Plot a Normal Distribution in R

Authored by

Mohammed looti

November 9, 2025

The Normal Distribution, often referred to as the Gaussian distribution or the bell curve, is arguably the most critical concept in modern statistics and data analysis. Visualizing this distribution is essential for understanding concepts like probability, sampling, and inferential testing. In the R programming language, users have two primary pathways for generating these plots: leveraging the foundational capabilities of Base R or utilizing the powerful extensions provided by specialized packages like ggplot2. Choosing the right method depends on the required level of customization and the overall aesthetic goal for the visualization. This guide details both approaches, providing clean, executable code for creating professional distribution plots.

Using Base R

The Base R environment offers robust, built-in functions suitable for quick and efficient visualization of statistical models. While perhaps less aesthetically flexible than packages like `ggplot2`, the base plotting functions require no external installations and are perfect for foundational statistical work. We will demonstrate three distinct methods for generating a Normal Distribution plot using these core R capabilities, starting with the explicit definition of the x and y axes.

Example 1: Standard Normal Distribution (Mean = 0 and Standard Deviation = 1)

The standard approach involves manually defining a sequence of x-values and then calculating the corresponding probability density values (y-values) using the density function specific to the normal model. The standard Normal Distribution has a population mean (μ) of 0 and a Standard Deviation (σ) of 1, which serves as a foundational reference point in statistics. By defining the plot parameters explicitly, we gain precise control over the range and resolution of the resulting curve.

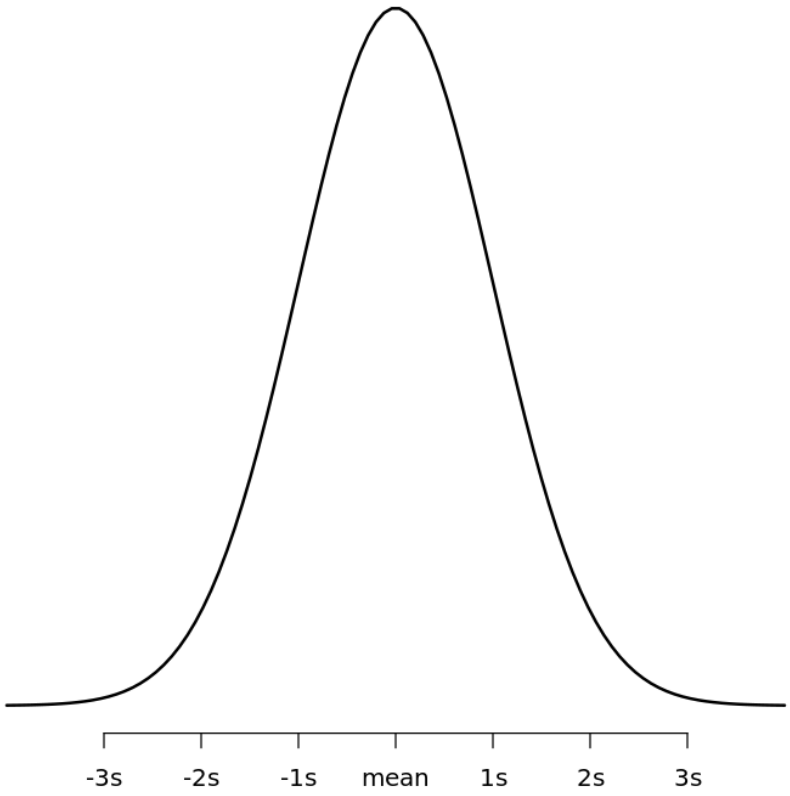
The code below first uses the `seq()` function to generate 100 equally spaced points across the range where the distribution is non-negligible (from -4 to 4 Standard Deviations). It then uses `dnorm()`, which calculates the height of the probability distribution function for the specified x-values, assuming the default parameters (mean=0, sd=1). Finally, the `plot()` function is used to render the curve, connecting the points with a line (`type = "l"`) and customizing the axis labels to reflect standard deviations.

