How to Easily Create Heatmaps in R Using pheatmap()

Authored by

stats writer

November 19, 2025

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

stats writer (2025). *How to Easily Create Heatmaps in R Using pheatmap()*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=97137

The pheatmap package in R provides a robust and highly customizable function, pheatmap(), specifically designed for generating elegant heatmaps. These visualizations are essential tools in fields like bioinformatics, statistics, and data science for exploring large volumes of data. The primary input for this function is a numeric data matrix, where rows often represent observations or features (e.g., genes) and columns represent conditions or samples (e.g., treatments).

A core feature of the pheatmap() function is its automatic implementation of hierarchical clustering. By default, it applies clustering to both rows and columns, arranging similar data points closer together. This rearrangement helps reveal underlying structures, trends, and correlations within the dataset that might otherwise be hidden in raw numerical tables. The resulting visual output is a sophisticated, colored cluster heatmap, providing an immediate understanding of data distribution and relationships.

Beyond simple visualization, pheatmap() offers extensive options for customization. Users can precisely control the visual presentation, including defining the color scale, adjusting the font size of labels, selecting specific clustering methods (e.g., complete, average, single linkage), and incorporating complex row and column annotations. This flexibility ensures that the generated heatmap is not just aesthetically pleasing but also highly informative, tailored exactly to the needs of the statistical analysis being performed.

Setting Up the Environment and Data Preparation

To begin utilizing the powerful features of the **pheatmap** package, we first need to ensure the necessary package is installed and loaded into the R environment. If you do not yet have the **pheatmap** package, you would typically install it using `install.packages("pheatmap")`. Once installed, invoking the library command makes the functions accessible for immediate use. Following environment setup, the next critical step is preparing the data matrix, which must contain only numerical values suitable for distance calculation and visualization.

For demonstration purposes, we will construct a synthetic dataset. This matrix simulates typical biological or experimental data, perhaps representing gene expression levels across various treatments. By using the `set.seed(1)` command, we guarantee that the sequence of pseudo-random numbers generated is identical every time the code is executed. This practice is crucial for maintaining **reproducibility** in scientific and statistical computing, ensuring that others can replicate the exact visualization outcomes presented here.

The code below demonstrates the creation of a 20x5 matrix named `data`. We initialize it with random normal deviates and then introduce distinct blocks of higher values in specific regions to simulate structured biological responses or patterns. These artificial patterns will allow us to clearly observe how the **pheatmap()** function organizes and visualizes clustering tendencies. Finally,

ARABPSYCHOLOGY.COM

meaningful row names (Gene 1 through 20) and column names (T1 through T5) are assigned to enhance interpretability of the final visualization.

```
#make this example reproducible
set.seed(1)

#create matrix with fake data values
data = matrix(rnorm(100), 20, 5)
data = data + 3
data = data + 2
data = data + 4

#add column names and row names
colnames(data) = paste("T", 1:5, sep = "")
rownames(data) = paste("Gene", 1:20, sep = "")

#view matrx
data
```

```
T1 T2 T3 T4 T5
Gene1 2.37354619 3.918977 2.8354764 5.401618 2.431331
Gene2 3.18364332 3.782136 2.7466383 2.960760 2.864821
Gene3 2.16437139 3.074565 3.6969634 3.689739 4.178087
Gene4 4.59528080 1.010648 3.5566632 3.028002 1.476433
Gene5 3.32950777 3.619826 2.3112443 2.256727 3.593946
Gene6 2.17953162 2.943871 2.2925048 3.188792 3.332950
Gene7 3.48742905 2.844204 3.3645820 1.195041 4.063100
Gene8 3.73832471 1.529248 3.7685329 4.465555 2.695816
Gene9 3.57578135 2.521850 2.8876538 3.153253 3.370019
Gene10 2.69461161 3.417942 3.8811077 5.172612 3.267099
Gene11 1.51178117 3.358680 2.3981059 2.475510 1.457480
Gene12 0.38984324 1.897212 1.3879736 1.290054 3.207868
Gene13 -0.62124058 2.387672 2.3411197 2.610726 3.160403
Gene14 -2.21469989 1.946195 0.8706369 1.065902 2.700214
Gene15 1.12493092 4.622940 7.4330237 4.746367 7.586833
Gene16 -0.04493361 5.585005 7.9803999 6.291446 6.558486
Gene17 -0.01619026 5.605710 5.6327785 5.556708 4.723408
Gene18 0.94383621 5.940687 4.9558654 6.001105 5.426735
Gene19 0.82122120 7.100025 6.5697196 6.074341 4.775387
Gene20 0.59390132 6.763176 5.8649454 5.410479 5.526599
```

