How to Create a Relative Frequency Histogram in R: A Step-by-Step Guide

Authored by

stats writer

March 11, 2026

Understanding the Fundamentals of Data Visualization in R

The field of **data analysis** relies heavily on the ability to represent complex datasets through visual mediums. Among the most fundamental tools in the **R** programming language for this purpose is the **histogram**. While a standard histogram displays the raw frequency or "count" of observations within specific intervals, a **relative frequency** histogram provides a more nuanced perspective by displaying the proportion or percentage of total observations. This distinction is critical when comparing datasets of different sizes, as it normalizes the **y-axis**, allowing for a direct comparison of the **probability distribution** across various groups.

In **statistical computing**, **R** serves as a premier environment for generating these visualizations due to its extensive library of packages and flexible syntax. By utilizing **R**, researchers can transform raw data into insightful graphical representations that highlight **central tendency**, **skewness**, and **kurtosis**. This tutorial provides a comprehensive guide on how to construct these graphs, specifically focusing on the **lattice package** and the base R **hist()** function, ensuring that your data storytelling is both accurate and visually compelling.

The **relative frequency** is calculated by dividing the absolute frequency of a bin by the total number of observations in the **dataset**. This mathematical conversion is essential for **exploratory data analysis** because it shifts the focus from "how many" to "how likely." Whether you are working in **bioinformatics**, **finance**, or **social sciences**, mastering the creation of a relative frequency histogram in **R** is an indispensable skill that enhances the clarity of your findings and the robustness of your **statistical inference**.

Initial Data Preparation and Loading Procedures

Before generating any **visualization**, it is imperative to ensure that your data is correctly structured and imported into the **R environment**. The primary method for importing external data is through the **read.csv** function, which handles **comma-separated values** files. It is vital to verify that the columns intended for the histogram are in a **numeric** or **integer** format. Qualitative data or strings will not be compatible with the mathematical requirements of a histogram, as the process involves **data binning** along a continuous **number line**.

Once the data is loaded, a preliminary inspection using the **summary()** or **str()** functions is recommended. This step ensures that there are no **missing values** or **outliers** that could inadvertently distort the **bin widths** or the overall shape of the histogram. In **data science**, the quality of the output is strictly dependent on the quality of the input; therefore, cleaning your data by removing **NAs** or handling **anomalies** is a prerequisite for generating a meaningful **relative frequency histogram**.

After confirming the data's integrity, you can proceed to assign the target variable to a specific

object. In **R**, this is typically done using the assignment operator. For example, isolating a specific column from a **data frame** allows the **histogram()** function to process the values efficiently. Proper data handling at this stage prevents **syntax errors** and ensures that the transition from raw numbers to a **graphical representation** is seamless and reproducible in professional **reporting** environments.

Constructing a Base Histogram Using Core R Functions

The most straightforward method to create a histogram in **R** is by using the **hist()** function, which is part of the standard **graphics** package. By default, this function generates a frequency histogram. To convert this into a **relative frequency** version, the user must include the logical argument **freq = FALSE**. This specific command tells **R** to plot the **density** rather than the counts, effectively scaling the area of the bars to sum to one, which is the hallmark of a **probability density function**.

While the base **hist()** function is powerful, it requires additional parameters to reach a professional standard. Users can define the **x-axis** limits using the **xlim** argument and manage the granularity of the plot with the **breaks** argument. Understanding how **R** calculates these breaks is essential; it often defaults to **Sturges' rule**, but manual overrides are frequently necessary to better represent the underlying **distribution** of the data. This level of control is why **R** remains a favorite among **statisticians**.

To finalize a base histogram, the **plot()** function or the automatic output of **hist()** can be displayed in the **RStudio** viewer. Adding **metadata** such as titles and axis labels during this step is crucial for **accessibility**. A graph without a descriptive **main** title or clearly defined **units of measurement** can be misleading. By layering these arguments within the function call, you ensure that the **relative frequency** of your data is interpreted correctly by your audience, providing a clear window into the **stochastic** nature of your observations.

