

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

Authored by
stats writer

March 10, 2026

RECOMMENDED CITATION

stats writer (2026). *How to Plot a Log Normal Distribution in R: A Step-by-Step Guide*.
PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=135058>

Fundamental Concepts of the Log-Normal Distribution in R

The **log-normal distribution** represents a continuous probability distribution of a **random variable** whose **logarithm** is normally distributed. In the realm of **statistics** and data science, this distribution is particularly significant because it models phenomena that are inherently positive and skewed, such as income levels, scientific measurements, or the duration of specific biological processes. When working within the **R programming language**, researchers have access to a robust suite of tools designed to simulate, analyze, and visualize these distributions with high precision and flexibility.

To effectively visualize a **log-normal distribution**, one must understand the relationship between the underlying **normal distribution** and the resulting exponential transformation. Unlike a standard **normal distribution**, which is symmetric around its **mean**, the log-normal variety is characterized by a long right-side tail, indicating that while most values are clustered at the lower end of the scale, there is a possibility of extremely high values. This makes the distribution essential for modeling **quantitative finance** data and environmental science metrics where negative values are physically impossible.

The process of generating a plot in **R** involves utilizing specialized functions that calculate the **probability density function** (PDF). By specifying the core **parameters** that define the distribution's shape and location, users can create accurate representations of theoretical data. This introductory guide will explore the syntax, **algorithms**, and graphical functions necessary to produce professional-grade visualizations of log-normal data, ensuring that your **data visualization** efforts are both statistically sound and aesthetically clear.

In **R**, the primary functions for handling these distributions are part of the stats package, which is loaded by default. These functions allow for the calculation of densities, cumulative probabilities, quantiles, and random deviates. For the purpose of plotting, the focus remains on the density function, which provides the height of the curve at any given point along the x-axis. Mastering these functions is the first step toward advanced statistical modeling and effective communication of complex data structures.

Decoding the Mathematical Parameters: Meanlog and Sdlog

When defining a **log-normal distribution**, it is crucial to distinguish between the **mean** and **standard deviation** of the distribution itself and the **parameters** used in **R**. The function **dlnorm()** requires two primary inputs: **meanlog** and **sdlog**. These values do not represent the **arithmetic mean** or variance of the log-normal variable; rather, they are the **mean** and **standard deviation** of the variable's natural **logarithm**. This nuance is vital for accurate **statistical parameter** estimation.

The **meanlog** parameter determines the location of the distribution. If you increase the **meanlog**,

the entire curve shifts toward the right along the horizontal axis, though the shape remains fundamentally governed by the **standard deviation**. Because the log-normal distribution is defined on a log scale, small changes in the **meanlog** can result in significant shifts in the actual values of the **random variable**. Understanding this relationship is key to interpreting the scale of your **data visualization** correctly.

The **sdlog** parameter, or the **standard deviation** on the log scale, is responsible for the "shape" of the distribution. A smaller **sdlog** results in a curve that appears more like a **normal distribution**, though still skewed. As the **sdlog** increases, the distribution becomes significantly more skewed, with a much taller peak near the origin and a much longer, thinner tail extending toward infinity. This parameter is the primary driver of the distribution's **skewness** and **kurtosis**.

In **R**, the default values for these **parameters** are set to **meanlog = 0** and **sdlog = 1**. This is often referred to as the standard log-normal distribution. When these defaults are used, the resulting distribution is the exponentiation of the standard **normal distribution**. For most practical applications, however, users will need to specify these values based on empirical data or theoretical requirements to reflect the specific behavior of the system being modeled.

The Essential R Functions: dlnorm and curve

To plot the **probability density function** for a **log-normal distribution** in **R**, we rely on the synergy between two powerful functions:

dlnorm(x, meanlog = 0, sdlog = 1): This function calculates the density of the **log-normal distribution** at specific points. It is the mathematical engine that defines the height of the y-axis for every value of x.

curve(expr, from = NULL, to = NULL): This is a versatile **graphics** function in **R** used to draw the curve of a function over a specified interval. It automatically handles the generation of x-values and evaluates the corresponding y-values.

The **dlnorm()** function is part of a family of functions in **R** (including **plnorm**, **qlnorm**, and **rlnorm**) designed for the **log-normal distribution**. While **rlnorm** is used for generating random numbers and **plnorm** for the **cumulative distribution function**, **dlnorm** is the specific tool required for plotting the "shape" or "density" of the distribution. It takes a vector of quantiles and returns the corresponding probability densities.

