Learning Guide: Plotting Multiple Histograms for Distribution Comparison in R

Authored by

Mohammed looti

November 3, 2025

The Value of Comparative Distribution Analysis

Histograms serve as fundamental instruments in the R programming language, providing essential visual insights into the underlying probability distribution of a dataset. While a single histogram reveals the central tendency and spread of one variable, the true power of sophisticated statistical investigation often lies in comparative analysis. Plotting multiple histograms on the same graphical plane allows data analysts to simultaneously assess, contrast, and interpret differences across distributions regarding their mean, variability, and overall shape.

This comprehensive guide dissects the two primary methodologies available in R for constructing these overlapping visualizations. The first method employs the core functionality found in Base R, offering a rapid, procedural approach ideal for quick exploratory checks. The second, more advanced method utilizes the powerful, declarative framework of the ggplot2 package, which excels in producing highly customized, publication-quality graphics, though it requires preliminary data restructuring.

A critical consideration when superimposing distributions is maintaining visual integrity and clarity. Since the bars representing distinct groups may occupy identical bin ranges, a direct overlay often results in one distribution completely obscuring another. Therefore, mastering specialized techniques--such as adjusting transparency (known as alpha blending) or setting specific positional arguments--is fundamental for effective comparative data visualization and ensuring that all data series are clearly interpretable.

Fundamental Syntax for Creating Overlapping Visualizations

Irrespective of whether the analyst employs Base R or the ggplot2 ecosystem, the core conceptual principle remains consistent: establishing a primary plot canvas and then sequentially adding subsequent distribution plots onto that existing space. The technical execution of this layering, however, differs significantly between the two systems due to their distinct architectural philosophies.

When utilizing the procedural environment of **Base R**, the process hinges on executing the native `hist()` function multiple times. The crucial parameter that enables layering is `add=TRUE`. This Boolean argument instructs R to draw the new histogram elements without clearing the previously rendered graphical device, thus achieving the desired overlay effect:

hist(data1, col='red')
hist(data2, col='blue', add=TRUE)

Conversely, working within the structured framework of ggplot2 requires restructuring the data into

a long format, where a dedicated grouping variable identifies the separate distributions. The visualization is then constructed using the `geom_histogram()` layer, mapping the grouping variable to the `fill` aesthetic, which automatically handles the partitioning:

ggplot(df, aes(x = x_var, fill = grouping_var)) +
geom_histogram(position = 'identity', alpha = 0.4)

The subsequent detailed examples will demonstrate the implementation and necessary refinements for both of these powerful methodologies, ensuring that the resulting overlapping histograms are both statistically accurate and aesthetically effective for communication.

Method 1: Overlapping Histograms using Base R (Procedural Approach)

The Base R plotting system is frequently chosen for tasks requiring minimal setup and maximum speed, as it relies on core dependencies already present in the R installation. To successfully plot multiple distributions using this system, it is imperative that the initial histogram call properly establishes a plotting window large enough to accommodate the full data ranges of all subsequent distributions. If the initial x-axis limit (specified by the `xlim` parameter) is too conservative, the second histogram's data points lying outside those boundaries will be truncated, leading to an incomplete and misleading visualization.

In the practical demonstration provided below, we generate two synthetic datasets, `x1` and `x2`, utilizing the `rnorm` function to simulate data that follows a normal distribution but possess distinct means and standard deviations. The `set.seed()` function ensures that these random data generation steps are fully reproducible, a best practice in statistical coding.

The plotting sequence initiates with `hist(x1)`, which defines the canvas dimensions and axis labels. We then immediately call `hist(x2, add=TRUE)` to superimpose the second distribution. A key manual step unique to Base R is the required addition of a legend; unlike specialized packages, Base R does not automatically generate a descriptive legend for overlapping plots, necessitating the use of the `legend()` function to differentiate the colored series clearly.