#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 (using the standard dnorm settings: mean=0, sd=1)
y <- dnorm(x)

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

Executing this comprehensive code block generates a canonical representation of the standard bell curve, providing clear visual markers for the central tendency and dispersion based on standard deviation units.



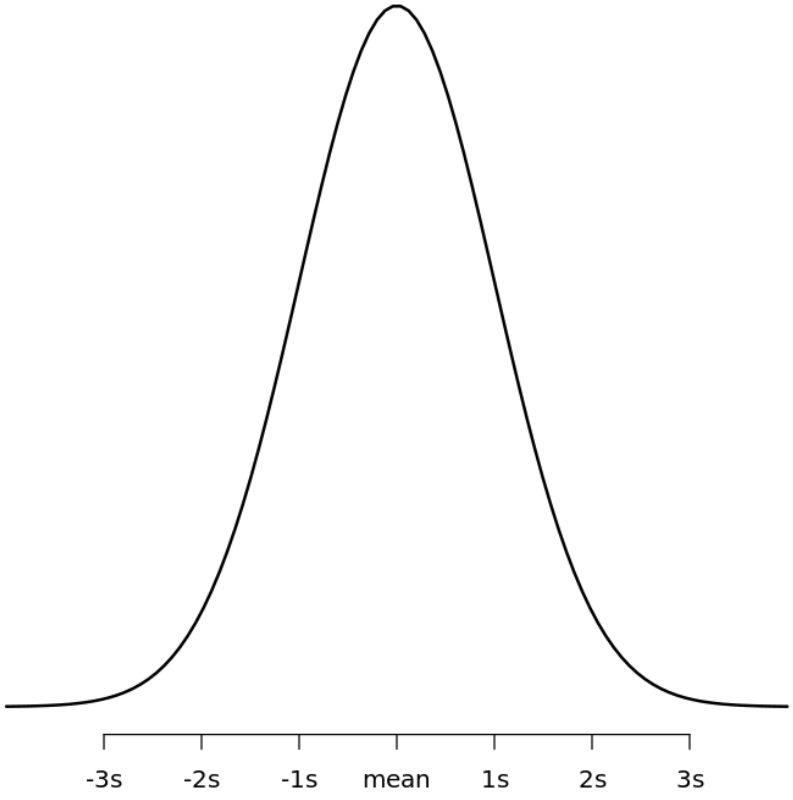
Example 2: Standard Normal Distribution (Using the curve function)

While the previous method offers granular control, R provides a more succinct function specifically designed for plotting mathematical expressions over a range: `curve()`. This function simplifies the process by internally handling the generation of the sequence of x-values and the calculation of the corresponding y-values based on the supplied function name.