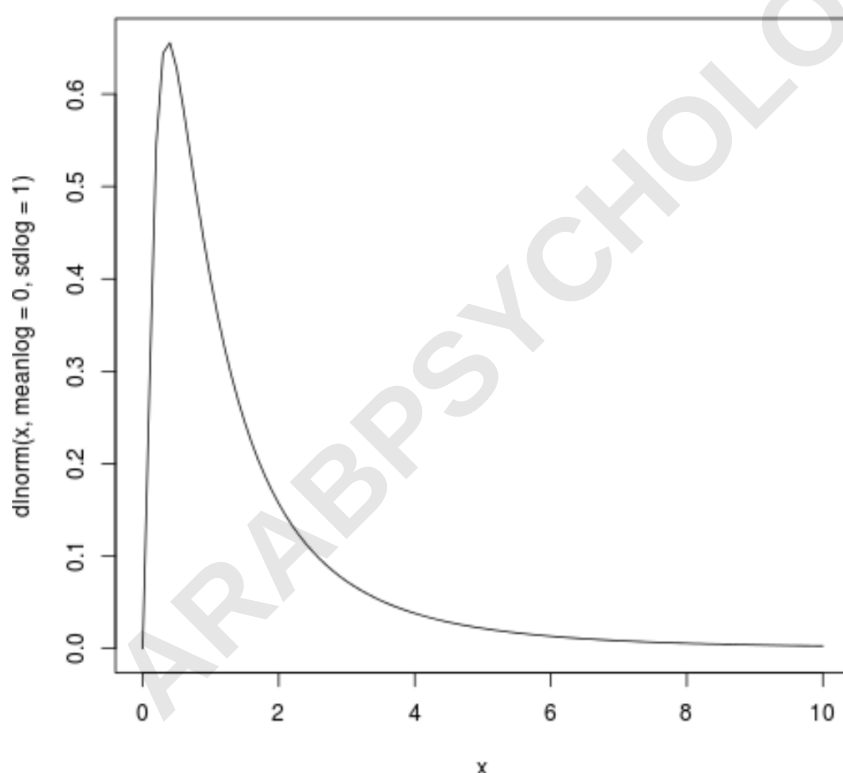
The **curve()** function simplifies the **data visualization** process significantly. Instead of manually creating a sequence of x-values and calculating their densities, you can pass the **dlnorm** function directly as an expression. The **from** and **to** arguments are critical here, as they define the range of the horizontal axis. Since the **log-normal distribution** is only defined for $x > 0$, the **from** parameter should typically be zero or a small positive value.

By combining these two functions, **R** users can generate high-quality plots with very little code. This efficiency is one of the reasons why **R** remains a preferred tool for statistical **data analysis** and academic research. The following sections will demonstrate how to apply these functions in practice, starting with the most basic implementation and moving toward advanced customization.

Generating a Basic Log-Normal Density Plot

The most straightforward method to visualize a **log-normal distribution** is to use the default **parameters** provided by **R**. In the following example, we illustrate how to plot a **probability density function** where the **meanlog** is 0 and the **standard deviation** (on a log scale) is 1. We will define the x-axis range from 0 to 10 to capture the most significant part of the distribution's "hump" and the beginning of its tail.

```
curve(dlnorm(x, meanlog=0, sdlog=1), from=0, to=10)
```

