

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

Authored by  
**stats writer**

March 2, 2026

## RECOMMENDED CITATION

stats writer (2026). *How to Plot a Normal Distribution in R: A Step-by-Step Guide*.  
PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=133465>

## Fundamentals of Visualizing the Normal Distribution in R

The **normal distribution**, often referred to as the Gaussian distribution, is a cornerstone of **statistical analysis** and data science. In the **R programming language**, visualizing this distribution is a fundamental skill that allows researchers and analysts to understand the spread and central tendency of their data. Whether you are conducting hypothesis testing or performing exploratory data analysis, the ability to generate a clear and accurate bell curve is essential for communicating complex statistical concepts to a broader audience.

R provides a robust suite of built-in functions designed specifically for handling probability distributions. The core function used for this purpose is **dnorm**, which calculates the **probability density function** (PDF) for a given set of values. By combining **dnorm** with plotting functions like **plot**, **hist**, or **curve**, users can transform abstract mathematical parameters into intuitive visual representations. This process involves defining a range of input values and calculating their corresponding densities based on specific parameters such as the **mean** and **standard deviation**.

Beyond simple visualization, mastering these techniques in R enables a deeper level of customization. Users can overlay multiple distributions to compare different populations, highlight specific areas under the curve to represent **p-values** or confidence intervals, and adjust graphical parameters to meet the high standards of academic publishing. This guide will explore several methodologies for plotting the **normal distribution**, ranging from the straightforward syntax of **Base R** to the sophisticated capabilities of the **ggplot2** package.

### Utilizing Base R for Statistical Graphics

Before diving into external libraries, it is crucial to understand the power and efficiency of **Base R** graphics. These functions are included by default in the R environment, meaning they require no additional installations and offer a fast, reliable way to generate plots. The **plot** function is highly versatile, allowing for the creation of everything from simple **scatterplots** to complex line graphs. When plotting a **normal distribution**, Base R requires the user to explicitly define the coordinates, which provides a transparent look at how the data is being constructed.

The process generally begins with the creation of a sequence of numbers using the **seq** function. This sequence serves as the x-axis, representing the range of values over which the distribution will be plotted. Following this, the **dnorm** function is applied to the sequence to generate the corresponding y-axis values. By specifying a **mean** of 0 and a **standard deviation** of 1, one can easily visualize the **standard normal distribution**, which is the baseline for many statistical transformations.

Customization in Base R is achieved through various arguments within the **plot** and **axis** functions. For instance, the **type = "l"** argument ensures that the points are connected by a smooth line

rather than displayed as individual dots. Furthermore, users can suppress the default axes to create custom labels that highlight standard deviation units, making the plot more informative for educational purposes. This level of control is one of the primary reasons why many statisticians still prefer Base R for quick and precise visualizations of mathematical functions.

## Example 1: Plotting a Standard Normal Distribution

To create a **normal distribution** plot where the **mean** is zero and the **standard deviation** is one, we utilize a programmatic approach to define our data points. This specific configuration is known as the **standard normal distribution**. By generating a dense sequence of numbers between -4 and 4, we ensure that the tails of the distribution are adequately represented, covering more than 99% of the area under the curve.

The following code snippet demonstrates the step-by-step process of defining the x-values, calculating the density with **dnorm**, and rendering the final image. Note how the **axis** function is used to replace numerical ticks with descriptive labels like "-1s" or "mean," which provides immediate context to the viewer regarding the properties of the curve.

```
#Create a sequence of 100 equally spaced numbers between -4 and 4
```

```
x <- seq(-4, 4, length=100)
```

```
#create a vector of values that shows the height of the probability distribution#for each value in x
```