Leveraging the Lattice Package for Advanced Visualization

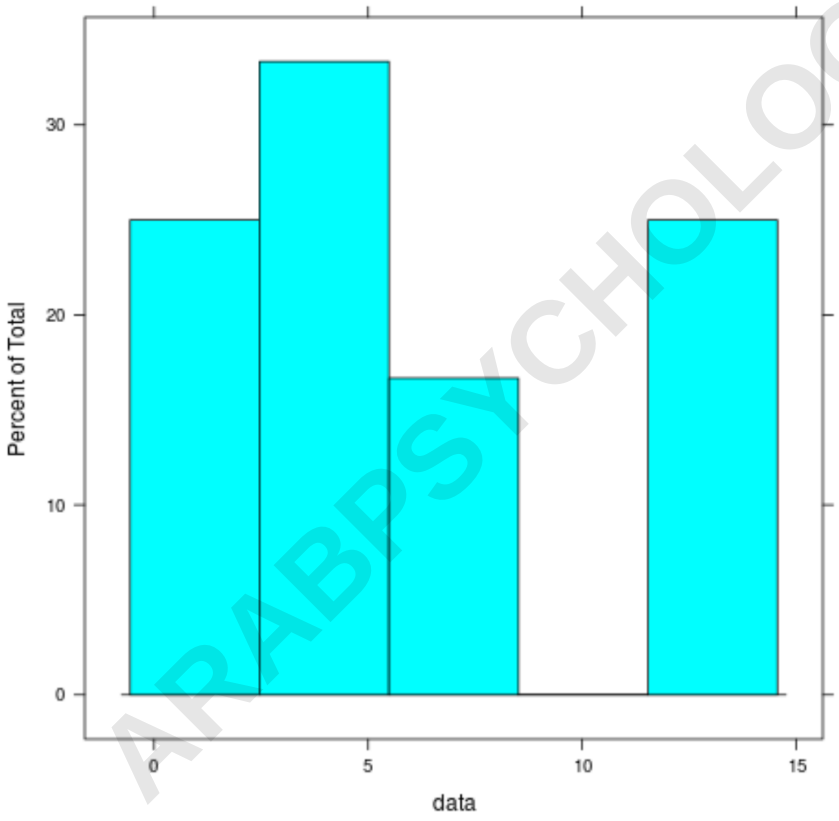
While base **R** provides the essential tools, the **lattice package** offers a more sophisticated framework for **multivariate data visualization**. Inspired by **Trellis graphics**, **lattice** allows for the creation of high-quality, complex plots with minimal code. To begin using this package, one must first install it from **CRAN** and then load it into the current session. The primary function for our needs is **histogram()**, which differs from the base function in both its **syntax** and its default output styling.

library(lattice)

One of the primary advantages of **lattice** is its handling of **formula syntax**, which allows users to easily split histograms across different **categorical variables**. For a single dataset, the **histogram()** function defaults to a **percent**-based **relative frequency histogram**. This is particularly useful because "percent" is often more intuitive for general audiences than "density," which is the default in base R. The following example demonstrates the default behavior of the package using a simple **numeric vector**.

#create data
data <- c(0, 0, 2, 3, 4, 4, 5, 6, 7, 12, 12, 14)

#create relative frequency histogram
histogram(data)



