How do you create a log scale in ggplot2?

Authored by

stats writer

December 19, 2025

When visualizing complex datasets, particularly those spanning several orders of magnitude, applying a log scale becomes essential. Logarithmic transformations help manage skewed data distributions and reveal patterns that would otherwise be obscured on a linear scale. To effectively implement this transformation within the powerful ggplot2 visualization package in R, users must correctly specify the transformation mechanism for the desired axis.

The process of creating a log scale involves transforming the data values before they are mapped to the physical coordinates of the plot area. Although ggplot2 offers specific functions like `scale_x_log10()` or `scale_y_log10()`, the most flexible and widely adopted method uses the generic continuous scale functions combined with the `trans` argument, set to `'log10'`. This method allows for seamless integration of custom breaks, limits, and specialized labels, providing granular control over the final visualization output.

Why Utilize Logarithmic Scales in Data Visualization?

When analyzing data where variables vary exponentially, such as population growth, economic indicators, or scientific measurements like pH levels, a linear axis can severely compress the lower values, making subtle changes invisible, while massively separating the highest values. A log scale compresses the high end of the scale and expands the low end, ensuring that relative differences remain consistent across the entire range of data. This allows viewers to observe proportionate changes rather than absolute differences, leading to a more insightful and accurate representation of the data's underlying structure.

The need to convert the x-axis or y-axis scale of a ggplot2 plot into a logarithmic format is a common requirement in statistical programming. ggplot2 provides two primary, highly versatile mechanisms for achieving this transformation. Understanding the subtle differences between these two methods is crucial for selecting the approach that best suits the specific visualization requirements, particularly regarding the display of axis labels and the underlying statistical meaning.

We will focus on two methods using the core functionality of ggplot2. Both rely on the `log10` transformation, which is standard for base-10 log scaling. These methods are preferred because they offer superior control and integration compared to simple pre-transformation of data before plotting.

Method 1: Utilizing scale_continuous() Functions for Transformation

The most robust way to handle scale transformations in R's ggplot2 is by using the scale_y_continuous() or `scale_x_continuous()` functions. These functions are designed to manage all aspects of continuous axes, including breaks, labels, limits, and, crucially, transformations. By passing the argument `trans = 'log10'`, you instruct ggplot2 to apply the log

base 10 transformation to the axis data space, while ensuring that the axis labels remain consistent with the original data values.

This approach transforms the raw data values internally before plotting them to the geometric space, but it ensures that the labels displayed on the axis correspond to the original, untransformed data. This is often the desired behavior for interpretability. The `trans` argument relies on the underlying scales package, which defines the mathematical procedures for various transformations, including log, square root, and reverse transformations. Using scale_y_continuous() provides a comprehensive solution for controlling every facet of the resulting log axis.

Below is the standard syntax for applying a log base 10 transformation to both the X and Y axes using this method:

ggplot(df, aes(x=x, y=y)) +
geom_point() +
scale_y_continuous(trans='log10') +
scale_x_continuous(trans='log10')

Method 2: Employing coord_trans() for Coordinate System Transformation

The second viable method involves altering the coordinate system itself rather than the individual scales. The coord_trans() function modifies the entire plotting coordinate system. Unlike the scale functions (Method 1), which transform the data before mapping, coord_trans() transforms the coordinates after the data has been mapped to the aesthetic space. This means it applies the transformation late in the plotting process.

While often yielding visually similar results to the `scale_continuous()` method, coord_trans() can sometimes lead to slight differences in how statistical summaries (like smooth lines or confidence intervals generated by `geom_smooth()`) are calculated and displayed, as those calculations occur before the coordinate transformation. If you are only interested in a quick visualization transformation and don't require complex custom scale settings, coord_trans() provides a clean, concise way to apply a transformation across the entire panel.

To apply a log base 10 transformation to both axes using this coordinate system approach, you specify the desired transformation type directly within the function call:

ggplot(df, aes(x=x, y=y)) +
geom_point() +
coord_trans(y ='log10', x='log10')

Enhancing Readability: Formatting Axis Labels with Exponents

When using a log scale, it is highly beneficial to format the axis labels to clearly display exponents (e.g., 10^2, 10^3, 10?) instead of the numerical values (100, 1000, 10000). This convention significantly aids in interpreting the logarithmic nature of the axis. To achieve this professional formatting, we must leverage specialized functions available within the supplementary scales package, which seamlessly integrates with ggplot2.

The scales package provides two key helper functions: `trans_breaks()` and `trans_format()`. `trans_breaks()` is used to calculate aesthetically pleasing break points on the transformed axis. It takes the name of the transformation (e.g., `'log10'`) and a function that performs the inverse transformation (e.g., raising 10 to the power of x). Subsequently, `trans_format()` defines how these break points should be visually displayed, often using `math_format()` to render exponents correctly.