When using `curve()`, we simply pass the name of the density function (`dnorm`) as the first argument, followed by the desired limits for the x-axis (e.g., -3.5 to 3.5). This results in highly condensed and efficient code while producing an identical output to the manually constructed plot. This approach is highly recommended for quick visualizations where explicit data point generation is unnecessary.

```
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 expected, this optimized code snippet generates the exact same visual representation of the standard Normal Distribution, confirming the efficiency of the `curve()` function for plotting density functions within Base R.



Example 3: Normal Distribution with Customized Mean and Standard

Deviation

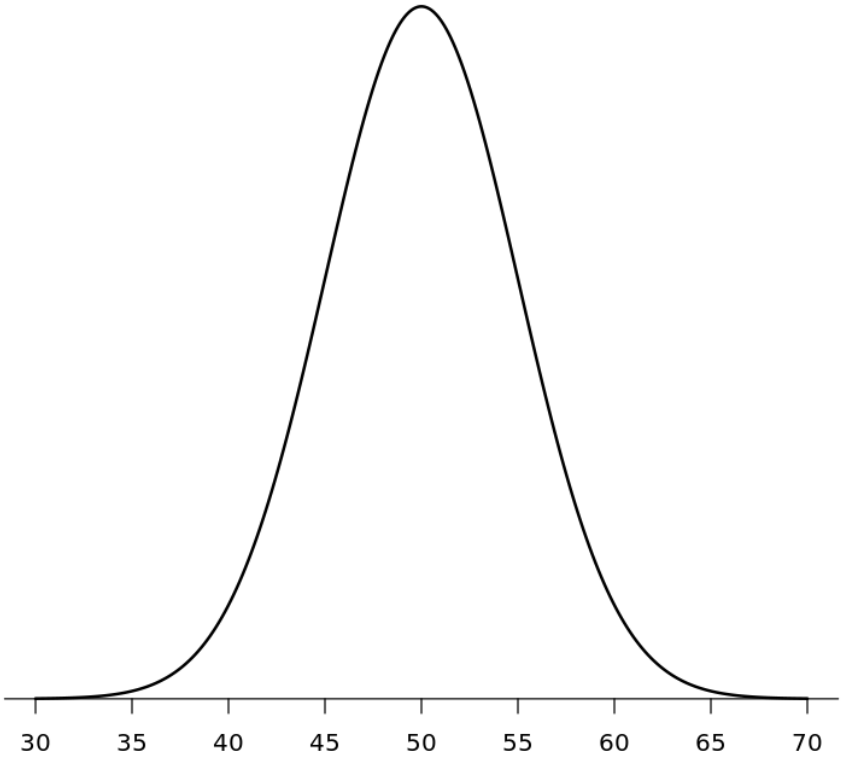
In real-world scenarios, data rarely follows the standard normal model (μ=0, σ=1). Instead, we often need to plot a theoretical normal curve that matches the specific population mean and standard deviation derived from observed data. The `dnorm()` function allows us to specify these parameters directly using the `mean` and `sd` arguments.

To ensure the resulting plot accurately spans the relevant range of the distribution (typically ±4 or ±5 Standard Deviations), we must customize the sequence of x-values. In the example below, we define a population with a mean of 50 and an SD of 5. The sequence `x` is generated by scaling the standard deviation multipliers (the sequence -4 to 4) by the new population SD and shifting the entire range by the population mean. This mathematical transformation ensures the curve is correctly centered and scaled.

```
#define population mean and standard deviation
population_mean <- 50
population_sd <- 5