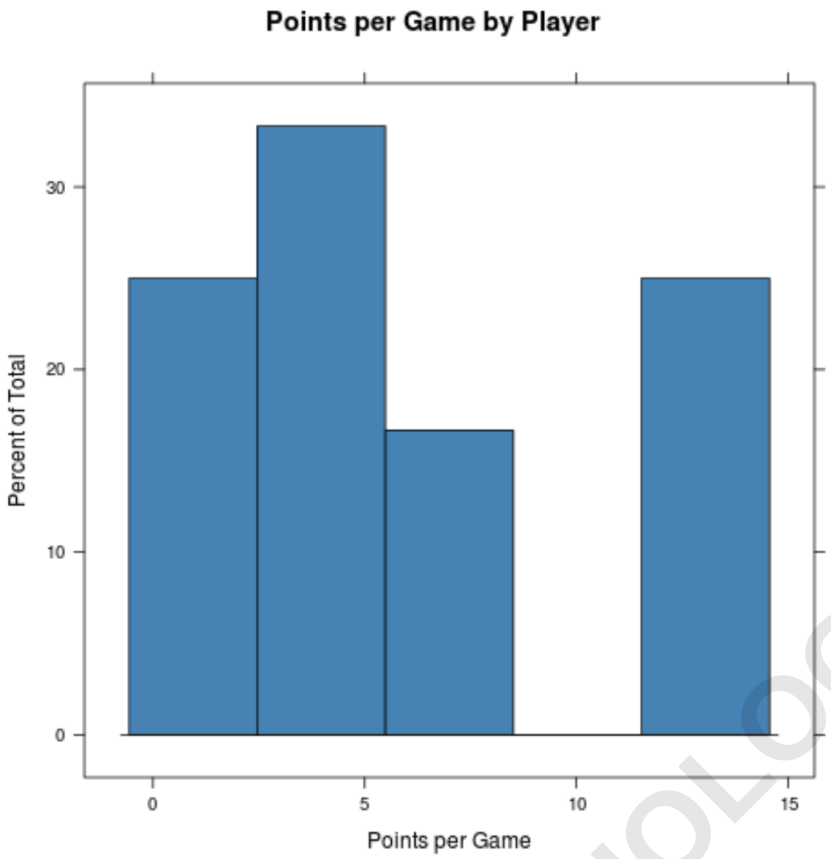
As seen in the resulting image, the **lattice** output is clean and formatted with a grid by default, which aids in the visual estimation of the **relative frequency** values. The **y-axis** clearly indicates the **percentage** of the data falling into each bin. This automated scaling is a significant time-saver during **rapid prototyping** of **statistical models**, as it eliminates the need for manual calculation of proportions or **normalization** of the data points.

Customizing Aesthetics for Professional Reporting

To ensure that a **relative frequency histogram** is suitable for publication or a business presentation, customization is required. The **histogram()** function in the **lattice package** accepts several arguments that modify the visual properties of the plot. The **main** argument is used to define the **header**, while **xlab** and **ylab** define the labels for the **horizontal** and **vertical** axes, respectively. Effective labeling is a cornerstone of **data communication**, ensuring that the viewer immediately understands what the **data** represents.

Furthermore, the **col** argument allows users to change the fill color of the bars, which can be used to align the visualization with a specific **brand identity** or to improve **contrast** for individuals with **color vision deficiency**. Choosing a professional color, such as "steelblue," can make the graph more visually appealing without distracting from the **statistical** information. The flexibility of **R** allows for the use of **hex codes** or standard color names to achieve the desired aesthetic effect.

#modify the histogram
histogram(data,
main='Points per Game by Player',
xlab='Points per Game',
col='steelblue')