Integrating these functions within scale_y_continuous() provides the definitive method for creating fully customized logarithmic axes:

```
ggplot(df, aes(x=x, y=y)) +
geom_point() +
scale_y_continuous(trans='log10',
breaks=trans_breaks('log10', function(x) 10^x),
labels=trans_format('log10', math_format(10^.x)))
```

The following detailed examples illustrate how to apply these methods in practice using sample data within the R environment.

Example 1: Log Scale Using scale_y_continuous()

This example demonstrates the utilization of the scale_y_continuous() function, focusing specifically on transforming the Y-axis (the vertical axis) into a log base 10 scale. This is the recommended default method for applying transformations to continuous axes in ggplot2 because it ensures that statistical operations are performed on the original data, and the transformation is applied only for visual presentation, maintaining data integrity.

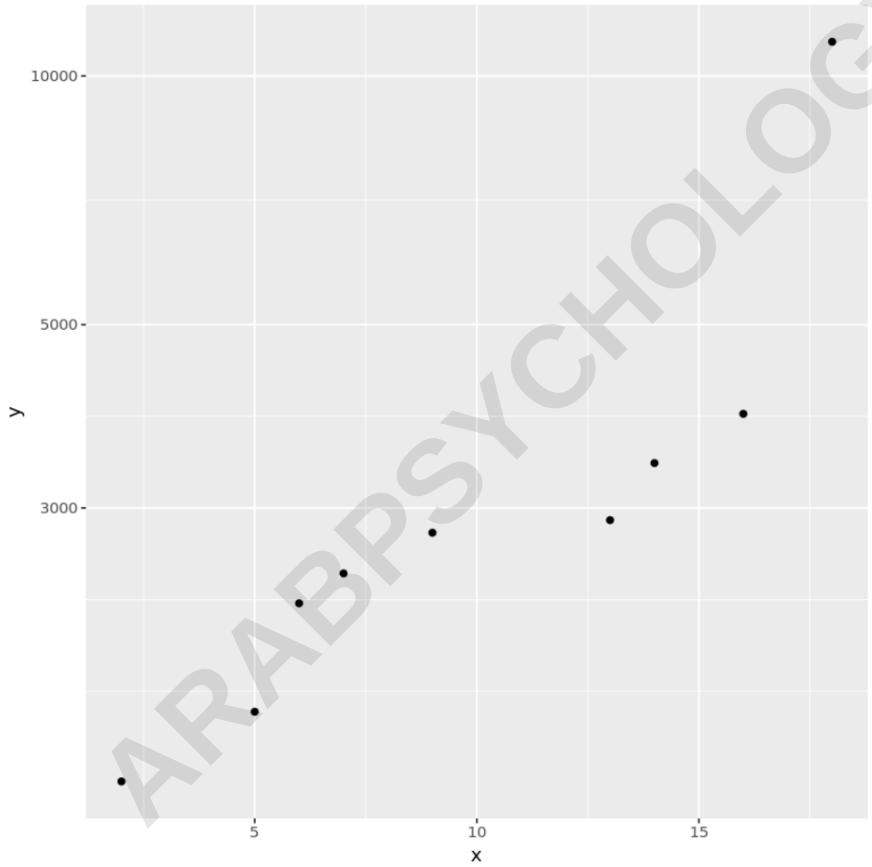
We first load the necessary library and create a simple data frame. This data frame contains one Y-value (11000) that is significantly higher than the others, making it an ideal candidate for logarithmic scaling to prevent compression of the lower values. The crucial step is adding the scale_y_continuous() layer and setting the `trans` argument to `'log10'`. Notice that the resulting labels are standard numeric values (e.g., 1000, 10000), but the spacing between them is

logarithmic.

library(ggplot2)

#create data frame with varying magnitudes
df <- data.frame(x=c(2, 5, 6, 7, 9, 13, 14, 16, 18),
y=c(1400, 1700, 2300, 2500, 2800, 2900, 3400, 3900, 11000))

#create scatterplot with log scale on y-axis using scale_y_continuous()
ggplot(df, aes(x=x, y=y)) +
geom_point() +
scale_y_continuous(trans='log10')



Observing the resulting plot, one can clearly see how the points are distributed. The large difference between the highest point (11000) and the rest of the data is managed effectively. The logarithmic spacing ensures that the clustering and relative change among points between 1000 and 4000 are much more visible than they would be on a standard linear plot where the 11000 point would dominate the upper half of the chart.