#make this example reproducible
set.seed(1)

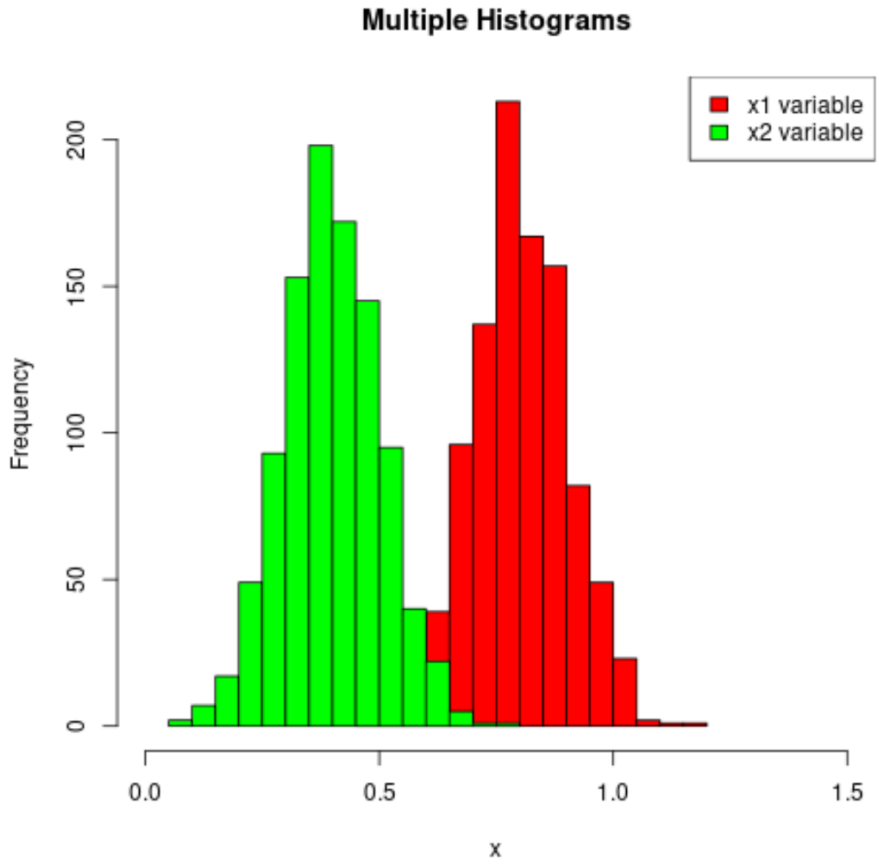
#define data
x1 = rnorm(1000, mean=0.8, sd=0.2)
x2 = rnorm(1000, mean=0.4, sd=0.1)

#plot two histograms in same graph
hist(x1, col='red', xlim=c(0, 1.5), main='Multiple Histograms', xlab='x')

```
hist(x2, col='green', add=TRUE)

#add legend
legend('topright', c('x1 variable', 'x2 variable'), fill=c('red', 'green'))
```

This resulting visualization effectively contrasts the two simulated distributions, visually confirming that the variable `x1` is centered around a notably higher mean (0.8) compared to the variable `x2` (0.4).



Deciphering Essential Base R Plotting Parameters

Successful implementation of overlapping histograms in **Base R** relies heavily on the meticulous control of the initial plotting environment setup. When the first histogram is generated via `hist(x1, ...)`, parameters such as `xlim`, `main`, and `xlab` are not merely aesthetic elements; they define the fundamental boundaries and textual context for the entire graphic. It is the responsibility of the analyst to manually ensure that the `xlim` range is sufficiently broad (in this case, extending up to 1.5) to encompass the full range of all subsequent data series, thereby preventing any unintended data truncation.

The core mechanical feature enabling the layering effect is the `add=TRUE` argument, which must be explicitly included in every subsequent `hist()` call after the first. This Boolean instruction compels the function to superimpose the new graphical elements onto the currently active graphics device, rather than initiating a completely new plot which would overwrite the previous one. Neglecting this crucial step is the most common error when attempting Base R overlays.

Furthermore, to ensure the comparative visualization is meaningful and accessible, a descriptive legend is indispensable. The `legend()` function requires careful configuration, including specifying its precise location (e.g., `'topright'`), defining the textual labels for each series, and associating the corresponding fill colors (e.g., `fill=c('red', 'green')`). This requirement for manual setup underscores the procedural and low-level control inherent in Base R graphics, contrasting sharply with the more automatic, aesthetic-driven approach found in packages like ggplot2.

Method 2: Advanced Visualization with ggplot2 (Grammar of Graphics)

While Base R provides speed and simplicity, the ggplot2 package offers unparalleled control over aesthetics and requires data to be rigorously structured according to the principles of the Grammar of Graphics. For plotting multiple distributions in ggplot2, the data must first undergo a transformation into the "long" format. This standardized format ensures that all numerical observations are consolidated into a single value column, while a separate, categorical variable explicitly identifies the group or distribution to which each observation belongs.

