Create a Histogram of Two Variables in R

Authored by

Mohammed looti

November 5, 2025

Introduction to Comparative Data Visualization in R

A histogram is an essential instrument in the statistical toolbox, serving as a powerful graphical representation that illustrates the underlying distribution of numerical data. By generating a histogram, analysts gain immediate insight into key distributional characteristics, including the central tendency, spread (variance), asymmetry (skewness), and the presence of multiple peaks (modality). Mastering these foundational steps is critical for conducting any rigorous statistical investigation.

While visualizing the characteristics of a single dataset is straightforward, the need often arises to perform comparative visualization--that is, displaying two or more distributions simultaneously. Researchers frequently compare distinct samples, such as control versus treatment groups, or demographic subsets, to visually assess whether their underlying characteristics differ significantly before proceeding to formal hypothesis testing. Effective comparison requires plotting these distributions on a single, unified coordinate system.

The statistical programming language R is renowned for its powerful base graphics package, which provides highly flexible functions for creating customized visualizations. This comprehensive guide details the specific techniques required to effectively overlay two histograms using R's native plotting capabilities, enabling a direct and insightful visual comparison of their respective data distributions.

Foundation: Plotting a Single Histogram with Base R

The core function for generating a histogram in R is the **`hist()`** command. When this function is applied to a single numeric vector, it automatically calculates optimal bin widths and frequencies, producing a standard graphical output designed for immediate analysis. This initial plot establishes the canvas upon which subsequent data layers will be added.

For instance, if a dataset is stored in a variable named `variable1`, the most basic plotting command is simply **`hist(variable1)`**. However, to prepare for a successful comparative plot, it is best practice to define certain aesthetic and structural parameters during this initial call. Specifying a color using the **`col`** argument is standard, but more importantly, the first **`hist()`** call must define the overall scale and limits of the graph.

The parameters set in this initial **`hist()`** function--such as the X-axis range (**`xlim`**), axis labels (**`xlab`**, **`ylab`**), and the main plot title (**`main`**)--will govern the entire appearance of the resulting combined visualization. This ensures that both variables are plotted against the exact same scale, a prerequisite for accurate visual comparison. The structure below conceptually outlines the steps for plotting the first variable and then superimposing the second.

hist(variable1, col='red')
hist(variable2, col='blue', add=TRUE)

Overlaying Distributions: The Critical Role of add=TRUE

The ability to plot multiple visualizations on the same coordinate system in R hinges entirely on the use of the **`add=TRUE`** parameter. This argument must be included within the second (and any subsequent) **`hist()`** function call. Its purpose is to override the default behavior of the plotting function, which is to open a new graphical device, and instead instructs R to superimpose the new plot directly onto the graphic currently displayed.

When setting up a comparative visualization, the first distribution is plotted conventionally, effectively setting the stage. The critical second step involves plotting the second distribution using **`add=TRUE`**. If this parameter is omitted, R will simply create a completely separate plot window, thus defeating the purpose of a direct visual comparison.

While the mechanism of overlaying the plots is straightforward using **`add=TRUE`**, the choice of visualization style is paramount. Simple color differentiation, such as red versus blue, works adequately only if the two distributions are completely separate. However, as we will demonstrate, this basic technique fails drastically when the datasets overlap significantly, as the solid bars of the second plot obscure the underlying frequency counts of the first.

Case Study 1: The Pitfalls of Simple Solid Colors

To effectively illustrate the concepts discussed, we must first generate reproducible, synthetic data. We use the **`rnorm()`** function, which is used to draw random samples from a standard normal distribution. Crucially, we employ **`set.seed(1)`** at the start of the script. This ensures that the random data generated is identical every time the code is executed, making the example fully reproducible--a cornerstone of reliable data science practice.

We define two variables, `x1` and `x2`, designed to have similar spread but distinct central tendencies, ensuring a clear area of overlap. Specifically, `x1` is centered around a mean of 0.6, while `x2` is centered around a mean of 0.4. Both variables are assigned a standard deviation (`sd`) of 0.1, indicating that their variation is comparable.

The following code executes the plotting using only solid colors (red and blue) for the two overlapping distributions. Observe how the simplicity of this approach leads to a major flaw in data representation.