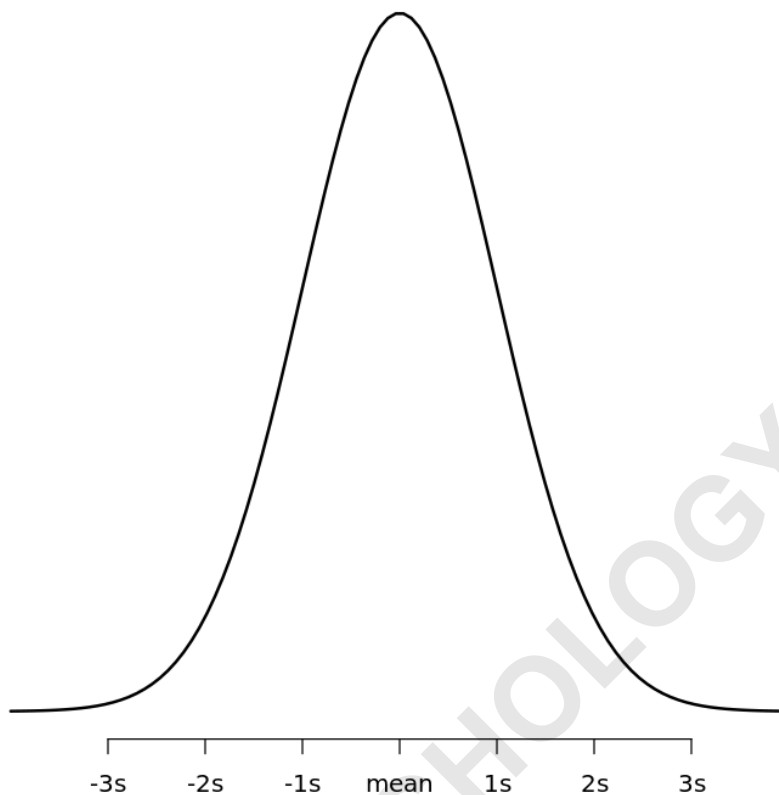
```
y <- dnorm(x)
```

```
#plot x and y as a scatterplot with connected lines (type = "l") and add#an x-axis with custom labels
```

```
plot(x,y, type = "l", lwd = 2, axes = FALSE, xlab = "", ylab = "")
```

```
axis(1, at = -3:3, labels = c("-3s", "-2s", "-1s", "mean", "1s", "2s", "3s"))
```

The resulting **data visualization** provides a clean, professional representation of the **normal distribution**. This method is particularly useful for textbooks or reports where clarity and simplicity are prioritized. By manually defining the axes, the analyst has full control over the aesthetic output, ensuring that the focus remains on the mathematical symmetry of the Gaussian curve.



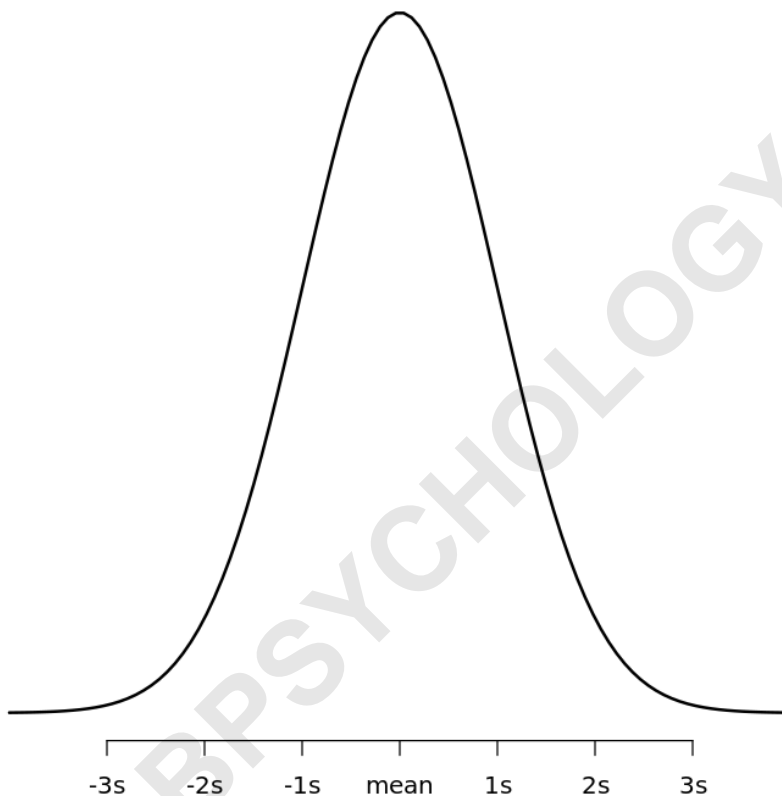
## Example 2: Streamlining Code with the Curve Function

For those looking for a more concise way to achieve the same result, R offers the **curve** function. This function is specifically designed to plot mathematical expressions over a specified interval, significantly reducing the amount of code required. Instead of manually creating vectors for x and y, the **curve** function handles the generation of coordinates internally, making the script cleaner and less prone to manual errors in sequence definition.