The following example illustrates this essential data transformation process. We begin by creating two separate variables (x1 and x2) and then combine them into a single data frame, `df`. In this structure, the `value` column aggregates all numerical data points, and the `var` column serves as the critical grouping variable, clearly distinguishing observations originating from x1 or x2.

Once the data frame is appropriately prepared, the ggplot2 plotting syntax becomes highly declarative and intuitive. The continuous variable (`value`) is mapped to the primary x-axis, and the categorical grouping variable (`var`) is mapped to the `fill` aesthetic. The `geom_histogram()` function then intelligently and automatically calculates the appropriate bins for each subgroup, generating a distinct, color-coded histogram for every level defined by the `fill` aesthetic.

library(ggplot2)

#make this example reproducible
set.seed(1)

#create data frame
df <- data.frame(var = c(rep('x1', 1000), rep('x2', 1000)),
value = c(rnorm(1000, mean=0.8, sd=0.1),

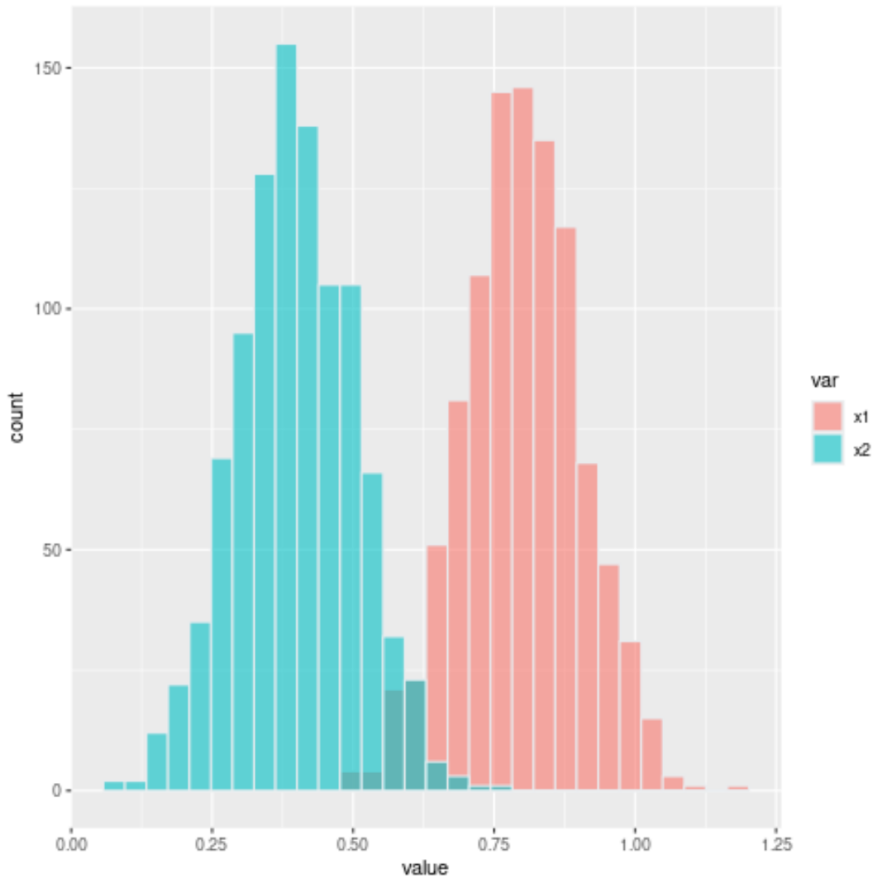
```
rnorm(1000, mean=0.4, sd=0.1)))

#view first six rows of data frame
head(df)

var value
1 x1 0.7373546
2 x1 0.8183643
3 x1 0.7164371
4 x1 0.9595281
5 x1 0.8329508
6 x1 0.7179532

#plot multiple histograms
ggplot(df, aes(x=value, fill=var)) +
geom_histogram( color='#e9ecef', alpha=0.6, position='identity')
```

A significant advantage of this ggplot2 method over Base R is the automatic generation of a precise legend based on the `fill` aesthetic mapping, which dramatically simplifies the final steps of the visualization process.



