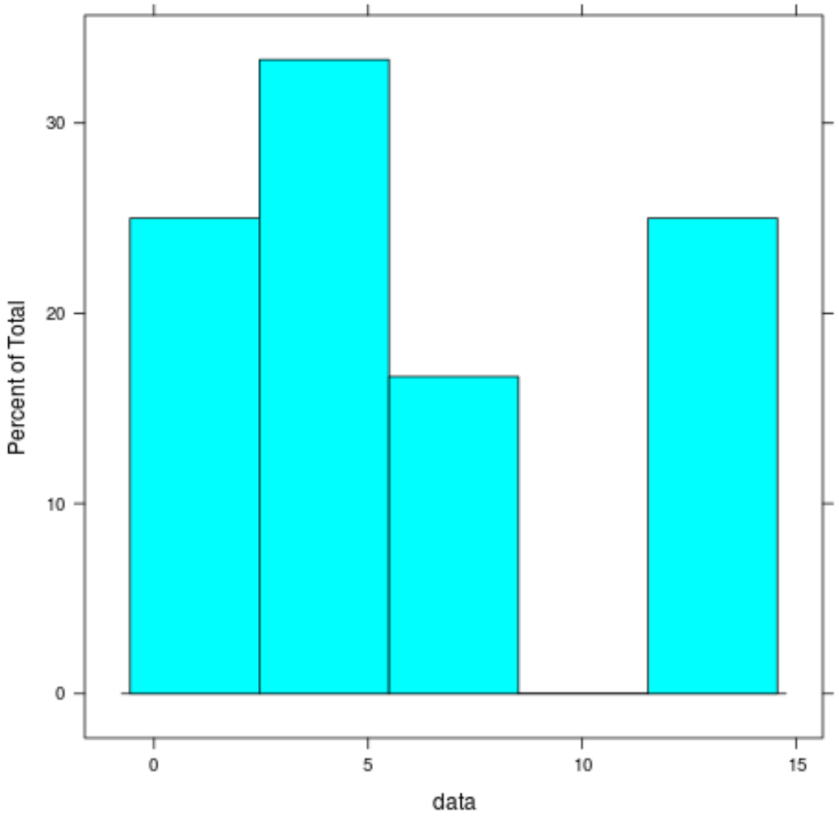
When executed without specifying the `type` argument, the **histogram()** function defaults to generating a relative frequency visualization where the vertical axis represents the **percent** of total observations falling into each interval. To demonstrate this foundational behavior, we will first

define a small, representative sample **data** vector, simulating hypothetical scores or measurements obtained during an experiment or game.

```
# Define a sample data vector representing points per game
data <- c(0, 0, 2, 3, 4, 4, 5, 6, 7, 12, 12, 14)

# Generate the default relative frequency histogram using the sample data
histogram(data)
```

The resulting graphic immediately displays the distribution of the data, using automatic binning rules determined by the **lattice** package. Although functional, this initial plot lacks the necessary labeling for clear interpretation, which necessitates the next steps in customization.