ARABPSYCHOLOGY.COM

Example 1: Generating the Default Hierarchical Heatmap

The simplest way to use the **pheatmap()** function is by providing only the data matrix as the primary argument. When executed this way, the function automatically implements several default behaviors optimized for data exploration. Specifically, it performs hierarchical clustering on both the rows and the columns using the Euclidean distance metric and the "complete" linkage method, which are standard choices for many analyses.

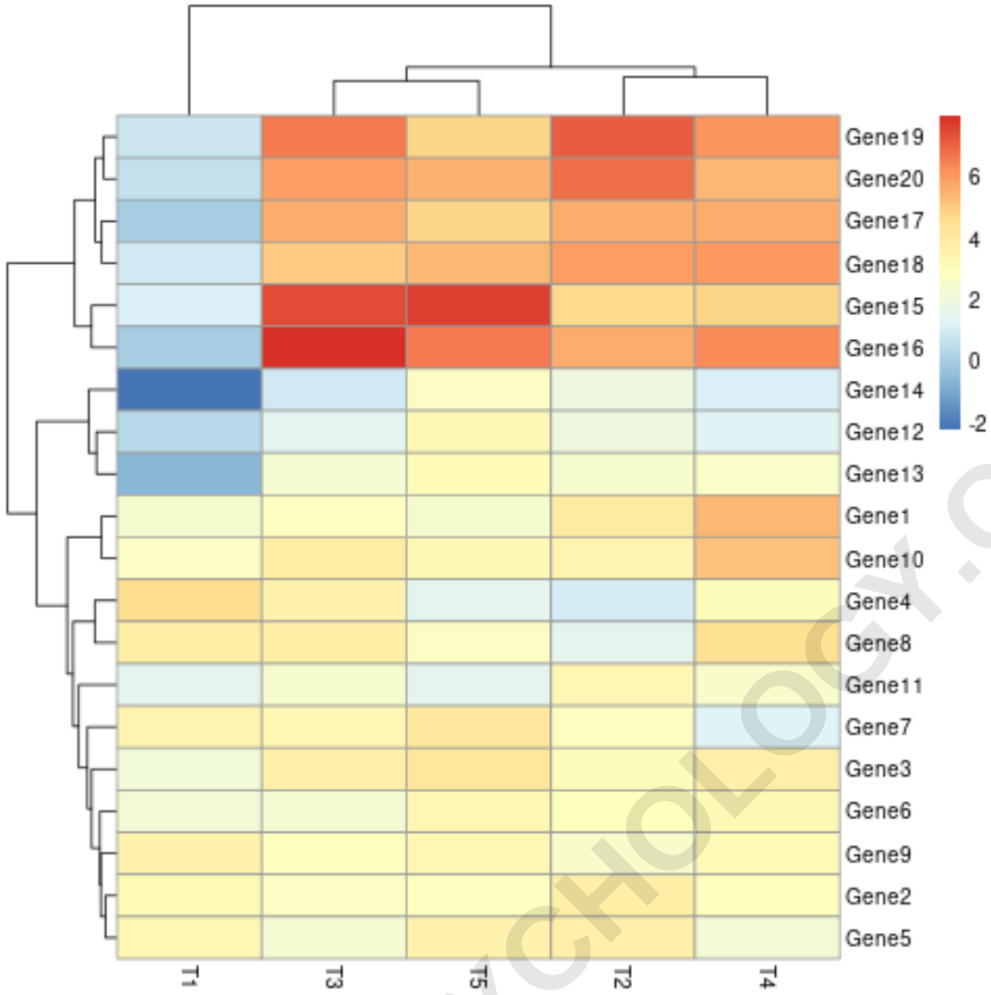
The resulting heatmap uses a default color gradient (typically varying shades of blue, white, and red) to represent the magnitude of the data values: low values are typically displayed in cool colors, while high values are shown in warm colors. Crucially, the dendrograms displayed on the side and top visually represent the results of the clustering process, demonstrating which genes (rows) and conditions (columns) share similar patterns across the dataset.