Example 2: Log Scale Using coord_trans()

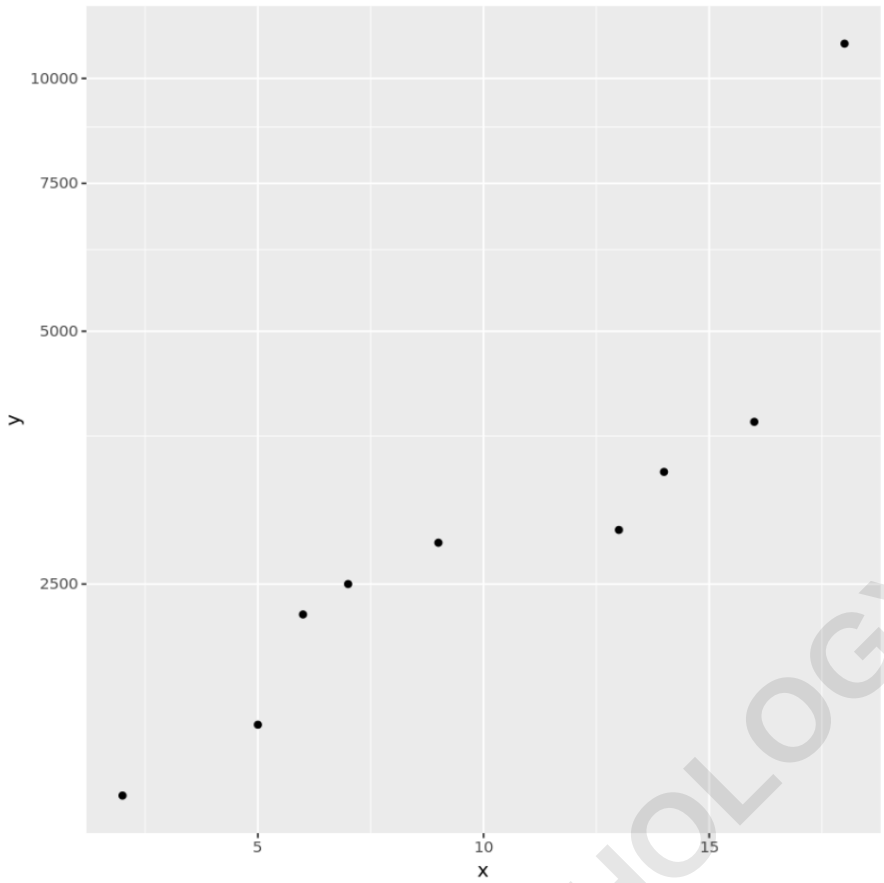
This second example demonstrates the use of the coord_trans() function to achieve a log scale transformation. As noted previously, this method operates on the coordinate system itself after the data has been positioned, rather than altering the scale mapping before positioning. This makes it a straightforward choice when only coordinate rearrangement is required and fine-tuning of axis breaks or labels is not a priority.

We use the exact same data frame as in the previous example to allow for a direct visual comparison of the output. The implementation is cleaner as it only requires one extra line specifying the transformation within the coord_trans() layer. We specify `y='log10'` to apply the log base 10 transformation solely to the vertical coordinates.

library(ggplot2)

```
#create data frame
df <- data.frame(x=c(2, 5, 6, 7, 9, 13, 14, 16, 18),
y=c(1400, 1700, 2300, 2500, 2800, 2900, 3400, 3900, 11000))

#create scatterplot with log scale on y-axis using coord_trans()
ggplot(df, aes(x=x, y=y)) +
geom_point() +
coord_trans(y='log10')
```



Visually, the output produced by coord_trans() is almost identical to the plot generated using scale_y_continuous() for this simple scatterplot. However, it is paramount to remember that the choice between Method 1 and Method 2 can have significant repercussions if you introduce complex statistical layers like quantile regression or complex error bars, which might behave differently depending on whether the data was scaled before or after the statistics were calculated.

Example 3: Custom Log Scale Labels with Exponents

The most visually informative way to present logarithmic data involves labeling the axes using exponential notation. This example demonstrates the advanced usage of the scale_y_continuous() function in conjunction with the powerful scales package. By customizing both the `breaks` and `labels` arguments, we can create a sophisticated and statistically accurate display.

