How to make pie charts in ggplot2 (With Examples)

Authored by

stats writer

December 20, 2025

Creating effective pie charts is a common requirement in data visualization. While ggplot2 does not have a dedicated `geom_pie()` function, it offers a powerful and flexible method for generating these circular visualizations by combining other geometric layers and coordinate systems. This technique allows data scientists and analysts using R to produce highly customizable and publication-ready graphics.

This comprehensive guide details the step-by-step process for crafting visually appealing and informative pie charts using the ggplot2 package. We will cover everything from initial data preparation in a data frame to advanced aesthetic customization, ensuring that your output is both accurate and compelling.

Fundamentally, a **pie chart** is a circular statistical graphic divided into slices to illustrate numerical proportion. In the context of R and ggplot2, we generate this graphic by taking a standard stacked bar chart and transforming its coordinate system. This tutorial focuses on using the specialized functions necessary to achieve this transformation seamlessly.

Understanding the Technique: Bar Charts to Pie Charts

Unlike some visualization libraries, ggplot2 requires a clever transformation to create a circular display. This process relies on two primary components: the `geom_bar()` function, which handles the segment sizing, and the `coord_polar()` function, which bends the resulting bars into a circle. The resulting output is mathematically equivalent to a traditional pie chart, representing proportions of a whole.

To begin, your data must be structured correctly. A key prerequisite is having data aggregated by category, where one variable defines the segments (the slices) and another variable defines the numerical value or count (the size of the slices). This setup is critical because the `geom_bar()` function expects summarized data when using the `stat="identity"` argument, which is essential for mapping pre-calculated values directly to bar heights.

Preparing the Data Frame for Visualization

Before any visualization can occur, we must define the input data. We utilize an R data frame to store the categories and their corresponding amounts. This structure ensures that ggplot2 can correctly map aesthetic properties like fill color and height (which becomes the slice size).

For this example, we will construct a simple data frame representing four distinct categories (A, B, C, D) and their respective numerical contributions. This dataset serves as the foundation for all subsequent chart constructions and modifications demonstrated in this tutorial. It is important that the sum of the amounts represents the total population being analyzed.

How to Make a Basic Pie Chart

The core mechanism for generating the chart involves three steps: initializing ggplot2, creating the stacked bar chart, and finally, applying the polar coordinate system. The `geom_bar()` function is configured with `stat="identity"` to ensure that the bar height corresponds directly to the `amount` variable in our data, rather than counting observations. Furthermore, setting `width=1` ensures the bar consumes the full width available, creating the necessary block for the circular transformation.

The crucial step is introducing `coord_polar()`. By applying `coord_polar("y")`, we transform the Cartesian coordinate system into a polar system. The 'y' argument specifies that the axis which formerly represented the height (the numerical value) should now represent the angle of the slice, effectively turning the stacked bars into circular segments.

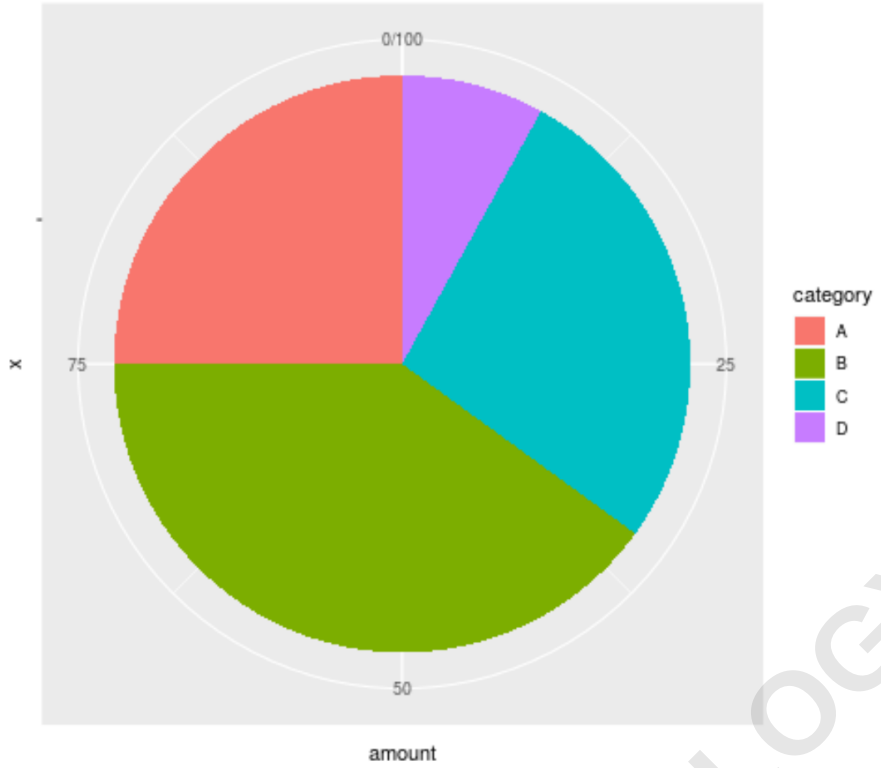
The following code block demonstrates how to load the required library, define the sample data, and construct the rudimentary pie chart:

library(ggplot2)