#define upper and lower bound (though not strictly necessary for the plot itself, this clarifies the range)
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.
#This scales a standard sequence (-4 to 4) to the actual distribution range.
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, using the custom mean and standard deviation parameters
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)
```

The result is a bell curve centered at 50, showcasing how the Base R plotting system can be effectively adapted to model any theoretical Normal Distribution by adjusting the foundational

parameters and coordinate definitions.



Using ggplot2

For data scientists and analysts prioritizing aesthetic quality, layer-based construction, and integration with the Tidyverse ecosystem, the ggplot2 package is the preferred tool in R. ggplot2 operates on the "Grammar of Graphics," allowing users to build complex plots piece by piece (layers, scales, coordinates, and statistics). When plotting theoretical distributions, ggplot2 utilizes the `stat_function()` geometric layer, which draws a curve defined by a function over a specified range.

Example 1: Standard Normal Distribution with ggplot2

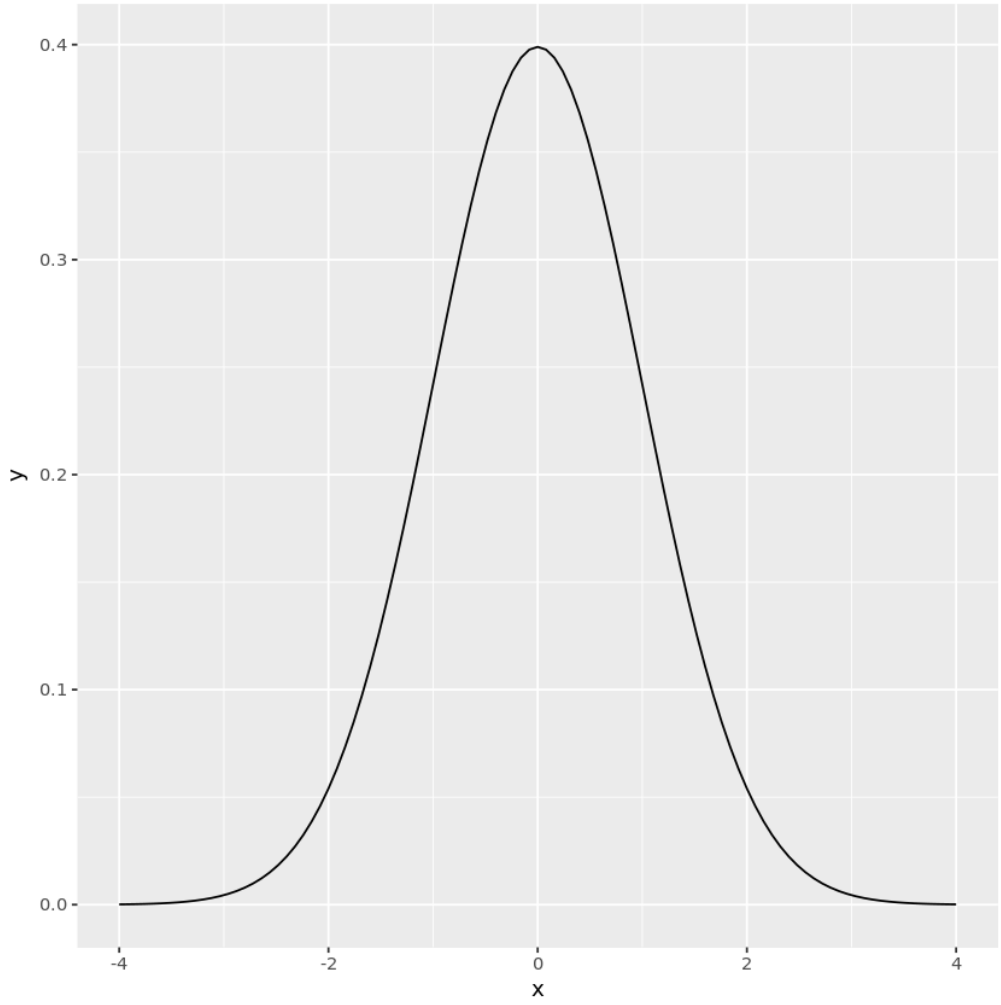
To plot the standard normal curve (μ=0, σ=1) using ggplot2, the process is streamlined significantly compared to Base R's manual sequencing. The core idea is to define a data frame that establishes the range of the x-axis, map that x-axis to the aesthetic layer (`aes(x = x)`), and then use

`stat_function()` to calculate and draw the curve.

The `stat_function()` layer takes the density function, `dnorm`, as its main argument (`fun = dnorm`). Since `dnorm` defaults to mean=0 and sd=1 when no arguments are supplied, this single line of code is sufficient to render the standard bell curve across the defined x-range (in this case, -4 to 4). It is crucial to ensure that the `ggplot2` library is loaded before attempting to execute the plotting commands.

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

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

This code yields a clean, high-resolution plot characteristic of the ggplot2 package, demonstrating its efficiency in visualizing theoretical probability distributions with minimal setup.



Example 2: Normal Distribution Using the 'mtcars' Dataset

A common requirement in data visualization is fitting a theoretical Normal Distribution curve over the empirical data distribution (e.g., a histogram or density plot) to assess how closely the observed data conforms to the normal assumption. This requires calculating the mean and standard deviation directly from the dataset and passing those calculated values to the `dnorm` function.

The following example uses the built-in mtcars dataset in R, specifically examining the miles per gallon (`mpg`) variable. Within the `stat_function()` call, we use the `args` argument to supply a list of parameters (mean and standard deviation) calculated dynamically from the `mpg` column. This ensures the theoretical curve is perfectly aligned with the central tendency and spread of the empirical data, and the `scale_x_continuous` layer adds a descriptive label to the x-axis.

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 technique is crucial for diagnostic work, allowing analysts to visually compare their dataset's characteristics against the assumption of normality, a prerequisite for many classical statistical tests. The resulting plot shows the theoretical normal curve fitted to the specific parameters of the `mpg` variable from the mtcars dataset.