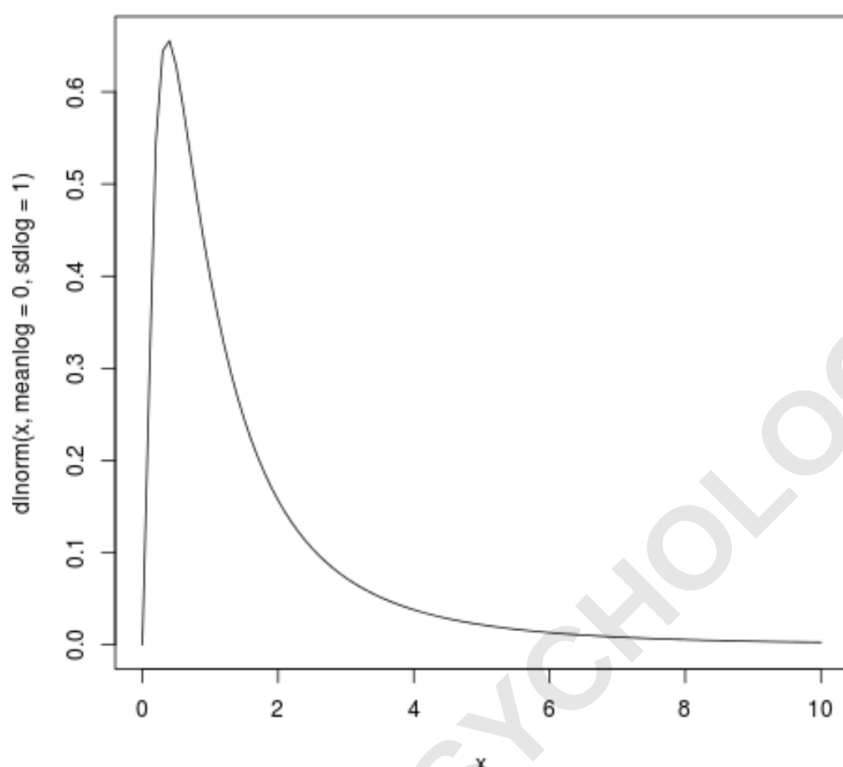


In the code block above, the **curve()** function takes the **dlnorm()** function as its first argument. Note that inside **curve()**, we use **x** as a placeholder variable. **R** internally evaluates this function across the range specified by **from=0** and **to=10**. This creates a clean line graph representing the **probability density function** of the specified log-normal **random variable**.

It is important to recognize that because **meanlog = 0** and **sdlog = 1** are the default settings in the

dlnorm() function, we can simplify the syntax even further. Executing the code without these explicit **parameters** will yield an identical visual result, which is helpful for rapid prototyping and exploratory **data analysis**.

curve(dlnorm(x), from=0, to=10)



This basic plot serves as the foundation for more complex visualizations. While the default output is functional, it lacks the descriptive elements necessary for formal reports or presentations. In the subsequent sections, we will explore how to enhance the **data visualization** by adding titles, modifying axis labels, and adjusting colors to improve readability and communication of the **statistical** findings.

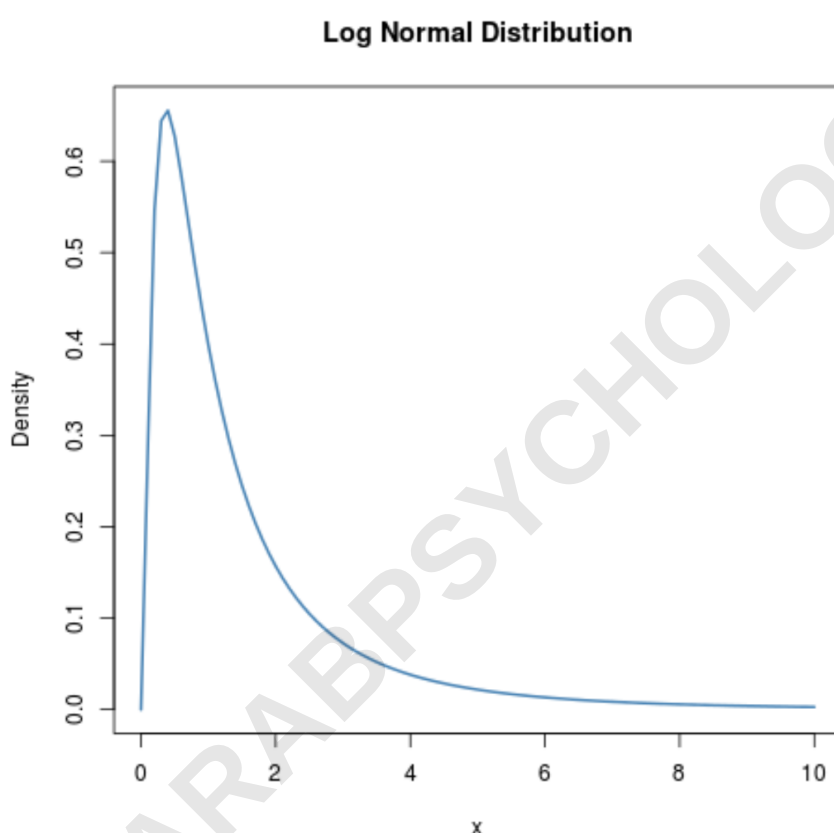
Customizing Plot Aesthetics for Clarity

To transform a basic graph into a professional **data visualization**, **R** provides a variety of graphical **parameters** that can be passed directly into the **curve()** function. Adding a title, descriptive axis labels, and modifying the line's physical appearance can significantly improve the viewer's ability to interpret the **probability density function**. This is especially important when sharing results with stakeholders who may not be familiar with the default **R** plotting style.