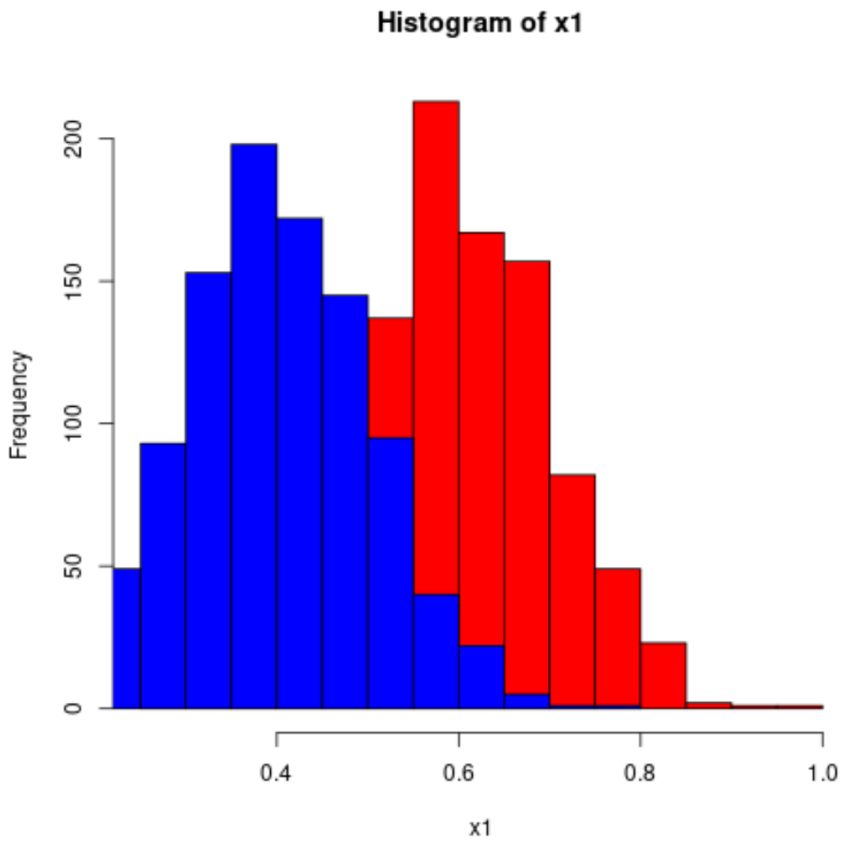
#make this example reproducible

set.seed(1)

#define data
x1 = rnorm(1000, mean=0.6, sd=0.1)
x2 = rnorm(1000, mean=0.4, sd=0.1)

#plot two histograms in same graph
hist(x1, col='red')
hist(x2, col='blue', add=TRUE)

As clearly visible in the resulting image, the use of solid, opaque colors severely compromises the interpretability of the histogram. Wherever the two distributions overlap, the blue bars (representing `x2`) completely mask the red bars (representing `x1`) underneath. This occlusion makes it impossible for the viewer to accurately determine the true frequency count for the first variable in the central, overlapping region of the plot. This limitation necessitates a more sophisticated approach to color handling.



Advanced Clarity: Leveraging Transparency via rgb()

Given that overlapping values render solid colors ineffective, the immediate solution for comparative visualization involves introducing transparency. This technique, managed through the alpha channel, allows the underlying data frequencies to remain visible even when bars are stacked or overlaid. This is a critical best practice in generating clear and unambiguous statistical graphics.

In R, the color specification becomes more flexible through the `rgb()` function. This function allows colors to be defined using the Red, Green, and Blue (RGB) color model, but crucially, it accepts a fourth parameter: the alpha value (A). The alpha value controls the opacity of the color.

The alpha parameter ranges from 0 (full transparency, completely invisible) to 1 (full opacity, solid color). For effective overlapping histograms, an alpha value typically between 0.2 and 0.4 is recommended. This range provides enough transparency for both overlapping bars to be clearly perceived, while still maintaining distinct color identity. Furthermore, to ensure a professional result, we must explicitly define the X-axis limits (`xlim`) in the first `hist()` call to encompass the entire range of both variables, providing a clean and consistent boundary for interpretation.