Customizing Visual Aesthetics for Clarity and Publication Readiness

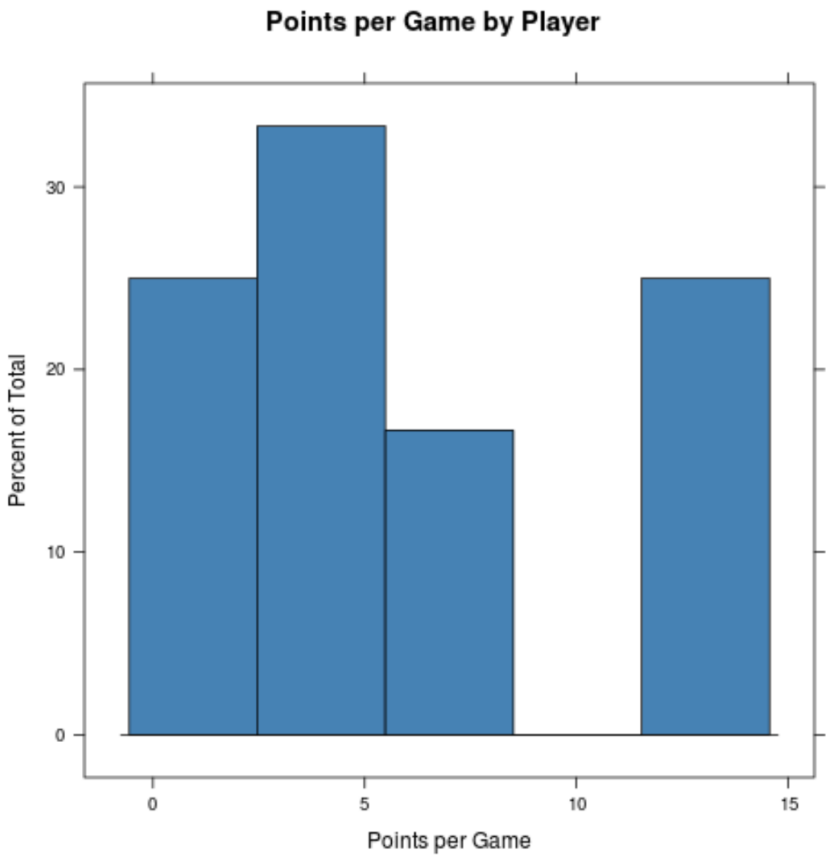
While R's default graphical output is computationally sound, high-quality statistical visualization demands clarity and context. A histogram is only effective if its titles and labels accurately communicate the underlying information. The **histogram()** function within the **lattice** package provides extensive support for customizing the visual presentation, ensuring the output is not only correct but also immediately interpretable and suitable for reports or publications.

To enhance the visual appeal and informational richness of our plot, we can utilize several key arguments to control its appearance. These arguments allow us to specify a descriptive title that encapsulates the plot's purpose, define precise labels for both axes, and select an appropriate fill color for the visualization bars.

main: Defines the overall, descriptive title placed at the top of the plot, summarizing the data distribution being visualized.
xlab: Specifies the label for the horizontal (x) axis, which represents the numerical variable being measured and partitioned into intervals.
ylab: Sets the label for the vertical (y) axis, confirming the metric used for relative frequency (e.g., "Percentage" or "Density").
col: Controls the fill color of the histogram bars, significantly enhancing the plot's visual distinction and aesthetic appeal.

Applying these arguments transforms the generic default plot into a meaningful graphic. The following code demonstrates how to incorporate context--specifically, that the data represents points scored per game by an athlete--using a strong title, clear axis labels, and a standard color like 'steelblue'.

```
# Apply comprehensive visual customization including titles and color
histogram(data,
main='Points per Game by Player',
xlab='Points per Game',
col='steelblue')
```