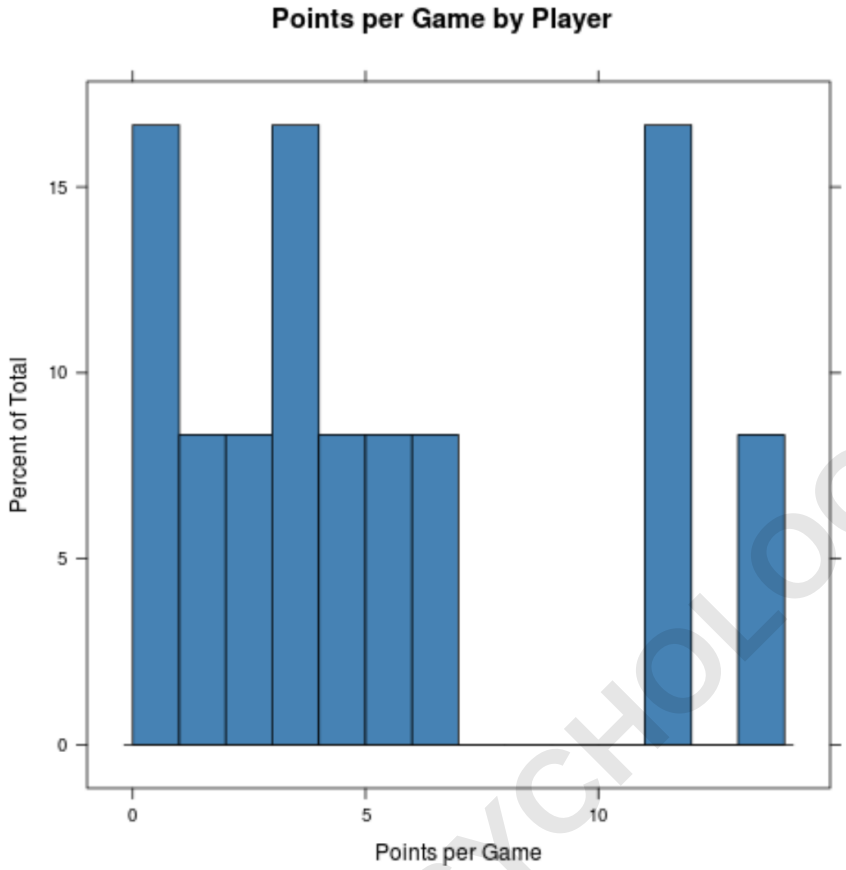
By refining these elements, the **histogram** transitions from a simple **diagnostic plot** to a polished **data asset**. It is also possible to add a **legend** or adjust the **font size** of the labels through the **scales** argument in **lattice**. These small adjustments contribute significantly to the **readability** and **professionalism** of your work, making the **relative frequency** insights more accessible to stakeholders who may not be familiar with **R** or **advanced statistics**.

The Impact of Bin Selection on Data Interpretation

The choice of **bin size**, or the number of **bins**, is perhaps the most critical decision when creating a **histogram**. In **R**, this is controlled by the **breaks** argument. If the bins are too wide, the histogram may "oversmooth" the data, hiding important **modes** or **clusters**. Conversely, if the bins are too narrow, the **relative frequency histogram** may become too "noisy," making it difficult to discern the overall **shape of the distribution**. This phenomenon is often referred to as the **bias-variance tradeoff** in the context of **density estimation**.

#modify the number of bins
histogram(data,
main='Points per Game by Player',
xlab='Points per Game',

col='steelblue',
breaks=15)

