How to create a Stacked Dot Plot in R?

Authored by

stats writer

December 11, 2025

To create a **Stacked Dot Plot** in R, analysts typically rely on the robust functionality provided by the `geom_dotplot()` function within the widely adopted ggplot2 package. This function requires a data frame as input, allowing users to precisely specify the variables intended for plotting and control the visual stacking order of the data points. Furthermore, `ggplot2` offers extensive capabilities for adding plot aesthetics such as dot size, color, transparency, and intricate theme customizations to achieve a highly refined and professional visualization.

A **stacked dot plot** is an essential type of plot in descriptive statistics, designed specifically to display frequencies or counts of observations using dots that stack vertically above a numerical axis. This visualization technique is invaluable for assessing the distribution shape of a single quantitative variable, providing clarity on clustering, gaps, and central tendencies.

There are two primary, distinct methods available within the R ecosystem for creating a high-quality stacked dot plot:

Method 1: Utilizing the core graphics function `stripchart()` available in Base R.
Method 2: Employing the versatile `geom_dotplot()` function from the powerful ggplot2 package.

This comprehensive tutorial will guide you through detailed examples of how to implement each of these methods, highlighting both the basic setup and necessary customization steps to produce clear and insightful stacked dot plots.

Understanding the Significance of Dot Plots

Dot plots offer a clear, immediate representation of data distribution that is often superior to simple histograms, especially when dealing with smaller datasets or discrete variables. Unlike bar charts, which necessitate summarizing data into defined bins, the stacked dot plot preserves the individual identity of each observation or small groups of observations, allowing viewers to easily identify central tendency, spread, and the presence of outliers or clusters within the data with high fidelity.

The stacking mechanism is what crucially differentiates a standard dot plot from a Stacked Dot Plot. When multiple observations share the exact same value or fall within a narrow bin range (depending on the implementation), the corresponding dots are stacked vertically. This stacking provides a direct visual frequency count for that specific value, making direct comparisons between different data points intuitive and efficient. This clarity makes the stacked dot plot a favored tool in preliminary exploratory analysis of a data frame.

Choosing the correct visualization tool is critical for effective statistical communication. For datasets where individual data points matter and the overall distribution needs to be visualized without heavy aggregation, the Stacked Dot Plot provides an optimal balance between granular detail and distributional overview. We will now explore how to leverage R's capabilities, starting

with the robust functions built into its core.

Prerequisites and Setup for R Visualization

Before proceeding with the coding examples, it is essential to ensure you have a functional installation of R and a suitable integrated development environment like RStudio. While the first method we explore relies solely on functions built into Base R and requires no extra packages, the second method necessitates the installation and loading of the highly popular `ggplot2` package, which is fundamental to modern R visualization.

If you have not yet installed `ggplot2`, you must do so using the command `install.packages("ggplot2")` executed in your R console. Once installed, it must be loaded into your current session using `library(ggplot2)` before executing any code that utilizes its specialized functions, such as `geom_dotplot()`. Performing this preparation step ensures all necessary visualization utilities are loaded and ready for immediate use.

All coding examples provided here utilize a small, internally generated sample dataset to illustrate the plotting concepts clearly and reproducibly. In real-world applications, you would naturally replace this data generation step with loading and processing your own structured data, which should always be organized within a data frame object for compatibility with visualization libraries.

Method 1: Generating Stacked Dot Plots using Base R's `stripchart()`

The first and most direct way to create a Stacked Dot Plot in R is by utilizing the `stripchart()` function, which is a key component of the core Base R graphics package. This function was originally designed to produce one-dimensional scatter plots (or strip charts), but critically, it includes specific parameters that allow for effective stacking, thereby transforming the output into a functional stacked dot plot suitable for frequency visualization.

The crucial parameter that dictates the stacking behavior is `method`. By setting the argument `method = "stack"`, we explicitly instruct R to handle coincident data values by stacking the dots vertically above the numerical axis. Other available options for the `method` parameter exist--such as `"overplot"`, where dots overlap completely, and `"jitter"`, where dots are randomly offset to minimize overlap--but `"stack"` is mandatory for generating the desired stacked frequency visualization.

While the `stripchart()` function is known for its speed and its freedom from external package dependencies, its customization options are generally more constrained compared to the layered, grammar-based approach offered by `ggplot2`. Nevertheless, for quick, on-the-fly exploratory analysis and the generation of simple, uncluttered visualizations, it remains a highly efficient and robust choice within the core R environment.

