

# How to Plot a t Distribution in R: A Step-by-Step Guide

Authored by  
**stats writer**

March 10, 2026

## RECOMMENDED CITATION

stats writer (2026). *How to Plot a t Distribution in R: A Step-by-Step Guide*.

PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=135049>

## Comprehensive Overview of Visualizing the t-Distribution in R

In the field of **statistical analysis**, the **Student's t-distribution** serves as a fundamental theoretical framework, particularly when dealing with small sample sizes or unknown population standard deviations. For researchers and data scientists utilizing the **R programming language**, generating a visual representation of this distribution is a critical step in understanding the behavior of data and the implications of **hypothesis testing**. Visualizing a **t-distribution** allows one to observe the probability density across different values, providing insights into the likelihood of observing specific test statistics. This article provides an in-depth guide on how to programmatically construct these plots using **base R** functions, ensuring both clarity and precision in your **data visualization** efforts.

The process of plotting in **R** is highly modular, allowing for significant customization of the graphical output. To create a standard plot of the **probability density function** (PDF) for a t-distribution, we primarily rely on two built-in functions: **dt()** and **curve()**. The former is responsible for calculating the density values based on specific **degrees of freedom**, while the latter handles the rendering of these values onto a coordinate system. By mastering these tools, you can transition from simple default plots to sophisticated, publication-quality graphics that effectively communicate statistical findings.

Understanding the mathematical foundation behind these functions is essential for accurate interpretation. The **t-distribution** is defined by its **degrees of freedom**, which dictate the thickness of the tails of the distribution. As the degrees of freedom increase, the t-distribution begins to converge toward a **normal distribution**. This relationship is easily observable through **statistics**-based plotting, making **R** an invaluable asset for educational and professional analytical tasks. In the following sections, we will explore the syntax, implementation, and enhancement of these plots.

### Core Functions for Statistical Plotting in R

To effectively plot the **probability density function** for a **t-distribution**, it is necessary to understand the specific roles of the functions provided by the **R** statistical environment. The environment provides a suite of functions for various distributions, and for the t-distribution, the **dt()** function is the primary tool for density calculation. This function computes the "height" of the distribution at any given point along the x-axis, which represents the test statistic values.

The **curve()** function is a versatile plotting utility in **base R** that allows for the drawing of functions over a specified interval. Unlike the **plot()** function, which often requires a pre-defined set of x and y coordinates, **curve()** can take a functional expression directly, making it much more efficient for continuous mathematical distributions. When combined, **dt()** and **curve()** provide a seamless workflow for generating theoretical curves without the need for manual data frame construction.

**dt(x, df):** This function calculates the probability density function. The **x** parameter represents the vector of quantiles, and the **df** parameter represents the degrees of freedom.

**curve(function, from = NULL, to = NULL):** This function is used to draw the curve of the specified mathematical function. The **from** and **to** arguments define the range of the x-axis over which the function is evaluated.

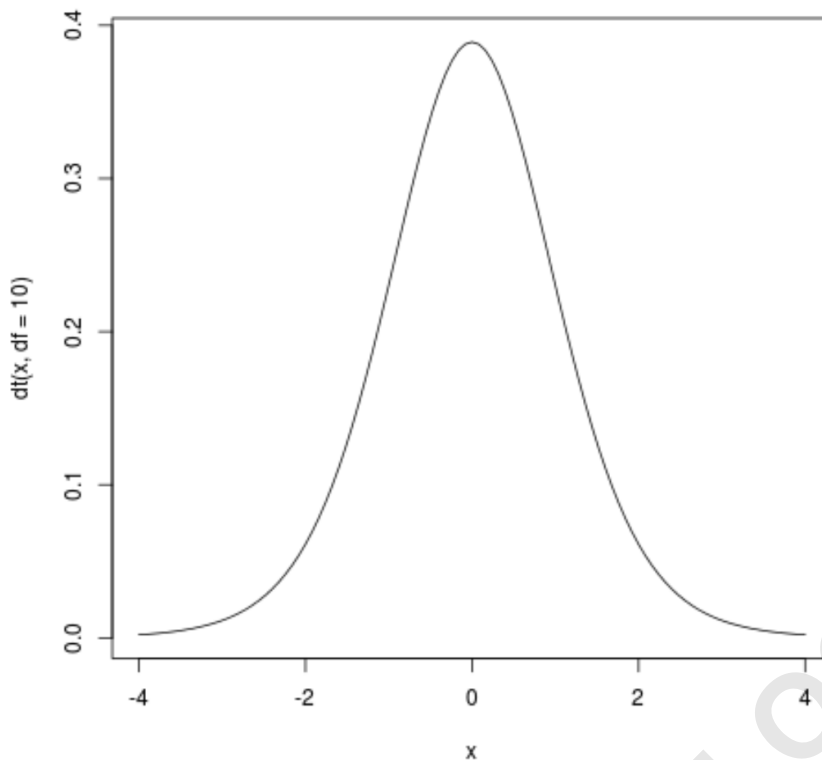
## Implementation of the Basic t-Distribution Plot