#create data frame
data <- data.frame("category" = c('A', 'B', 'C', 'D'),
"amount" = c(25, 40, 27, 8))

#create pie chart
ggplot(data, aes(x="", y=amount, fill=category)) +
geom_bar(stat="identity", width=1) +
coord_polar("y", start=0)

As illustrated below, the initial output, while structurally correct, includes unnecessary elements such as axis labels and background grids inherited from the default polar coordinate system. These elements often clutter the visual field and distract from the core proportional data, necessitating further refinement.



Refining Aesthetics with `theme_void()`

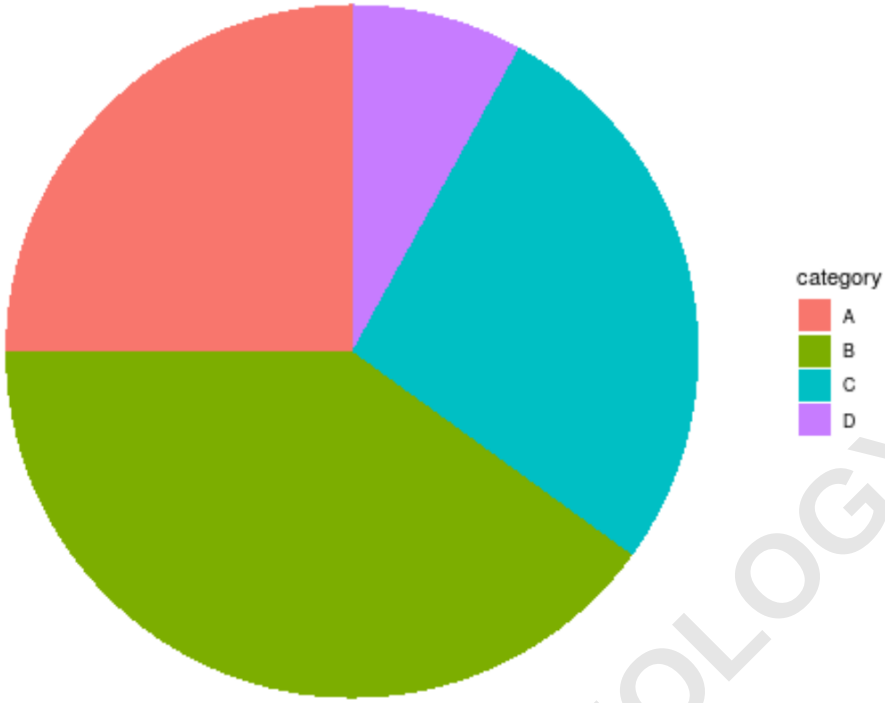
The standard ggplot2 theme is designed for Cartesian plots, meaning it displays axis lines, ticks, and labels that are generally redundant or confusing on a pie chart. The simplest and most effective way to address this visual clutter is by applying the `theme_void()` function. This theme is specifically designed to minimize non-data ink, removing backgrounds, axis lines, and ticks, thus immediately improving the clarity of the visualization.

By incorporating `theme_void()` into the plot pipeline, we focus the viewer's attention entirely on the proportional slices and the legend, which now serves as the primary key for decoding the chart. This small addition dramatically transforms the chart from a technical diagram into a polished graphic suitable for presentations or reports. It is considered best practice when displaying pie charts in R.

```
ggplot(data, aes(x="", y=amount, fill=category)) +
geom_bar(stat="identity", width=1) +
coord_polar("y", start=0) +
theme_void()
```

The result of applying this theme is a clean, minimalist chart, free from distracting elements, allowing the proportions themselves to convey the necessary information clearly. This marks a

significant visual improvement over the default output, showcasing the power of ggplot2's thematic system.