When using **curve**, you simply pass the name of the function, such as **dnorm**, and the limits of the x-axis. This approach is highly efficient for rapid prototyping or when you need to visualize the distribution without the overhead of data frame management. It maintains the same level of flexibility as the **plot** function, allowing for the adjustment of line width, colors, and custom axes to match the desired output style.

```
curve(dnorm, -3.5, 3.5, lwd=2, axes = FALSE, xlab = "", ylab = "")  
axis(1, at = -3:3, labels = c("-3s", "-2s", "-1s", "mean", "1s", "2s", "3s"))
```

As seen below, this method produces an identical plot to the first example but with much greater brevity. This exemplifies the philosophy of the **R project**: providing multiple paths to the same analytical goal, whether you prefer explicit data manipulation or high-level functional programming. The **curve** function is an excellent tool for any **data analyst** who values code readability and speed.



### Example 3: Customizing Mean and Standard Deviation

In many real-world scenarios, data does not follow the **standard normal distribution**. Instead, it has a specific **mean** and **standard deviation** that reflect the population being studied. Visualizing these custom distributions requires adjusting the arguments within the **dnorm** function to reflect the desired population parameters. This is critical for comparing different datasets, such as the heights of people in different countries or the test scores of students in different districts.

To visualize a custom distribution, we define variables for the **mean** and **standard deviation**. This allows for dynamic code that can be easily updated if the parameters change. By calculating the x-

axis range based on these parameters (usually mean +/- 4 standard deviations), we ensure that the entire relevant part of the curve is visible. This practice prevents the common mistake of "clipping" the distribution, where the curve is cut off prematurely on the plot edges.

### **#define population mean and standard deviation**

```
population_mean <- 50
```

```
population_sd <- 5
```

```
#define upper and lower bound
```

```
lower_bound <- population_mean - population_sd
```

```
upper_bound <- population_mean + population_sd
```

```
#Create a sequence of 1000 x values based on population mean and standard deviation
```

```
x <- seq(-4, 4, length = 1000) * population_sd + population_mean
```

```
#create a vector of values that shows the height of the probability distribution
```

```
#for each value in x
```

```
y <- dnorm(x, population_mean, population_sd)
```

```
#plot normal distribution with customized x-axis labels
```

```
plot(x,y, type = "l", lwd = 2, axes = FALSE, xlab = "", ylab = "")
```

```
sd_axis_bounds = 5
```

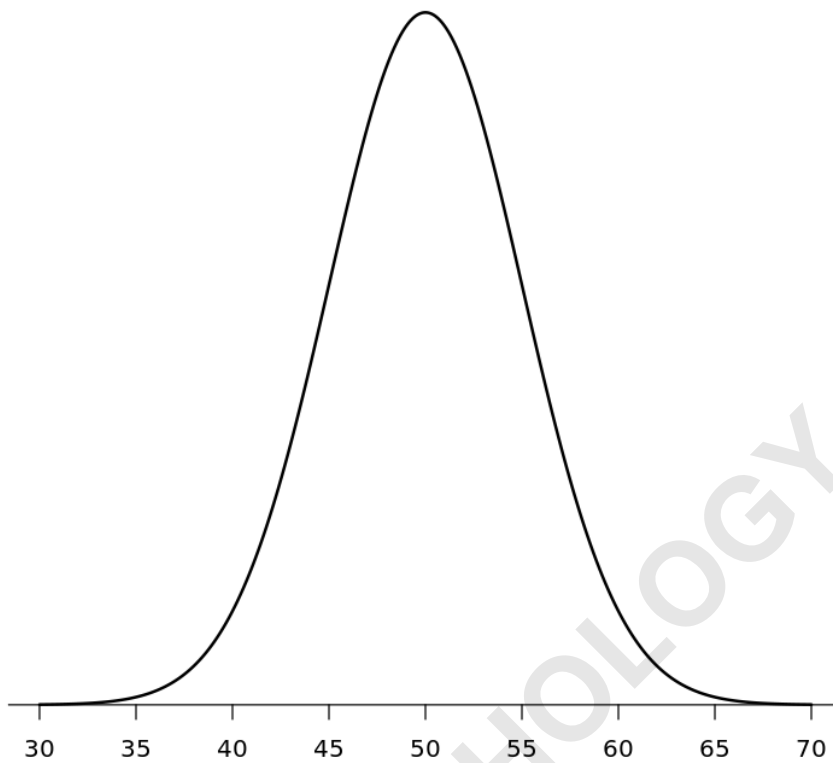
```
axis_bounds <- seq(-sd_axis_bounds * population_sd + population_mean,
```

```
sd_axis_bounds * population_sd + population_mean,
```

```
by = population_sd)
```

```
axis(side = 1, at = axis_bounds, pos = 0)
```