To initiate a plot, one must define the degrees of freedom within the **dt()** function. This parameter is crucial because it characterizes the shape of the curve; lower values result in "heavier" tails, indicating a higher probability of extreme values compared to a normal distribution. Furthermore, the **from** and **to** arguments in the **curve()** function must be set to cover a range that captures the bulk of the distribution's area, typically ranging from -4 to 4 for standard statistics visualizations.

Consider a scenario where you wish to visualize a t-distribution with 10 degrees of freedom. By setting the x-axis range from -4 to 4, you ensure that the central peak and the asymptotic decay of the tails are clearly visible. This range is standard for standardized distributions where the mean is zero. The resulting plot provides a clear visual of how the density is concentrated around the center, which is a hallmark of symmetrical distributions in statistics.

The following code block demonstrates the fundamental syntax required to generate this basic visualization. Note how the **dt()** function is nested within the **curve()** function, allowing **R** to evaluate the density dynamically across the specified range:

```
curve(dt(x, df=10), from=-4, to=4)
```



## Theoretical Symmetry and Distribution Shape

A defining characteristic of the **t-distribution** is its symmetry. Much like the standard **normal distribution**, it is centered at a mean of zero. This bell-shaped curve is symmetrical, meaning the area under the curve to the left of the mean is equal to the area to the right. This property is vital when performing two-tailed **hypothesis testing**, as it ensures that **statistics** calculated for one tail can be mirrored for the other.

While the t-distribution appears similar to the **normal distribution**, its shape is more spread out, especially when the **degrees of freedom** are low. This "heaviness" in the tails accounts for the added uncertainty associated with estimating the population standard deviation from a small sample. As the sample size grows, the uncertainty decreases, and the **t-distribution** visual becomes increasingly indistinguishable from the Z-distribution.

Visualizing these properties in **R** helps researchers verify that their data or model assumptions align with theoretical expectations. By observing the peak height (the mode) and the rate at which the curve approaches the x-axis, one can qualitatively assess the impact of sample size on the distribution's **probability density function**.

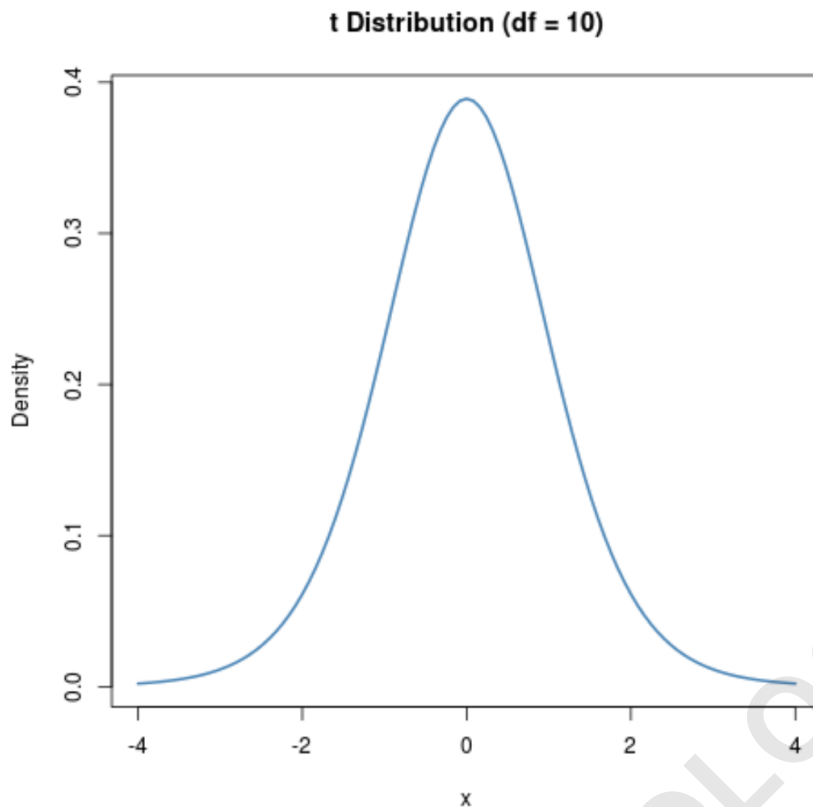
## Customizing Plots for Professional Clarity