Basic Implementation of `stripchart()` (Example 1)

To demonstrate the basic usage of `stripchart()`, we first need to generate a vector of random integers to simulate raw, unaggregated data. The inclusion of the `set.seed(0)` function at the start of the code ensures that the specific sequence of random data generated is completely reproducible, meaning that any user running this code will obtain the exact same data vector and consequently, the same output plot.

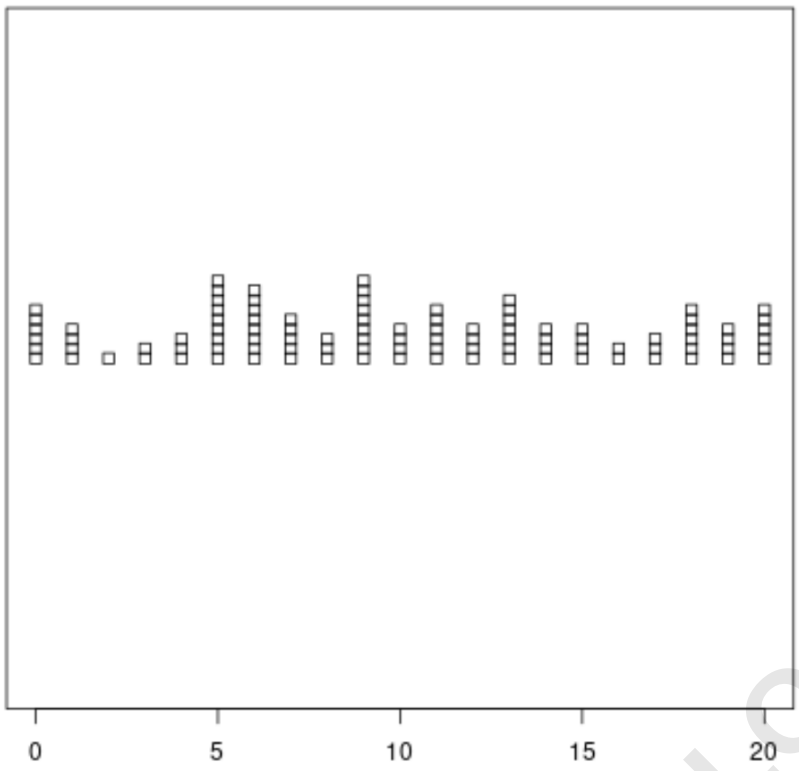
The sample data is generated using the powerful `sample()` function, drawing 100 observations from the discrete range of 0 to 20, with replacement allowed. This process effectively simulates a discrete variable where specific values are repeated frequently, providing the perfect data structure for the dot plot to visually demonstrate the underlying frequency distribution.

The following code snippet illustrates the minimal requirements needed to produce a functional Stacked Dot Plot using the default aesthetic settings of the `stripchart` function:

```
# Create some fake data to visualize the distribution
set.seed(0)
data <- sample(0:20, 100, replace = TRUE)

# Create the basic stacked dot plot using the "stack" method
stripchart(data, method = "stack")
```

The resulting plot clearly shows the distribution of frequencies. In this Base R output, each dot represents a single observation, and the vertical stacks precisely indicate how many times that specific value appears within the dataset. It is worth noting that by default, Base R utilizes hollow circles (`pch=1`) for the plot points, which may not offer optimal visibility or visual weight for very dense distributions, thereby emphasizing the need for aesthetic customization.