Adding Data Labels Inside the Slices

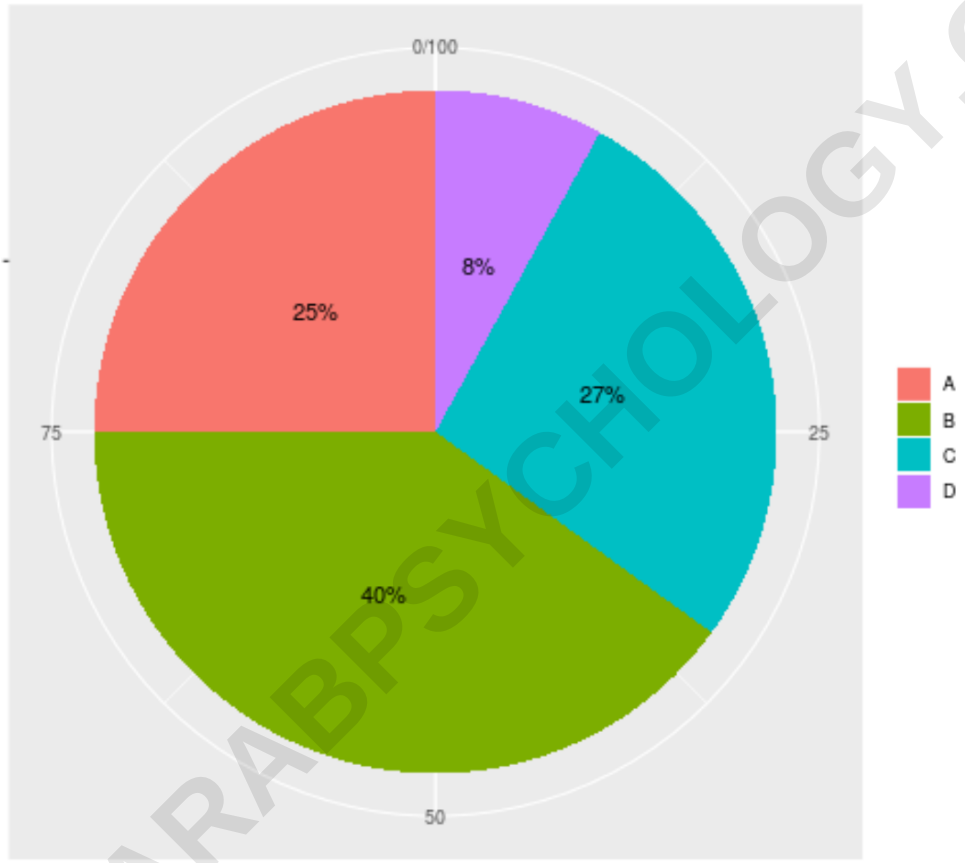
While the legend provides category identification, displaying the numerical values or percentages directly within the slices significantly enhances the chart's immediate informational value. To achieve this, we introduce the `geom_text()` layer, combined with calculated positioning and appropriate labeling functions.

The `geom_text()` function requires careful positioning to ensure labels are centered within their respective slices. This is achieved using `position = position_stack(vjust=0.5)`. The `position_stack()` geometry ensures that the text labels are stacked correctly on top of the bars (which are now slices), and `vjust=0.5` centers the text vertically within each segment. Furthermore, we use `labs(x = NULL, y = NULL, fill = NULL)` to explicitly remove any remaining axis or legend titles that might still be present.

For formatting the labels, we use `paste0(amount, "%")` within the `aes()` mapping of `geom_text`. Although our example data used raw values (25, 40, etc.), labeling them with a percentage sign is a common convention for pie charts, assuming the input values represent proportions or percentages summing to 100. This modification results in a highly legible and self-contained visualization.

```
ggplot(data, aes(x="", y=amount, fill=category)) +
geom_bar(stat="identity", width=1) +
coord_polar("y", start=0) +
geom_text(aes(label = paste0(amount, "%")), position = position_stack(vjust=0.5)) +
labs(x = NULL, y = NULL, fill = NULL)
```

The resulting image displays the numerical proportions clearly integrated into the visual segments. This approach significantly improves the chart's accessibility, allowing viewers to quickly identify both the size and the precise value of each category without relying solely on the legend.



Advanced Color Customization using `scale_fill_manual()`

While ggplot2 provides excellent default colors, tailoring the color palette is often necessary to align with corporate branding, publication standards, or to improve the visual contrast between slices. The `scale_fill_manual()` function offers granular control, allowing the user to specify exact hexadecimal color codes for each category.