A basic plot is often insufficient for formal reports or presentations. **R** provides a wealth of parameters within the **curve()** function to enhance the aesthetic quality and readability of your **data visualization**. Adding titles, modifying axis labels, and changing line properties are essential steps in transforming a raw graphic into a professional figure. These modifications help the audience immediately identify what is being represented without needing to parse the underlying code.

The **main** parameter allows for the addition of a descriptive title, while **ylab** and **xlab** are used to rename the y and x axes, respectively. To make the line itself more prominent, the **lwd** (line width) parameter can be increased. Additionally, the **col** parameter accepts color names or hex codes, allowing you to differentiate the curve from the standard black-and-white output. Such customizations are not merely aesthetic; they are functional improvements that enhance the **statistics** communication process.

In the example below, we apply several of these enhancements to the previous **t-distribution** plot. By changing the color to 'steelblue' and increasing the line thickness, the distribution becomes much more visually distinct:

```
curve(dt(x, df=10), from=-4, to=4,  
main = 't Distribution (df = 10)', #add title  
ylab = 'Density', #change y-axis label  
lwd = 2, #increase line width to 2  
col = 'steelblue') #change line color to steelblue
```



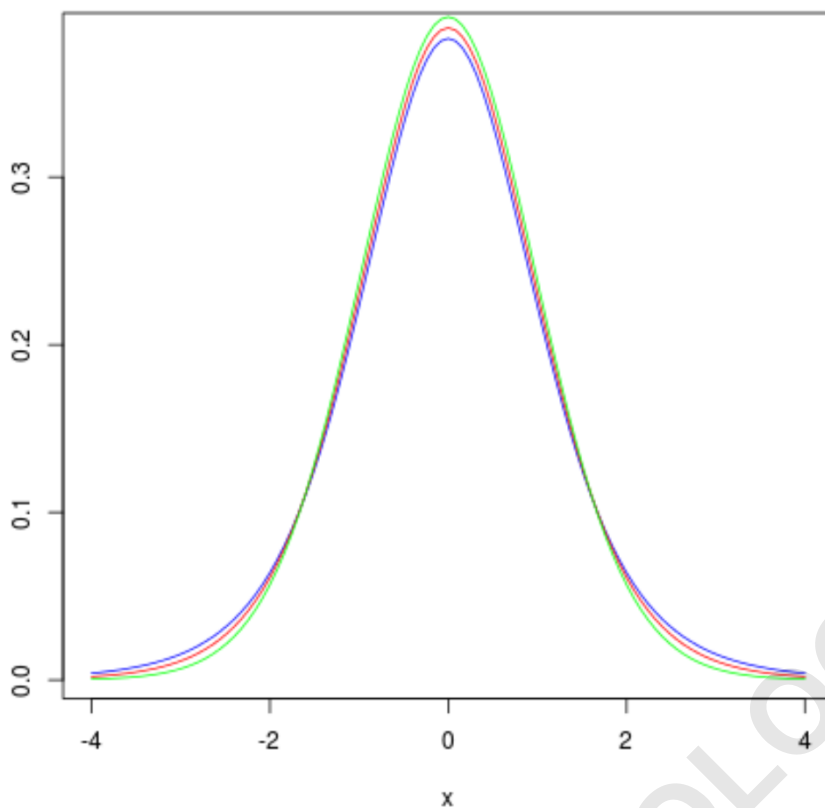
## Comparing Multiple Distributions on a Single Graph