Customizing Aesthetics in Base R for Enhanced Clarity

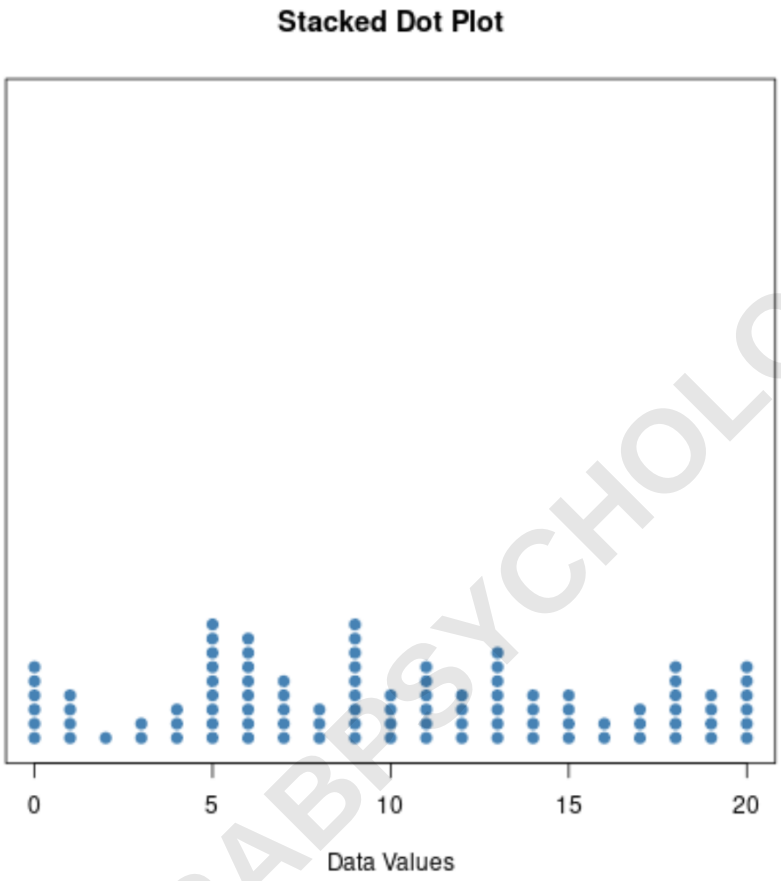
While the basic plot generated by the previous code is statistically functional, enhancing its visual appearance and overall readability is typically essential for professional presentation or academic publication. Base R allows for extensive customization through numerous direct arguments passed to the `stripchart()` function, enabling precise control over plot elements such as dot appearance, axis labels, titles, and margins.

In the following improved example, we introduce several key graphical parameters. The `offset` parameter controls the vertical spacing between the stacked dots, significantly enhancing separation and reducing visual clutter. Setting `pch = 19` specifies the use of solid, filled circles for the plot points, which dramatically improves their visual weight and prominence. The `col` argument sets the color of the dots to a distinct "steelblue," while standard arguments like `main` and `xlab` are used to provide clear, descriptive labels for the plot title and the horizontal axis, respectively, ensuring the plot is fully self-explanatory.

This attention to detail in customization ensures that the final visualization is not only statistically accurate but also highly professional and easy to interpret. The ability to control these fundamental aesthetic elements directly within the function call makes the Base R approach very efficient for rapid styling adjustments, provided the user is familiar with the wide range of available graphical parameters.

```
# Create the same fake data for consistency
set.seed(0)
data <- sample(0:20, 100, replace = TRUE)

# Create customized stacked dot plot with improved aesthetics
stripchart(data, method = "stack", offset = .5, at = 0, pch = 19,
col = "steelblue", main = "Stacked Dot Plot of Sample Data", xlab = "Data Values")
```



Method 2: Utilizing the Power of ggplot2 and geom_dotplot()

For R users who require superior control over plotting layers, thematic styling, and the overall grammar of graphics, the `ggplot2` package is the undisputed standard in modern R visualization. Within the `ggplot2` framework, the primary function dedicated to creating stacked dot plots is `geom_dotplot()`. This geometry function strictly adheres to the package's philosophy of building plots layer by layer, starting with the foundational data source and the necessary aesthetic mappings.

A significant procedural difference when using `ggplot2` compared to Base R is the mandatory

requirement that the input data be supplied in a structured data frame format, rather than a simple vector. Furthermore, the aesthetic mappings (defined using `aes()`) must explicitly link the specific data variables to the appropriate plot axes. For a simple univariate dot plot, we only need to map the variable of interest to the x-axis.

The `geom_dotplot()` function inherently possesses sophisticated mechanisms to handle the stacking process based on the input variable, often treating the x-axis data as continuous and using an algorithm to determine the optimal bin width for dot aggregation and stacking (which can be manually overridden if needed). This layered methodology provides superior flexibility for integrating the dot plot with other visualization components, such as statistical summary transformations, conditional coloring, or faceting based on additional grouping variables.