This basic visualization immediately highlights the block structure we introduced in the data creation step. We can clearly observe the distinct clustering of genes based on their expression profiles across the five treatments (T1-T5), providing an immediate visual interpretation of the data structure before any further customization is applied. Always start with the default plot to gain a fundamental understanding of the inherent patterns.

library(pheatmap)

#create basic heatmap
pheatmap(data)

ARABPSYCHOLOGY.COM



ARABPSYCHOLOGY.COM

Example 2: Displaying Numerical Values within the Cells

While the colors in a heatmap efficiently convey magnitude, there are situations where researchers require the exact numerical values overlaid directly onto the visualization. This is particularly useful when the analyst needs to compare precise values between adjacent cells or confirm specific outlier magnitudes within the data matrix. The **pheatmap()** function allows this through the `display_numbers` argument.

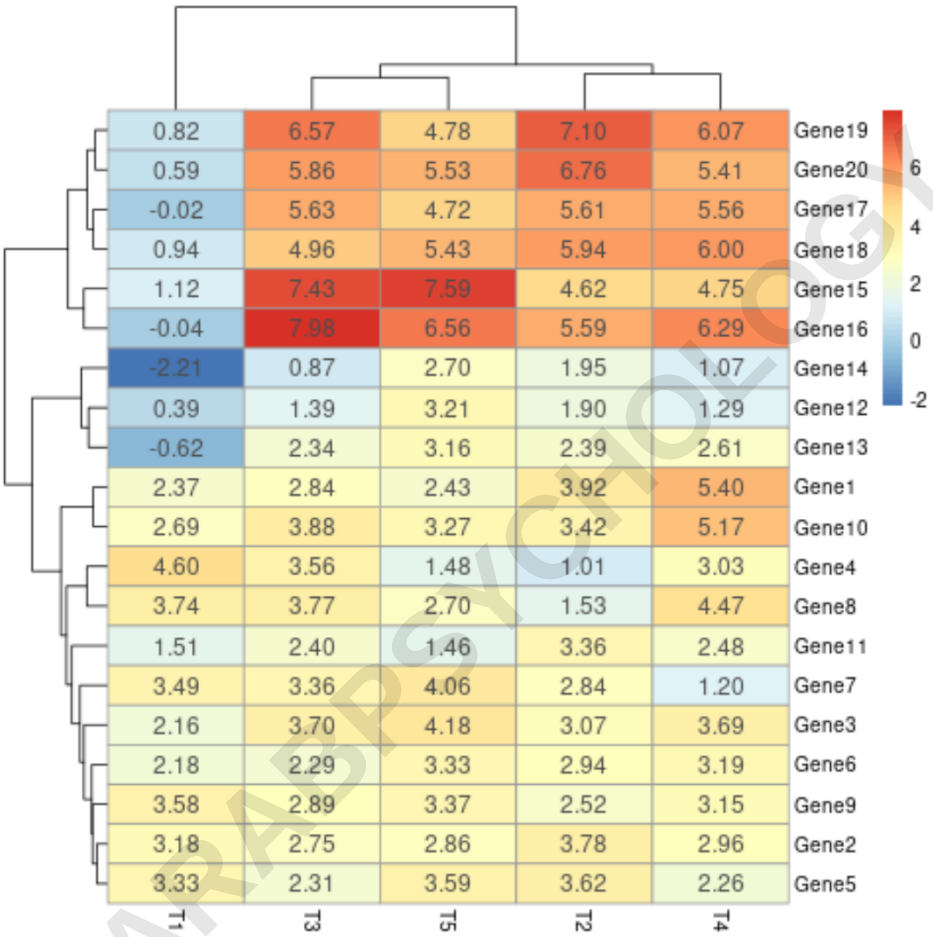
To enable this feature, the `display_numbers` argument must be set to `TRUE`. By default, **pheatmap()** uses a small font size to minimize clutter, often defaulting to 8 points. However, depending on the size of the matrix and the resolution of the output device, this default size may be illegible. We can adjust the font size using the companion argument, `fontsize_number`, to ensure clarity and readability.