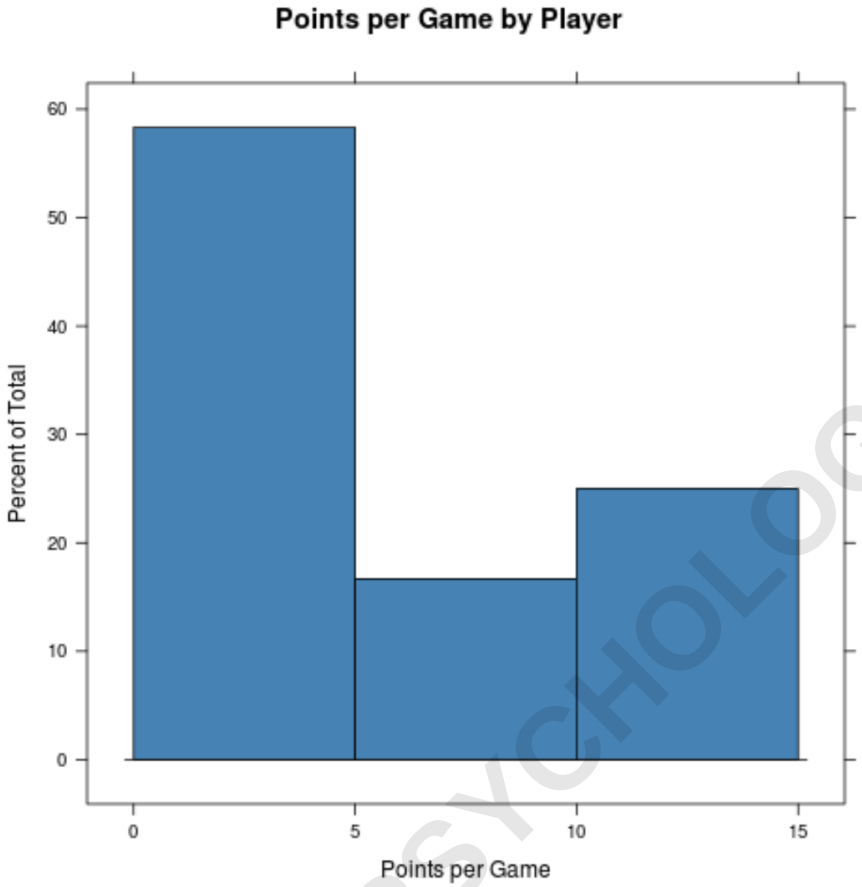


When the **breaks** are set to a higher value, such as 15, the **histogram** provides a **granular** view of the **data**. This can reveal **gaps** or **outliers** that were previously hidden. In **statistical analysis**, this level of detail is vital for identifying non-normal **distributions** or **bimodal** patterns. However, one must be cautious not to over-interpret small fluctuations in **relative frequency** that may simply be the result of **sampling error** rather than a true underlying **population** characteristic.

On the other hand, reducing the number of **bins** can help in identifying the **broad trends** within a dataset. By aggregating the data into fewer categories, the **relative frequency** of each bin increases, creating a more stable and simplified **visualization**. This is often useful for **executive summaries** where the goal is to provide a quick "at-a-glance" understanding of the **data distribution**. Finding the "Goldilocks zone"--the **optimal number of bins**--is a skill developed through experience and the application of **statistical rules of thumb**.

#modify the number of bins
histogram(data,
main='Points per Game by Player',

xlab='Points per Game',
col='steelblue',
breaks=3)



Interpreting Relative Frequency in the Context of Probability

A **relative frequency histogram** is more than just a **bar chart**; it is a visual approximation of a **probability distribution**. When the **y-axis** represents **density** or **proportion**, the total area of the bars equals 100% (or 1.0). This property allows **statisticians** to estimate the **probability** of a value falling within a certain range by calculating the area of the corresponding bars. This is the foundational logic behind the **Normal Distribution** and other **continuous distributions** used in **hypothesis testing**.

When comparing two different datasets--for instance, the "Points per Game" of two different basketball seasons--a **relative frequency histogram** is essential if the number of games played in each season differs. If one season had 82 games and another had only 50, a standard **frequency histogram** would make the 82-game season appear much "larger" on the graph. By using **relative frequency**, we normalize the data, allowing us to see if the **scoring pattern** (the **distribution**) has

actually changed, regardless of the sample size.

Ultimately, the goal of using **R** to create these **visualizations** is to gain a deeper understanding of **variance** and **probability**. By mastering the **lattice** package and the **histogram()** function, you equip yourself with the ability to communicate **quantitative** insights clearly. Whether you are adjusting **bins**, changing **colors**, or interpreting **densities**, the **relative frequency histogram** remains a cornerstone of **modern data science** and **statistical reporting**.

Use Sturges' Rule to identify the optimal number of bins to use in a histogram when you are unsure where to start. This mathematical formula provides a solid baseline for **binning** based on the size of your **sample**, ensuring that your **relative frequency histogram** is neither too cluttered nor too simplistic.