Creating a Simple Dot Plot with ggplot2 (Example 2)

The process of generating a stacked dot plot using `ggplot2` begins with the crucial step of loading the library into the current R session. Next, the sample data is created, but unlike the Base R method, it must be explicitly structured and stored into a data frame object named `data`, with the numerical variable appropriately labeled as `x`.

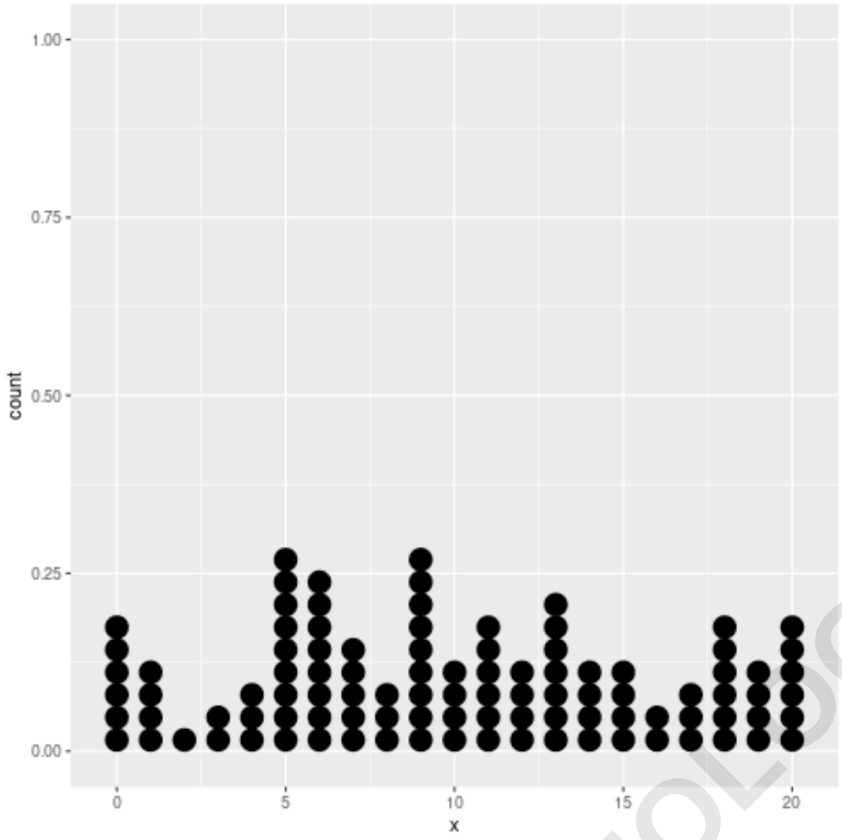
The plot itself is initialized using the `ggplot()` function, which defines the data source and the necessary aesthetic mapping (`aes(x = x)`). The actual visualization layer is then appended using the `geom_dotplot()` function. Running this code with only the required layers generates a plot that is structurally complete but utilizes the default themes, colors, and axis interpretations established by `ggplot2`.

This minimalist approach successfully generates the Stacked Dot Plot but, by default, `ggplot2` often includes a density or count axis (the y-axis) label, which can sometimes be considered redundant or distracting in a visualization primarily focused on the distribution along the x-axis.

```
# Load the essential ggplot2 package
library(ggplot2)

# Create data and store it in a data frame
set.seed(0)
data <- data.frame(x = sample(0:20, 100, replace = TRUE))

# Initialize the plot and add the dot plot geometry layer
ggplot(data, aes(x = x)) +
geom_dotplot()
```



Advanced Customization and Styling in ggplot2

To refine the `ggplot2` visualization further and make it suitable for publication, we leverage specific arguments within `geom_dotplot()` and supplementary scale and labeling functions provided by the package. The primary goal in advanced customization is typically to remove unnecessary visual elements and enhance the focus on the actual data distribution.

Within `geom_dotplot()`, we use `dotsize` to precisely control the diameter of the individual dots and `stackratio` to adjust the vertical density of the stacking, ensuring optimal visual separation. The `fill` argument modifies the interior color of the dots. Crucially, we employ `scale_y_continuous(NULL, breaks = NULL)` to effectively suppress the default y-axis scale and tick marks. This step focuses the plot entirely on the x-axis distribution, aligning with the standard presentation for a univariate Stacked Dot Plot.