Defining Data Granularity: Customizing the Number of Bins

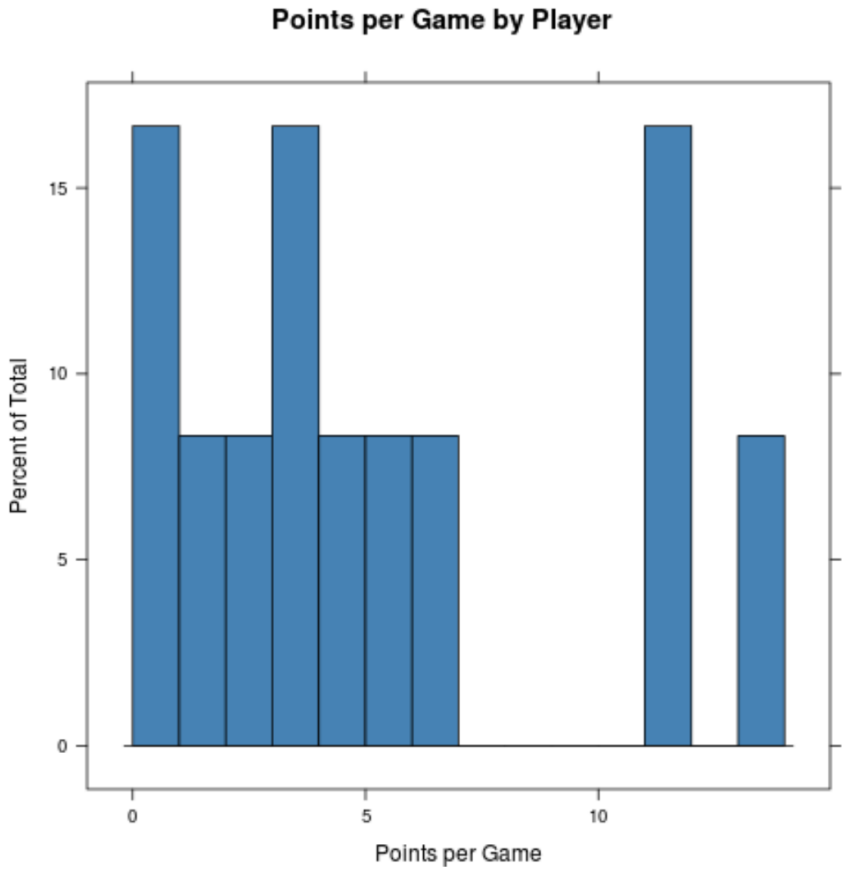
The interpretation of any **histogram** is profoundly dependent on the selection of the number of **bins**, which are the non-overlapping intervals used to categorize the continuous data. These bins define the granularity of the visualization; a poorly chosen bin count can either mask critical features of the distribution or introduce spurious noise. Effectively, the number of bins dictates whether the resulting plot appears smooth, reflecting general trends, or highly detailed, capturing minor fluctuations.

The **histogram()** function allows explicit control over this aspect through the use of the **breaks** argument. Determining the optimal number of bins requires a careful balancing act. If the number of breaks is too low, the visualization aggregates too much data, potentially obscuring bimodality or skewness. Conversely, an excessively high number of breaks results in narrow bins, causing the histogram to appear erratic and sensitive to minor data perturbations, thereby masking the true underlying shape of the data.

To showcase the effect of high granularity, we modify our previous code by setting **breaks=15**. This forces R to create fifteen distinct intervals across the range of the scores. The resulting plot offers a much finer-grained view of the distribution, making it easier to identify specific score

ranges where the relative frequency peaks or drops significantly.

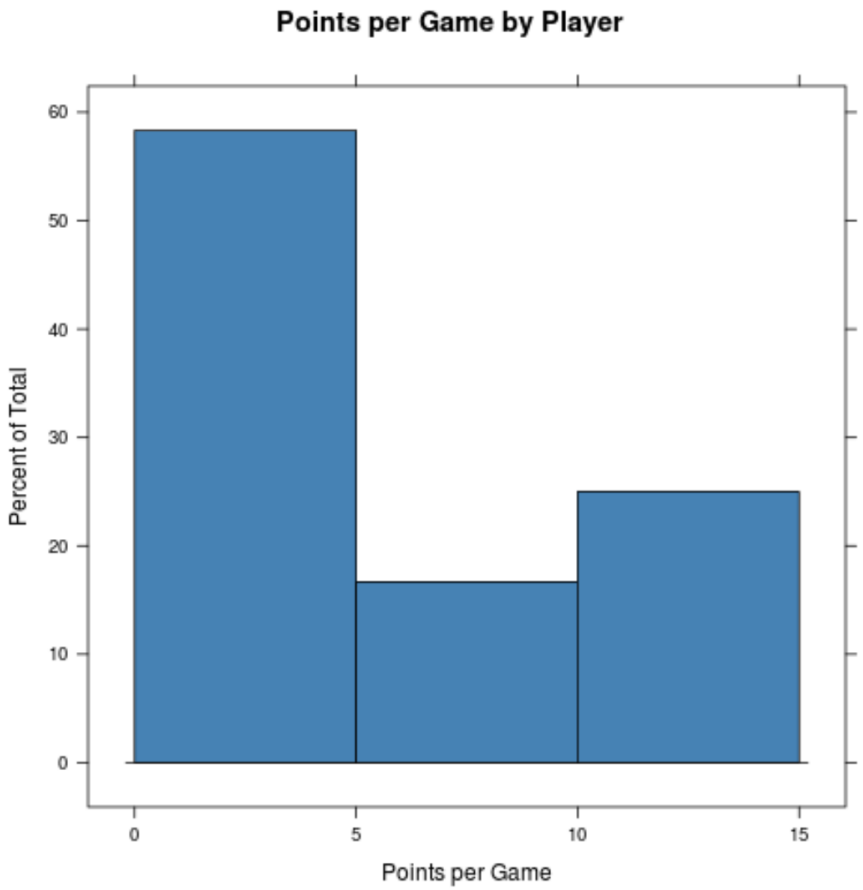
```
# Modify the number of bins to 15 for high granularity visualization
histogram(data,
main='Points per Game by Player',
xlab='Points per Game',
col='steelblue',
breaks=15)
```