In the following example, we set `fontsize_number` to 12. This increases the visibility of the numerical labels, making the specific values within the clustered heatmap immediately accessible.

It is important to remember that using cell labels is often most practical for smaller matrices (like our 20x5 example); visualizing numbers on a large matrix (e.g., 1000x100) would render the plot entirely unusable due to overlapping text.

library(pheatmap)

#create heatmap with numerical labels in cells
pheatmap(data, display_numbers=TRUE, fontsize_number=12)



As noted, the default value for `fontsize_number` is **8**. Adjusting this parameter is key to balancing information density with visual clarity, especially when preparing plots for publication or presentation.

Example 3: Implementing Custom Color Palettes for Enhanced Interpretation

One of the most powerful aspects of **pheatmap()** lies in its control over the color mapping. While the default color scheme is generally effective, scientific and clinical contexts often require specific

color ranges or the use of palette choices optimized for colorblindness or specific publication standards. The `color` argument allows users to pass a customized color vector, overriding the default gradient.

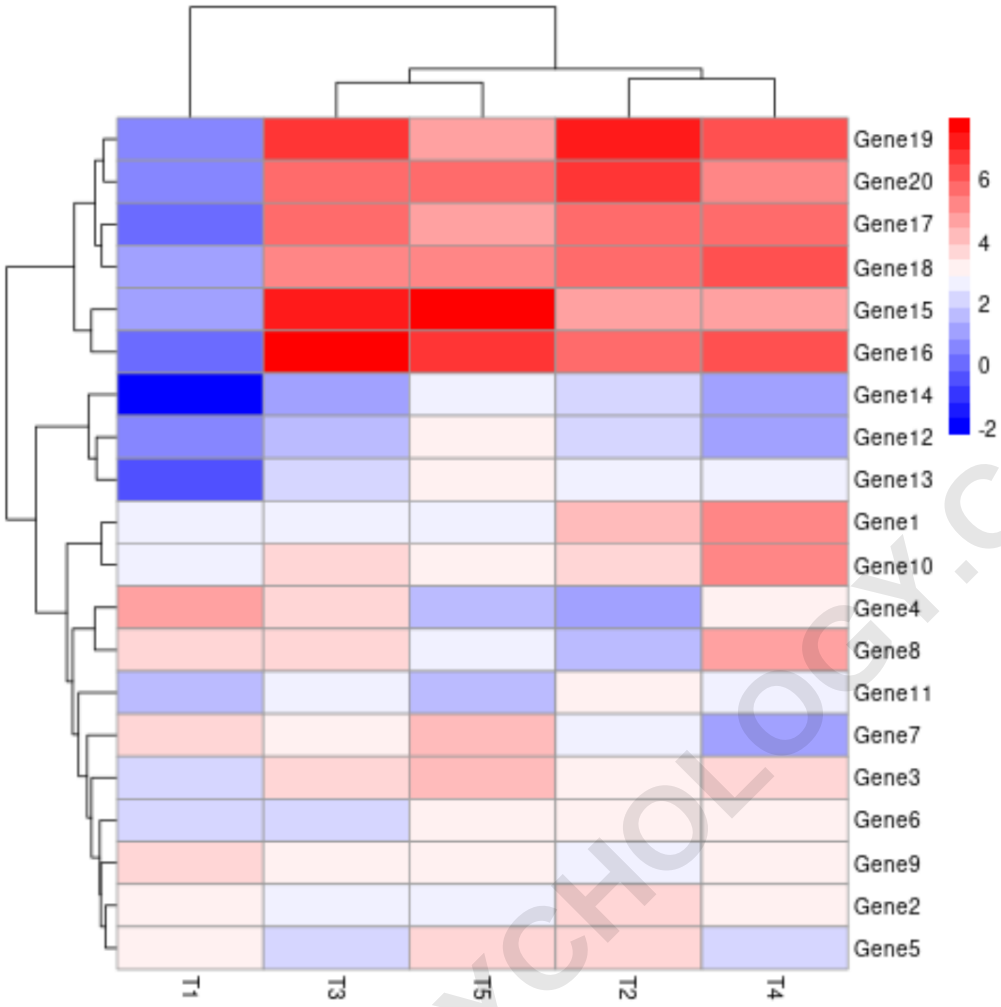
To define a bespoke color gradient, we utilize the R function colorRampPalette, which smoothly interpolates colors between specified anchor points. This function is extremely flexible, taking a vector of desired colors (e.g., "blue", "white", "red") and generating a specified number of intermediate colors to create a continuous gradient. The number in parentheses, such as `(20)` in the example below, defines the total number of colors in the resulting palette, which directly affects the smoothness and resolution of the color scale used in the heatmap.