Case Study 2: Achieving Clarity with Alpha Channels

We now revisit the previous synthetic data (`x1` and `x2`) but implement the advanced coloring technique using the alpha channel. For the blue distribution (`x1`), we define the color as `rgb(0,0,1,0.2)`, signifying pure blue (0,0,1) with an alpha level of 0.2. Similarly, for the red distribution (x2), we use `rgb(1,0,0,0.2)`.

Note the inclusion of comprehensive plot parameters in the first function call, specifically `xlim=c(0, 1)`, which guarantees that the plotting window spans the entire possible range of both variables. This attention to detail in plot setup is essential for creating high-quality, unambiguous graphics.

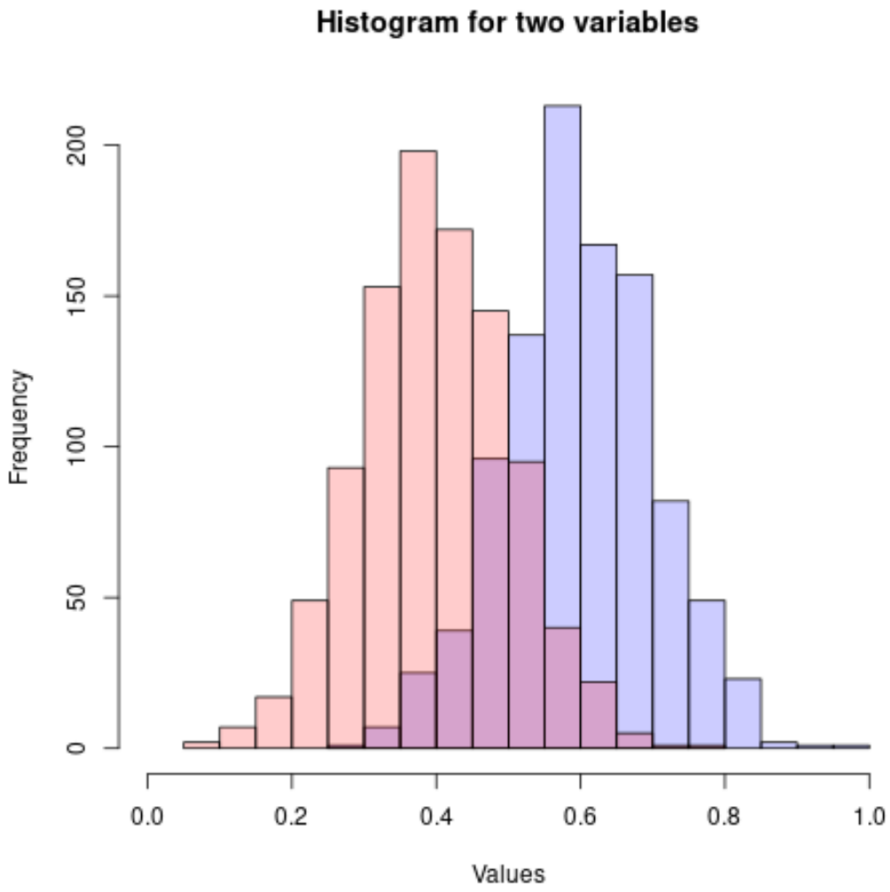
```
#make this example reproducible
set.seed(1)

#define data
x1 = rnorm(1000, mean=0.6, sd=0.1)
x2 = rnorm(1000, mean=0.4, sd=0.1)

#plot two histograms in same graph
hist(x1, col=rgb(0,0,1,0.2), xlim=c(0, 1),
xlab='Values', ylab='Frequency', main='Histogram for two variables')
```

hist(x2, col=rgb(1,0,0,0.2), add=TRUE)

The introduction of transparency fundamentally transforms the visualization. In the region where the distributions overlap, the colors blend, resulting in a darker or denser appearance, which clearly communicates the combined frequency count. Crucially, both the red and blue components remain distinctly visible, allowing for simultaneous visual assessment of the individual frequency counts for both `x1` and `x2` within the shared bins. This allows for immediate and accurate visual interpretation of the degree of overlap and separation between the two variables.



Final Polish: Interpreting Results and Adding a Legend

While the use of transparent colors significantly enhances the visual separation of data, a finalized, professional statistical graphic must be fully self-explanatory. This requires the inclusion of a clear key, or legend, which maps the colors used back to the corresponding data variables. This is accomplished using R's **`legend()`** function.