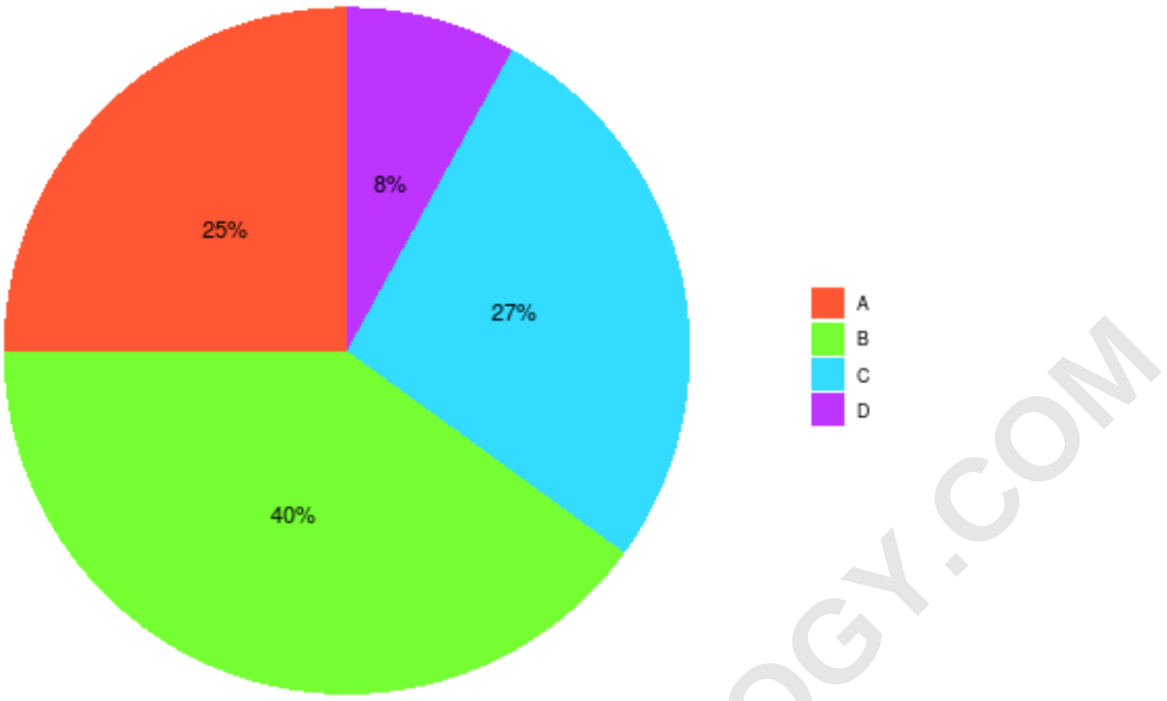
To implement custom colors, we must provide a vector of hex colors that corresponds precisely to

the categories defined in the data frame. In this example, we not only define custom colors but also refine the overall theme further by utilizing `theme_classic()` and manually removing remaining axis elements using `theme()` arguments (`axis.line = element_blank()`, `axis.text = element_blank()`, and `axis.ticks = element_blank()`). This provides a highly sophisticated and customized appearance.

Selecting harmonious colors is crucial for effective data visualization. Tools like the provided Hex Color Picker are invaluable for identifying color combinations that are both aesthetically pleasing and accessible, ensuring that the chart communicates effectively across various mediums.

```
ggplot(data, aes(x="", y=amount, fill=category)) +
geom_bar(stat="identity", width=1) +
coord_polar("y", start=0) +
geom_text(aes(label = paste0(amount, "%")), position = position_stack(vjust=0.5)) +
labs(x = NULL, y = NULL, fill = NULL) +
theme_classic() +
theme(axis.line = element_blank(),
axis.text = element_blank(),
axis.ticks = element_blank()) +
scale_fill_manual(values=c("#FF5733", "#75FF33", "#33DBFF", "#BD33FF"))
```

The customization using `scale_fill_manual()` provides the ultimate flexibility, allowing designers to meticulously control the visual identity of their pie chart to match specific requirements. This demonstrates the extent to which ggplot2 can be manipulated beyond its default settings.



Tip: *Use this Hex Color Picker to find combinations of hex color codes that go well together for your visualization projects.*

Leveraging Predefined Palettes with `scale_fill_brewer()`

While manual color selection is powerful, it can be time-consuming. For rapid, yet high-quality, color selection, ggplot2 integrates seamlessly with the RColorBrewer palettes through the `scale_fill_brewer()` function. RColorBrewer offers scientifically chosen, perceptually uniform palettes optimized for data display, ensuring that color differences accurately reflect data differences.