One of the most powerful aspects of **data visualization** in **R** is the ability to overlay multiple curves on a single set of axes. This is particularly useful for demonstrating how the **t-distribution** changes as the **degrees of freedom** vary. By comparing distributions with different parameters, such as  $df = 6$ ,  $df = 10$ , and  $df = 30$ , we can visually confirm that higher degrees of freedom lead to a narrower, taller central peak and thinner tails.

To achieve this in **base R**, the **add=TRUE** argument is utilized in subsequent calls to the **curve()** function. This prevents the previous plot from being overwritten, allowing multiple **probability density function** curves to occupy the same space. This technique is indispensable for educational demonstrations and comparative **statistics**, providing a clear visual proof of convergence theories.

The following R code demonstrates how to layer three different t-distributions. By assigning unique colors to each curve, the viewer can easily distinguish between the various levels of **degrees of freedom**:

```
curve(dt(x, df=6), from=-4, to=4, col='blue')
curve(dt(x, df=10), from=-4, to=4, col='red', add=TRUE)
curve(dt(x, df=30), from=-4, to=4, col='green', add=TRUE)
```



## Constructing an Informative Legend

When multiple curves are present in a single plot, a **legend** becomes an essential component for clarity. Without a legend, the reader cannot identify which color corresponds to which set of **degrees of freedom**. The **legend()** function in **R** provides a highly customizable way to add this information to your graphics. It allows you to specify the position, text, and styling of the legend box to ensure it does not obstruct the data.

The **legend()** function uses a variety of arguments to define its appearance. These include coordinates for placement (or keywords like "topright"), the labels for the data, and the visual markers (like lines or colors) that match the plot. Proper use of the **cex** (character expansion) argument ensures the text size is appropriate for the overall dimensions of the plot, while **lty** (line type) ensures the legend markers match the style of the curves.

The syntax for the **legend()** function is structured as follows:

**legend(x, y=NULL, legend, fill, col, bg, lty, cex)**

The parameters are defined as:

**x, y:** These represent the coordinates used to position the legend on the plot area.

**legend:** A character vector containing the text labels to be displayed.

**fill:** This specifies the colors used to fill boxes in the legend, if applicable.

**col:** The vector of colors used for the lines or points within the legend.

**bg:** The background color of the legend box itself.

**lty:** The line style, which should correspond to the **data visualization** style used in the curves.

**cex:** A scaling factor for the text size within the legend.

## Final Integration of Comparative Plots and Legends

In the final step of creating a professional statistical graphic, we combine the curve generation and the legend placement into a single cohesive script. This approach ensures that all elements of the **t-distribution** comparison are perfectly aligned. By plotting the distributions with 6, 10, and 30 **degrees of freedom** and then immediately calling the **legend()** function, we create a self-contained visualization that is ready for analysis.

This comprehensive method of plotting is not limited to the **t-distribution** but can be applied to any **probability density function** in **R**. Mastering the layering of curves and the addition of legends is a core skill for any statistician. It allows for the clear communication of complex mathematical concepts, such as the convergence of distributions and the impact of sample size on statistical certainty.

The complete code for a high-quality, comparative t-distribution plot is provided below. This example illustrates the best practices for **statistics** reporting in **base R**:

**#create density plots**

```
curve(dt(x, df=6), from=-4, to=4, col='blue')
```

```
curve(dt(x, df=10), from=-4, to=4, col='red', add=TRUE)
```

```
curve(dt(x, df=30), from=-4, to=4, col='green', add=TRUE)
```

**#add legend**

```
legend(-4, .3, legend=c("df=6", "df=10", "df=30"),
```

```
col=c("blue", "red", "green"), lty=1, cex=1.2)
```