Conversely, reducing the number of **bins** drastically aggregates the **data**. While this sacrifices detail, it can be useful for highlighting major, overarching trends and smoothing out minor variations. For instance, setting **breaks=3** forces the distribution into only three large categories. This generalization clearly illustrates the impact of bin selection on the overall visual interpretation of the distribution shape.

```
# Modify the number of bins to 3 for low granularity and generalized shape
histogram(data,
main='Points per Game by Player',
xlab='Points per Game',
```

```
col='steelblue',
breaks=3)
```



Selecting the Vertical Axis Metric: The 'Type' Argument

A key feature of the **histogram()** function is its flexibility in defining the metric used on the vertical (y) axis via the **type** argument. This allows the user to switch seamlessly between different measures of frequency, moving beyond the default percentage view to suit specific analytical requirements. A thorough understanding of these options is crucial for ensuring the statistical output is interpreted correctly.

The **type** argument supports three primary modes of frequency visualization:

percent: This mode displays the proportion of the total observations within each bin, scaled as a percentage. In a histogram of this type, the collective sum of the heights of all bars will invariably equal 100%. This is the preferred method for standard relative frequency reporting.
count: Choosing this option reverts the histogram to a display of absolute frequency. The y-axis then shows the raw, non-normalized count of data points corresponding to each bin. While this

sacrifices the "relative" comparison capability, it is useful for direct assessment of the sample size allocation across the range.

density: This option generates a probability density plot. In this normalized representation, the total area enclosed by all the histogram bars sums precisely to 1. Density plots are indispensable when the goal is to compare the empirical distribution shape against theoretical probability functions, such as overlaying a theoretical normal curve, providing a powerful tool for distributional assessment.

For instance, preparing the plot for advanced statistical comparison--perhaps by normalizing the area to unity--simply requires setting the argument as `type='density'`. This inherent flexibility confirms why the **lattice package** remains a highly robust choice for exploratory **data set** analysis within the **R programming environment**.

Conclusion and Best Practices for Bin Selection

The creation of an effective **relative frequency histogram** moves beyond mere code execution; it demands deliberate, thoughtful choices regarding visual presentation and, most importantly, bin configuration. Through thorough mastery of the **histogram()** function and the robust capabilities of the **lattice package**, users of **R** are empowered to maintain precise control over these critical parameters, ensuring their visualizations accurately reflect the underlying statistical distribution.

While manual tuning of the **breaks** argument provides flexibility, relying solely on subjective choice can introduce unintentional bias into the graphical representation. A recommended best practice for determining the ideal number of **bins** is to employ objective, rule-based methods. Standard statistical approaches, such as the well-established Sturges' formula or the robust Freedman-Diaconis rule, offer mathematically derived solutions for identifying the optimal number of breaks suitable for a specific **data set** size and spread. Integrating these methodologies significantly enhances the objectivity and reliability of the resulting visualization.

Related: To further optimize bin selection in R, consider utilizing built-in functions dedicated to this purpose, such as `nclass.Sturges` or `nclass.FD`, which programmatically identify and suggest the mathematically appropriate number of bins for your histogram based on established statistical rules.