This customized **normal distribution** plot is an essential tool for **inferential statistics**. It allows the analyst to see exactly where the data lies in relation to the center and the spread. Using programmatic labels on the x-axis ensures that the visualization is accurate and scalable, providing a clear window into the characteristics of the specific population being modeled.



## Advanced Visualization with ggplot2

While Base R is powerful, **ggplot2** has become the industry standard for **data visualization** in R. Developed by Hadley Wickham, **ggplot2** is based on the **Grammar of Graphics**, a framework that allows users to build plots by layering components. This approach is highly modular and allows for the creation of aesthetically pleasing and complex graphics with relatively simple syntax. For plotting mathematical functions like the **normal distribution**, **ggplot2** offers a specialized function called **stat\_function**.

The **stat\_function** layer is particularly useful because it computes values on the fly. Instead of providing a full dataset of x and y coordinates, you only need to provide the range for the x-axis and the name of the function you wish to map. **ggplot2** then takes care of the sampling and line smoothing, ensuring a high-quality curve every time. This integration makes it very easy to combine theoretical distributions with actual observed data in a single plot.

Beyond its ease of use, **ggplot2** excels in its ability to handle "themes" and "geoms." You can

easily change the entire look of a plot--from a classic grey background to a clean white "minimalist" style--with a single line of code. This makes it the preferred choice for data scientists who need to produce consistent, brand-aligned graphics for business presentations or academic publications. Learning to plot a **normal distribution** in **ggplot2** is an important step toward mastering the broader **Tidyverse** ecosystem.