The **main** argument allows you to add a header to the plot, while **ylab** and **xlab** allow for the

customization of the vertical and horizontal axis labels, respectively. Furthermore, the **lwd** (line width) parameter can be used to make the curve thicker and more prominent, and the **col** (color) parameter can change the line's color to something more visually striking or consistent with a specific brand palette, such as 'steelblue'.

```
curve(dlnorm(x), from=0, to=10,  
main = 'Log Normal Distribution', #add title  
ylab = 'Density', #change y-axis label  
lwd = 2, #increase line width to 2  
col = 'steelblue') #change line color to steelblue
```



By applying these adjustments, the resulting plot is much more informative. The **density plot** now clearly communicates what it represents, making it suitable for inclusion in a **peer-reviewed** article or a technical blog post. Effective use of white space and color can draw the eye to the most important features of the **log-normal distribution**, such as the location of the peak (the mode) and the rate at which the tail decays.

Beyond these basic settings, **R** allows for even deeper customization, including grid lines, font changes, and background colors. However, for most statistical purposes, the **parameters** shown above provide the best balance between simplicity and professional quality. Consistency in these

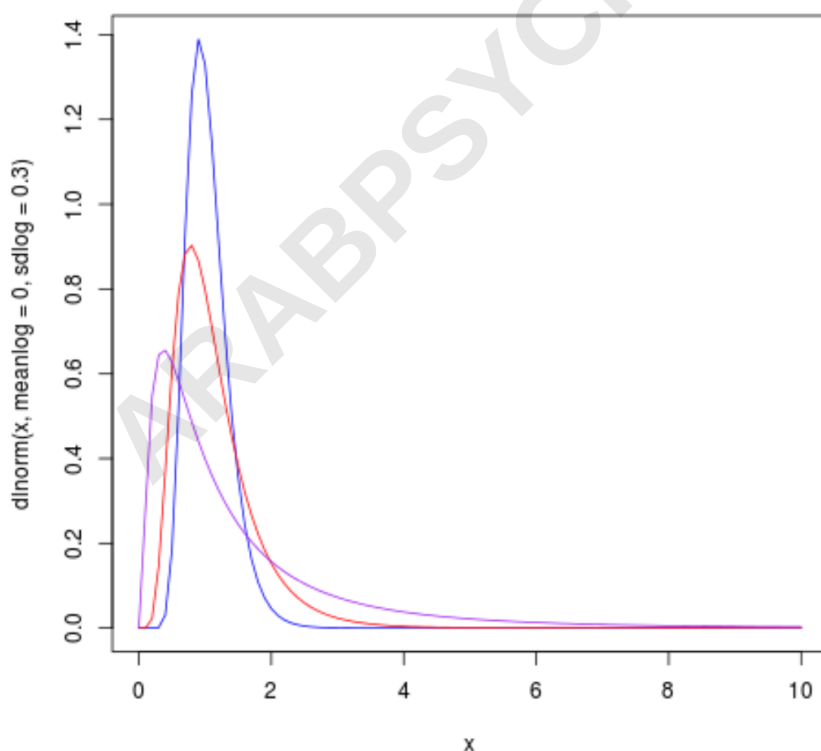
aesthetics across multiple plots ensures a coherent narrative throughout your **data analysis** workflow.

Comparative Analysis: Overlaying Multiple Distributions

In many statistical scenarios, it is useful to compare multiple **log-normal distributions** on a single set of axes. This comparative approach allows researchers to see how changes in **parameters** like **sdlog** affect the shape, height, and spread of the **probability density function**. By overlaying curves, you can visually demonstrate the impact of increasing **variance** or shifting the **mean** of the underlying **normal distribution**.

To achieve this in **R**, we use the **add = TRUE** argument within the **curve()** function. The first call to **curve()** establishes the plot window and scales the axes. Subsequent calls with **add = TRUE** will draw new curves over the existing plot rather than refreshing the entire window. This is a standard technique in **Base R graphics** for building complex, multi-layered visualizations.

```
curve(dlnorm(x, meanlog=0, sdlog=.3), from=0, to=10, col='blue')  
curve(dlnorm(x, meanlog=0, sdlog=.5), from=0, to=10, col='red', add=TRUE)  
curve(dlnorm(x, meanlog=0, sdlog=1), from=0, to=10, col='purple', add=TRUE)
```



In this example, we compare three different **sdlog** values: 0.3, 0.5, and 1.0. Notice how the blue

curve ($\text{sdlog} = 0.3$) is very tall and narrow, indicating that the values are tightly clustered. As the **sdlog** increases to 0.5 (red) and then to 1.0 (purple), the peak of the **log-normal distribution** drops significantly and the tail extends much further to the right. This visual clearly illustrates how the **standard deviation** on the log scale controls the "spread" of the data.