Managing Transparency and Positioning in ggplot2

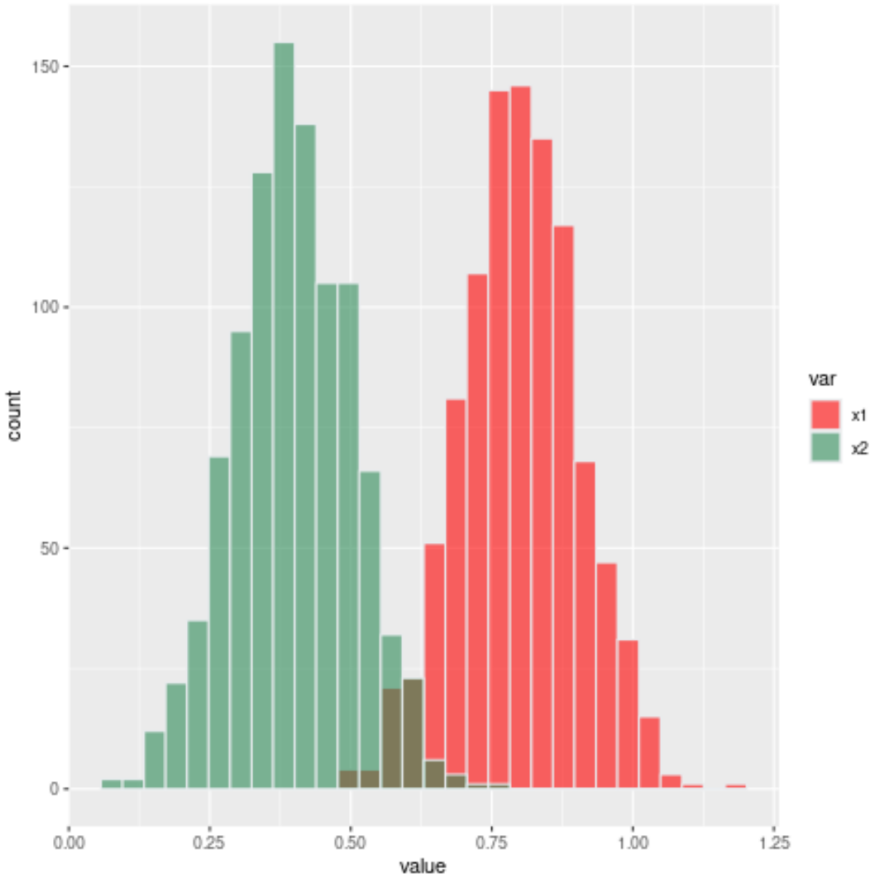
To ensure an effective and informative visual overlap within ggplot2, specialized parameters must be carefully applied within the `geom_histogram()` function to control the interaction between the histogram bars. The two most critical parameters for this purpose are `alpha` and `position`.

The `alpha` parameter governs the opacity or transparency of the graphical elements, with values ranging from 0 (fully transparent) to 1 (fully opaque). By setting `alpha=0.6`, as demonstrated in the code, the resulting color blending in areas of overlap clearly reveals the underlying density contributions from both groups. This technique significantly enhances visual interpretation; without adequate transparency, the distribution plotted last would completely conceal the frequencies of the first distribution in shared bins.

Equally important is the `position='identity'` argument. By default, `geom_histogram()` uses `position='stack'`, which awkwardly piles bars representing different groups on top of one another. Utilizing `'identity'` forces the bars to occupy the exact same space along the y-axis, which is essential for facilitating the direct visual comparison of frequencies across the shared bin structure.

For advanced aesthetic customization, particularly when preparing visualizations for formal reporting or publication, the default color palettes generated by ggplot2 may be inadequate. To override these standardized settings and enforce specific corporate or design requirements, the `scale_fill_manual()` function offers precise control over the color assigned to each distinct level of the grouping variable, guaranteeing consistency and adherence to strict visual guidelines.

Analysts can quickly and effectively change the default colors of the histograms by incorporating the **scale_fill_manual()** function into their ggplot pipeline:



Concluding Thoughts and Further R Resources

Mastering the technique of plotting overlapping histograms is a foundational milestone in comparative statistical graphics. Both the procedural Base R method and the declarative ggplot2 approach provide robust mechanisms for visual comparison, with the choice between them largely dependent on the project's scope, the required level of customization, and the complexity of the initial data structure.

To further develop expertise in advanced R visualization, analysts are encouraged to explore related graphical techniques. These include density plots, which offer a smooth, continuous

alternative to the stepped appearance of histograms, and box plots, which efficiently summarize the five-number summary (median, quartiles, and range) of a distribution for group comparisons.

The following tutorials explain how to create other common charts in R:

How to create density plots in R.
Generating side-by-side box plots for group comparison.
Utilizing faceted plots for visualizing multivariate data.