## Example 1: Standard Normal Curve in ggplot2

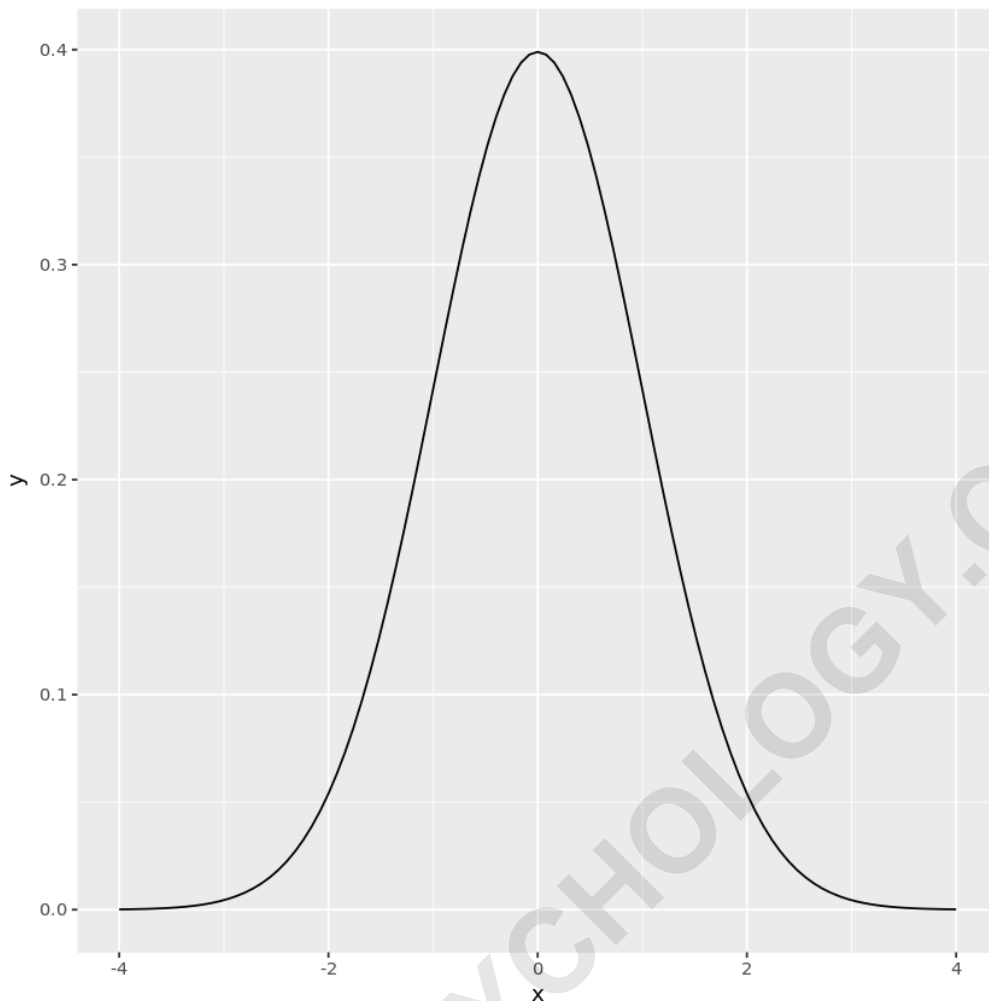
To generate a **standard normal distribution** using **ggplot2**, we first ensure the library is loaded. If it is not present in your environment, it can be installed from **CRAN**. The basic structure involves initializing a ggplot object with a dummy data frame that defines the x-axis limits, and then adding the **stat\_function** layer. By default, **stat\_function** uses the **dnorm** function if not otherwise specified, making the code incredibly clean.

This method is highly efficient because it leverages the declarative nature of **ggplot2**. You describe what you want the plot to look like (e.g., "plot the dnorm function from -4 to 4") rather than detailing every step of the calculation. This abstraction allows for faster iteration and reduces the likelihood of bugs in your **statistical computing** workflow.

```
#install (if not already installed) and load ggplot2  
if(!(require(ggplot2))){install.packages('ggplot2')}
```

```
#generate a normal distribution plot  
ggplot(data.frame(x = c(-4, 4)), aes(x = x)) +  
stat_function(fun = dnorm)
```

The resulting image is a modern, high-resolution curve that is ready for further customization. From here, one could easily add titles, change line colors, or shade areas under the curve to represent specific probabilities. The power of **ggplot2** lies in this extensibility, allowing a simple plot to evolve into a complex analytical dashboard.