When overlaying multiple curves, it is essential to ensure that the initial plot has an x-axis and y-axis range wide enough to accommodate all the data. If the first curve is very tall and the second is even taller, the second curve might be cut off at the top. To prevent this, you can manually set the **ylim** parameter in the first **curve()** call to provide enough vertical space for all planned **probability density functions**.

Implementing Informative Legends for Data Interpretation

A multi-curve plot is only useful if the viewer can distinguish between the different datasets. This is where the **legend()** function becomes indispensable. A well-placed legend provides the necessary context to decode the colors and line styles used in the **data visualization**. In **R**, the **legend()** function is highly customizable, allowing for precise control over placement, text size, and symbols.

The standard syntax for adding a legend involves specifying the coordinates (x, y) and the labels. The **parameters** are defined as follows:

x, y: The coordinates on the plot where the legend will be anchored.

legend: A character vector containing the text labels for each curve.

col: A vector of colors that match the colors used in the **curve()** functions.

lty: The line type (e.g., 1 for solid, 2 for dashed).

cex: A scaling factor for the text size, useful for improving readability.

bg: The background color of the legend box.

In the following final example, we combine all the previously discussed techniques to create a comprehensive, multi-line **density plot** with a clear, informative legend. This represents the pinnacle of **Base R** plotting for theoretical distributions.

#create density plots

```
curve(dlnorm(x, meanlog=0, sdlog=.3), from=0, to=10, col='blue')
```

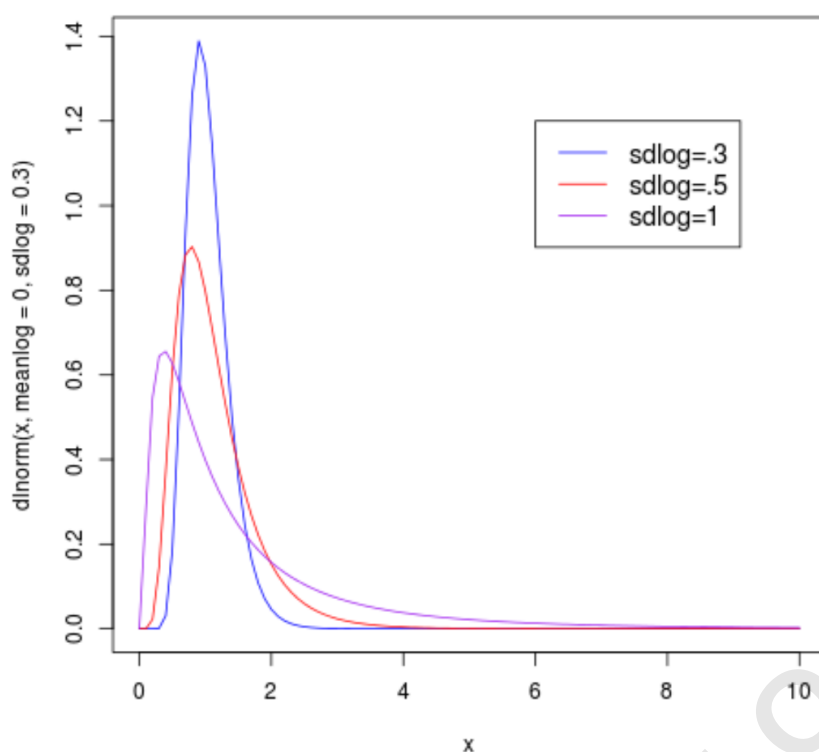
```
curve(dlnorm(x, meanlog=0, sdlog=.5), from=0, to=10, col='red', add=TRUE)
```

```
curve(dlnorm(x, meanlog=0, sdlog=1), from=0, to=10, col='purple', add=TRUE)
```

#add legend

```
legend(6, 1.2, legend=c("sdlog=.3", "sdlog=.5", "sdlog=1"),
```

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



By placing the legend at coordinates (6, 1.2), we ensure it does not overlap with the main peaks of the **log-normal distribution** curves. The **lty=1** argument ensures the legend shows solid lines, matching the lines in the plot. With this final addition, the **data visualization** is complete, providing a clear and mathematically accurate comparison of how varying the **sdlog** parameter transforms the **probability density function** in **R**.