The **`legend()`** function requires several critical arguments to be successful: first, the intended placement on the graph (e.g., **`'topright'`**, **`'bottomleft'`**); second, the vector of text labels

corresponding to each distribution; and third, the vector of fill colors that must exactly match the colors, including the alpha channel, used in the preceding `hist()` calls. Consistency between the `hist()` and `legend()` color specifications is mandatory.

By integrating a legend, the resulting visualization achieves maximum clarity, enabling any viewer to instantly understand which distribution corresponds to which variable. This completes the effective comparative analysis workflow in R's base graphics system.

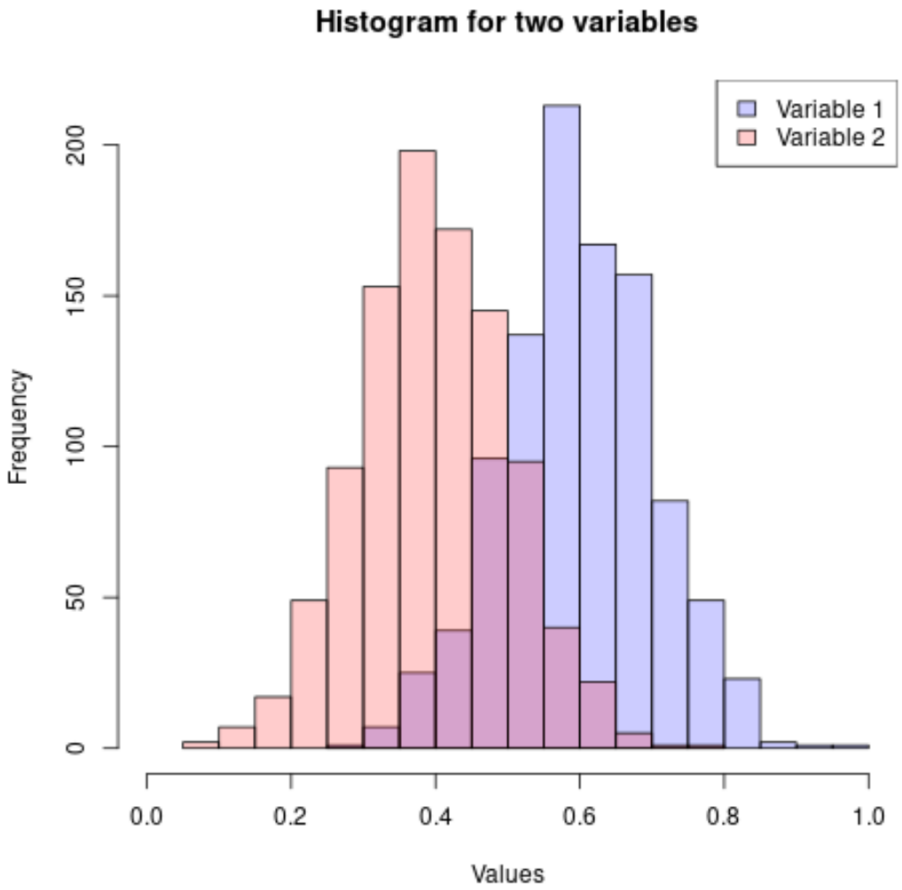
```
#make this example reproducible
set.seed(1)

#define data
x1 = rnorm(1000, mean=0.6, sd=0.1)
x2 = rnorm(1000, mean=0.4, sd=0.1)

#plot two histograms in same graph
hist(x1, col=rgb(0,0,1,0.2), xlim=c(0, 1),
xlab='Values', ylab='Frequency', main='Histogram for two variables')
hist(x2, col=rgb(1,0,0,0.2), add=TRUE)

#add legend
legend('topright', c('Variable 1', 'Variable 2'),
fill=c(rgb(0,0,1,0.2), rgb(1,0,0,0.2)))
```

The final graphic represents a clear, professional, and readily interpretable comparative histogram. This method, leveraging R's base graphics functions and the strategic implementation of alpha transparency, demonstrates a powerful technique for visualizing multiple datasets effectively in statistical analysis.



For further exploration of advanced data visualization techniques, conditional plotting, and other sophisticated statistical analysis tools within the R environment, please consult additional tutorials focusing on graphical parameter control and statistical modeling.