## Example 2: Plotting Real Data with mtcars

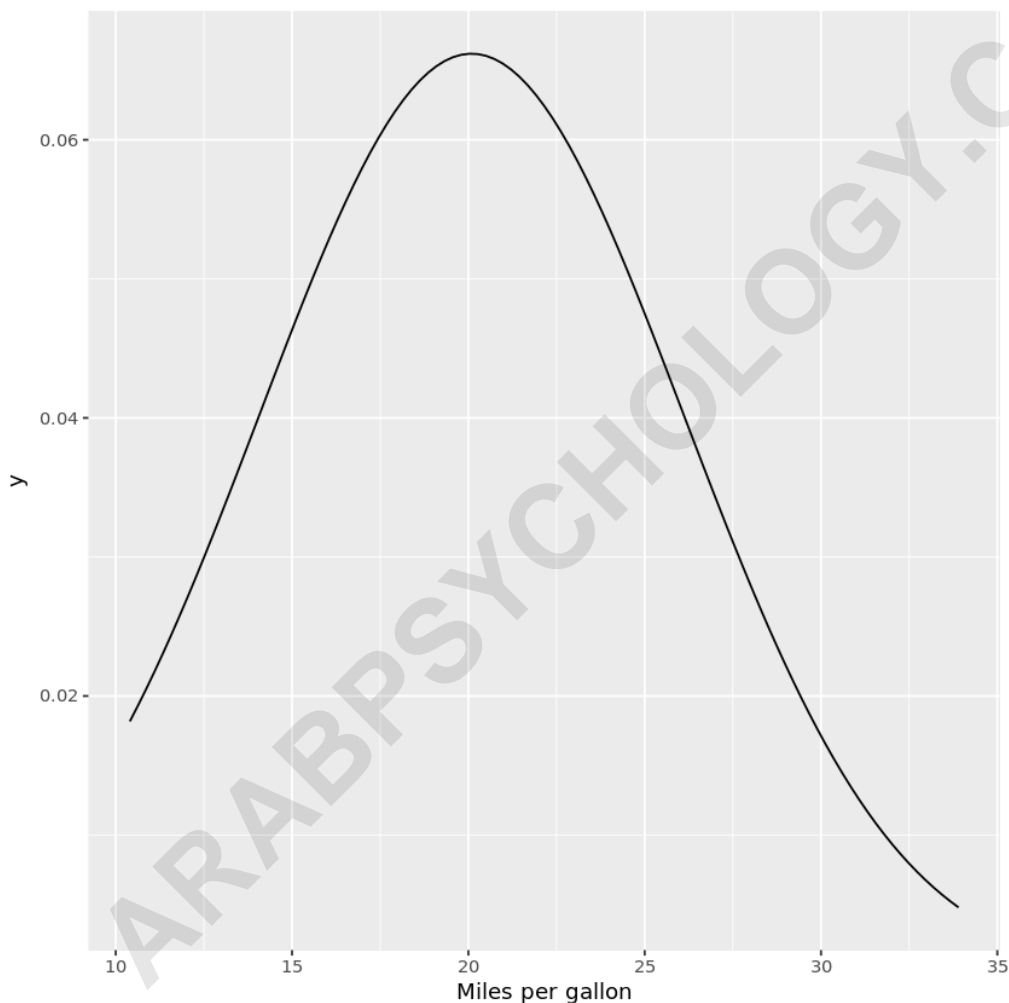
A frequent task in **data analysis** is to see how well an observed variable fits a theoretical **normal distribution**. To illustrate this, we can use the built-in **mtcars** dataset, which contains various performance metrics for several 1973-74 automobile models. Specifically, we can examine the "mpg" (miles per gallon) column to see if it follows a Gaussian shape by calculating its empirical **mean** and **standard deviation**.

In the following code, we pass the **mtcars** dataset to **ggplot** and use **stat\_function** to draw a curve based on the actual statistics of the "mpg" variable. The **args** parameter within **stat\_function** is used to pass the calculated **mean** and **standard deviation** of the dataset to the **dnorm** function, creating a theoretical curve that matches the data's scale.

```
ggplot(mtcars, aes(x = mpg)) +  
stat_function(  
fun = dnorm,
```

```
args = with(mtcars, c(mean = mean(mpg), sd = sd(mpg)))  
) +  
scale_x_continuous("Miles per gallon")
```

This visualization is incredibly useful for verifying **normality assumptions** before performing parametric statistical tests like **t-tests** or **ANOVA**. If the data points (which could be added via **geom\_histogram** or **geom\_density**) align closely with the plotted curve, the analyst can be more confident in the results of their statistical models.



## Conclusion and Best Practices for Distribution Plotting

Mastering the visualization of the **normal distribution** in R is a vital skill for anyone involved in **statistical analysis**. As we have seen, R provides multiple avenues for creating these plots, from the foundational **Base R** functions to the highly customizable **ggplot2** library. Choosing the right tool often depends on the specific requirements of the project--Base R is excellent for quick, low-

dependency tasks, while **ggplot2** is superior for publication-quality graphics and complex data layering.

When creating these visualizations, it is important to follow best practices to ensure clarity. Always label your axes clearly, especially when moving away from the **standard normal distribution**. Including the **mean** and **standard deviation** in the plot title or as annotations can provide essential context for the viewer. Additionally, ensure that your x-axis range is wide enough to show the full extent of the distribution tails, usually spanning at least three to four standard deviations from the center.

In summary, the **dnorm** function is your primary tool for calculating density, and functions like **plot**, **curve**, and **stat\_function** are your primary vehicles for display. By combining these tools with a solid understanding of **statistics**, you can create compelling visual stories that make complex data accessible and actionable. As you continue your journey with R, these plotting techniques will serve as a reliable foundation for all your **data visualization** needs.