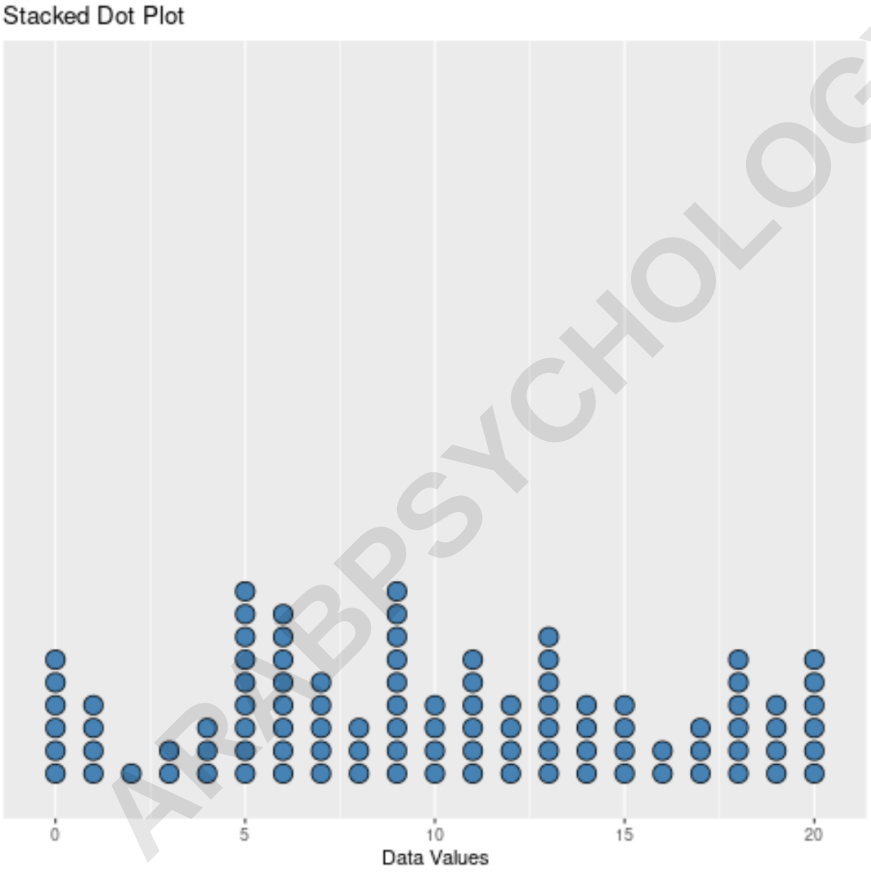
Finally, the `labs()` function is utilized for comprehensive and centralized labeling, setting the plot title and the x-axis label. By explicitly setting the y-axis label (`y = ""`) to an empty string, we remove the label text completely, resulting in a cleaner, professional, and interpretation-focused figure.

 # Load ggplot2 package

library(ggplot2)

```
# Create data frame
set.seed(0)
data <- data.frame(x = sample(0:20, 100, replace = TRUE))

# Create customized stacked dot plot using specific geom and scale arguments
ggplot(data, aes(x = x)) +
geom_dotplot(dotsize = .75, stackratio = 1.2, fill = "steelblue") +
scale_y_continuous(NULL, breaks = NULL) +
labs(title = "Customized Stacked Dot Plot using ggplot2", x = "Data Values", y = "")
```



Conclusion: Choosing the Right Method for Your Data

We have thoroughly explored two powerful and distinct methods for generating a Stacked Dot Plot within the R environment: the efficient `stripchart()` function from Base R graphics and the highly flexible `geom_dotplot()` function from the `ggplot2` package.

The optimal choice between these two methods should be guided by the user's specific

requirements and the overall complexity of the intended visualization. If the primary goal is rapid, straightforward visualization of a single variable distribution without the need for complex layering or intricate theme integration, the Base R `stripchart()` remains a swift and effective solution. Conversely, for projects that demand high aesthetic control, seamless integration with other geometric objects, or the capability to easily facet the plot based on additional grouping variables, the `ggplot2` approach is strongly recommended, despite the slightly more involved setup requiring strict adherence to the data frame structure.

Mastering both techniques ensures that you are fully equipped to handle any univariate distribution visualization task in R. Utilizing these visualization tools effectively allows researchers and analysts to convey complex frequency information clearly and engagingly, moving beyond simple numerical summaries to provide rich visual context for their data.

You can find more detailed R tutorials and official documentation on the comprehensive R project website.