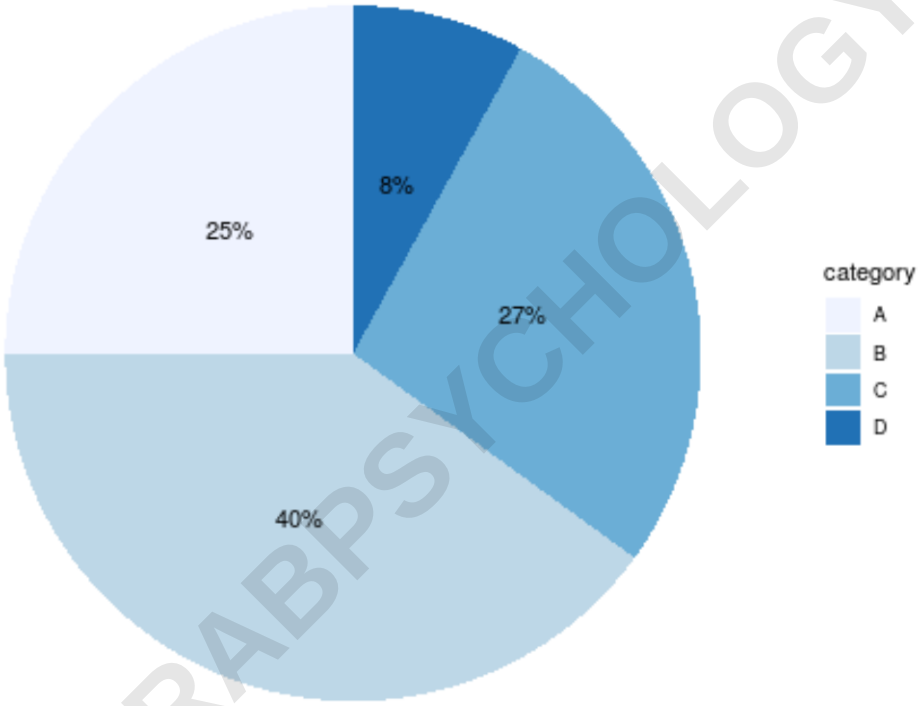
Using `scale_fill_brewer()` simplifies the color specification process significantly. Instead of providing a vector of individual hex codes, the user simply specifies a desired palette name (e.g., "Blues", "Set1", "YlGn"). These palettes are categorized as sequential, diverging, or qualitative, allowing analysts to choose the best color scheme based on the nature of their categorical data.

In the example below, we utilize the "Blues" palette, a sequential color scheme, which works well for displaying differences in magnitude, even if the categories themselves are qualitative. This provides a professional and accessible color application with minimal coding effort, showcasing an efficient method for achieving sophisticated aesthetic results in R.

ggplot(data, aes(x="", y=amount, fill=category)) +
geom_bar(stat="identity", width=1) +

```
coord_polar("y", start=0) +
geom_text(aes(label = paste0(amount, "%")), position = position_stack(vjust=0.5)) +
labs(x = NULL, y = NULL) +
theme_classic() +
theme(axis.line = element_blank(),
axis.text = element_blank(),
axis.ticks = element_blank()) +
scale_fill_brewer(palette="Blues")
```

The chart generated using `scale_fill_brewer()` demonstrates a rapid yet high-quality approach to color application, reinforcing the flexibility of ggplot2 for effective data visualization.



Summary of Key Functions for Pie Chart Generation

Creating a pie chart in ggplot2 is an exercise in layer combination and coordinate transformation. Mastering this technique allows for the creation of precise and highly customized circular graphics. The process relies on correctly sequencing the core functions:

`geom_bar()` with `stat="identity"`: This layer is used to define the height of the bars according to the input values, which will eventually become the angular size of the slices.
`coord_polar()`: This function is the engine of the transformation, converting the stacked bar

geometry into a circular layout. Specifying the 'y' axis is crucial here, as it maps the height dimension to the angular dimension.
`theme_void()`: Essential for cleaning up the visual output by removing extraneous axes and background elements that clutter the final circular plot.

By following these steps and utilizing the customization options for labels and colors, you can ensure your proportional data is presented clearly and professionally in any R environment.

Further Exploration in ggplot2

The techniques explored here--combining geometries, transforming coordinates, and manipulating scales--are central to creating various advanced visualizations in R. We encourage readers to explore other plotting capabilities offered by the ggplot2 library.

Below are links to additional tutorials covering other specialized charting methods within this powerful data visualization framework:

How to Create a Grouped Boxplot in R Using ggplot2
How to Create a Heatmap in R Using ggplot2
How to Create a Gantt Chart in R Using ggplot2