In this specific implementation, we define a diverging palette where low values are anchored in **blue**, intermediate values are centralized around **white**, and high values saturate towards **red**. This is a common practice for data that has been centered or scaled, making it easy to distinguish between values significantly below average (blue) and those significantly above average (red). This customization greatly enhances visual clarity and data storytelling.

library(pheatmap)

#create heatmap with custom colors
pheatmap(data, color=colorRampPalette(c("blue", "white", "red"))(20))

ARABPSYCHOLOGY.COM



ARABPSYCHOLOGY.COM

As demonstrated, the visualization now clearly displays low expression values in **blue**, median values in **white**, and elevated values prominently in **red**, providing immediate visual confirmation of the engineered patterns in our synthetic data matrix.

Example 4: Controlling and Disabling Hierarchical Clustering

While hierarchical clustering is a default feature designed to optimize the visual identification of patterns, there are scenarios where the automatic row or column reordering is undesirable or unnecessary. For instance, if the data is already ordered logically (e.g., chronological time points, dosage levels), applying clustering might distort the intended relationship or structure.

The **pheatmap()** function provides granular control over the clustering process through two main boolean arguments: `cluster_rows` and `cluster_cols`. Setting either of these arguments to `FALSE` disables the respective clustering operation. When clustering is disabled, the corresponding dendrogram disappears, and the rows or columns remain in the order they were defined in the input data matrix.

The code snippet below demonstrates how to disable clustering specifically for the rows while retaining the column clustering. This approach is frequently used when comparing gene expression data, where the genes (rows) might need to be displayed in a fixed, predefined order (e.g., based on genomic location), but the samples (columns) are still clustered to see sample similarity.

library(pheatmap)

#create heatmap with row clustering disabled
pheatmap(data, cluster_rows=FALSE, color=colorRampPalette(c("blue", "white", "red"))(20))

Furthermore, if finer control over the clustering algorithm is required, pheatmap() accepts arguments like `clustering_distance_rows` (default is "euclidean") and `clustering_method` (default is "complete"). By adjusting these parameters, users can experiment with different mathematical approaches to define similarity, such as using Manhattan distance or Ward's linkage method, tailoring the visualization to the underlying statistical assumptions of the dataset.

Example 5: Incorporating Metadata through Row and Column Annotations

A static heatmap provides visual information about the relationships within the data values themselves, but often, the columns and rows correspond to metadata--experimental factors, phenotypes, or clinical variables--that are essential for interpretation. The **pheatmap()** function excels in integrating this external metadata directly onto the plot using annotation bars.

Annotations are supplied to the function as separate data frames using the arguments `annotation_row` and `annotation_col`. These data frames must have row names that exactly match the corresponding column or row names of the input data matrix. Each column in the annotation data frame represents a categorical or continuous metadata variable that will be displayed as a colored bar adjacent to the heatmap.