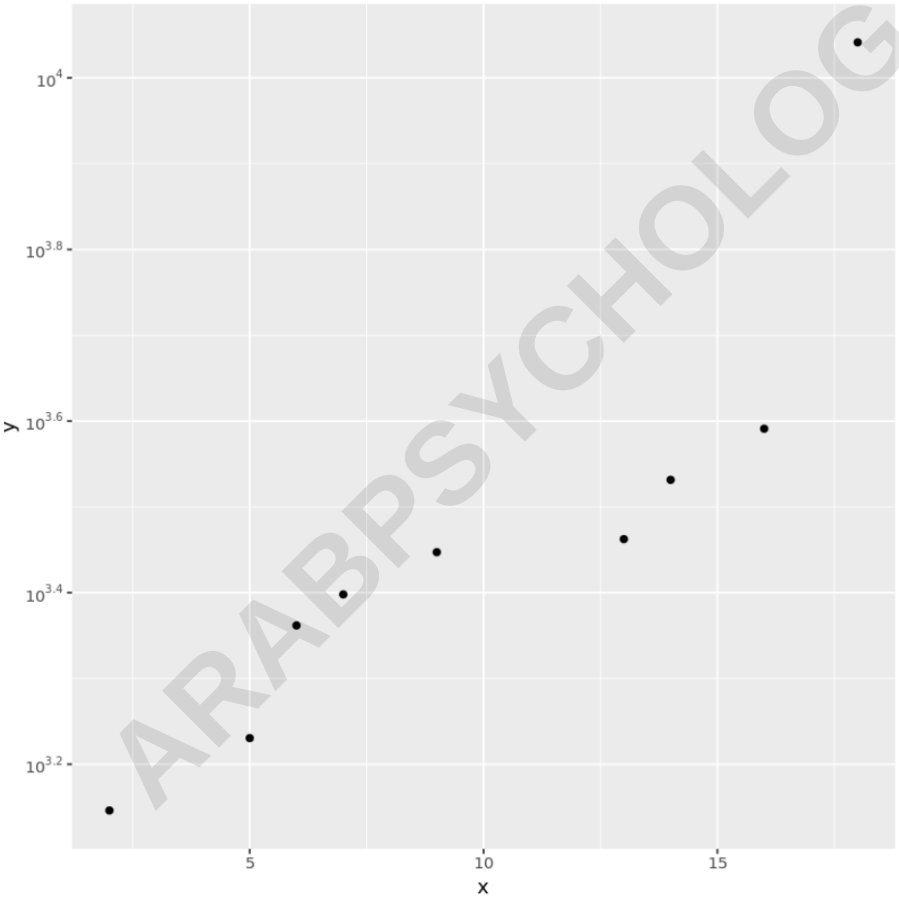
To implement this, we must load both the `ggplot2` library and the `scales` library. Within scale_y_continuous(), the `breaks` argument utilizes `trans_breaks()` to define break points as powers of 10. The `labels` argument then uses `trans_format()` with `math_format()` to render the resulting breaks using mathematical notation (superscripts).

library(ggplot2)

library(scales)

#create data frame
df <- data.frame(x=c(2, 5, 6, 7, 9, 13, 14, 16, 18),
y=c(1400, 1700, 2300, 2500, 2800, 2900, 3400, 3900, 11000))

#create scatterplot with log scale on y-axis and custom labels
ggplot(df, aes(x=x, y=y)) +
geom_point() +
scale_y_continuous(trans='log10',
breaks=trans_breaks('log10', function(x) 10^x),
labels=trans_format('log10', math_format(10^.x)))



Upon reviewing the resulting visualization, it is evident that the Y-axis labels now correctly display exponential notation (10³, 10?, etc.). This contrasts sharply with the plain numeric labels of the previous two examples and represents the standard professional presentation for data displayed on a logarithmic axis. The use of the scales package is indispensable for generating these specialized mathematical formats.

Conclusion: Choosing the Appropriate Transformation Method

Both scale_y_continuous() (or its x-axis counterpart) and coord_trans() successfully generate a log scale in ggplot2. However, the decision of which to use depends on the complexity of the plot and the underlying statistical requirements.

Use scale_y_continuous() (Method 1) when: You need precise control over axis breaks, limits, and labeling (especially custom exponential labels as shown in Example 3). This method transforms the data before any statistical smoothing or summaries are calculated, which is often statistically sounder.
Use coord_trans() (Method 2) when: You require a simple, quick logarithmic view and the plot does not involve complex statistical transformations or customized axis appearance. This method transforms the visual coordinates after the statistics are computed.

For most professional data visualization tasks, leveraging the flexibility of scale_y_continuous() and the advanced formatting capabilities of the scales package remains the definitive best practice for creating clear and accurate logarithmic plots in R.

For further reading and advanced customization techniques in ggplot2, consider the following tutorials:

The Complete Guide to ggplot2 Titles
A Complete Guide to the Best ggplot2 Themes
How to Create Side-by-Side Plots in ggplot2