Consider a scenario where our treatments (T1-T5) are grouped by "Type" (A or B) and a continuous "Dose" level. We would first construct an annotation data frame defining these variables for each column. The **pheatmap()** function automatically handles the mapping of categorical variables to colors, making it straightforward to visually link the data clusters to external experimental factors. This significantly boosts the descriptive power of the visualization, allowing for conclusions about which factors drive the observed clustering patterns.

library(pheatmap)

# Create column annotation data frame
annotation_col <- data.frame(
Type = factor(rep(c("A", "B"), c(2, 3))),

ARABPSYCHOLOGY.COM

```
Dose = c(10, 20, 5, 15, 25),
row.names = colnames(data)
)

# Create row annotation data frame (e.g., classifying genes)
annotation_row <- data.frame(
Pathway = factor(rep(c("Metabolic", "Signaling"), c(10, 10))),
row.names = rownames(data)
)

# Plotting with annotations
pheatmap(data,
annotation_col = annotation_col,
annotation_row = annotation_row,
color=colorRampPalette(c("blue", "white", "red"))(20)
)
```

The resulting plot, while not shown here, would feature colored bars along the top edge corresponding to Type and Dose for T1-T5, and bars along the left edge indicating the Pathway for Gene1-Gene20. This integration of metadata transforms the heatmap from a raw data visualization into a powerful, context-rich analytical summary.

Example 6: Standardizing Data using Scaling and Centering

In many analytical contexts, particularly when dealing with biological data or variables measured on vastly different scales, the direct use of raw values in a heatmap can be misleading. Variables with inherently larger magnitudes might dominate the color scale, obscuring subtle variations in variables with smaller ranges. To address this, it is standard practice to normalize the data before visualization, typically by scaling and centering.

The **pheatmap()** function facilitates this essential preprocessing step internally using the `scale` argument. When `scale = "row"` is specified, each row (e.g., each gene) is scaled such that its mean is zero and its standard deviation is one. This transformation allows for the comparison of expression patterns across samples, focusing on how much a specific gene is up-regulated or down-regulated relative to its own average behavior, irrespective of its overall expression level compared to other genes.

Alternatively, setting `scale = "column"` performs the standardization across the columns. This is useful when comparing the overall characteristics of samples across different experiments. If `scale = "none"` (the default), the raw values are used directly. Proper scaling ensures that the

ARABPSYCHOLOGY.COM

heatmap visualization accurately reflects relative differences and relationships rather than absolute magnitude bias.

```
library(pheatmap)

# Create heatmap scaled by row (Z-score normalization)
pheatmap(data,
scale = "row",
color=colorRampPalette(c("blue", "white", "red"))(20)
)
```

By applying row scaling, the visual output changes significantly. The colors now represent **Z-scores**: red cells indicate values that are significantly higher than the row mean, blue cells indicate values significantly lower, and white cells represent values close to the mean for that specific row. This transformation dramatically improves pattern recognition within the rows, making the visualization far more robust for high-throughput data analysis.

Summary of pheatmap Customization Capabilities

The pheatmap package stands out as an indispensable tool within the R ecosystem for generating highly informative and visually compelling heatmaps. Unlike simpler plotting functions, **pheatmap()** integrates complex statistical methods, such as hierarchical clustering and data standardization, directly into the visualization workflow.

From controlling the color scheme using the colorRampPalette function to precisely managing which dimensions are clustered or incorporating detailed metadata annotations, the versatility of **pheatmap()** ensures that analysts can produce publication-quality graphics tailored to almost any analytic requirement. Mastering its core arguments--such as `scale`, `display_numbers`, `cluster_rows`, and `annotation_col`--allows for deep data exploration and effective communication of complex patterns discovered within the numerical data matrix.

We have demonstrated how to move beyond the default visualization to create a plot that is both numerically explicit and contextually rich, thereby leveraging the full power of **pheatmap()** for robust data analysis in R.

ARABPSYCHOLOGY.COM