Learning to Create Heatmaps in R with pheatmap()

Authored by

Mohammed looti

November 15, 2025

Introduction to Heatmaps and the pheatmap Package in R

The effective communication of complex scientific and analytical insights relies heavily upon powerful **data visualization** techniques. Among the most versatile methods available, **heatmaps** stand out as indispensable graphical tools, particularly well-suited for summarizing and exploring large, matrix-like datasets. A heatmap fundamentally transforms numerical data into a visual grid where the magnitude of each value is intuitively encoded by color intensity. This approach allows analysts to rapidly identify structural patterns, reveal hidden correlations, and spot outliers across extensive rows and columns simultaneously. Due to their high information density and intuitive nature, heatmaps are extensively utilized across diverse scientific disciplines, including **bioinformatics**, where they frequently visualize differential **gene expression**, as well as in finance and social sciences for advanced pattern recognition.

Within the **R** programming environment, which is internationally recognized for its superior statistical capabilities, the dedicated **pheatmap** package provides the definitive solution for generating high-quality heatmaps. The primary function, also named `pheatmap()`, offers a robust blend of flexibility and functionality, enabling users to create visualizations that are both aesthetically refined and analytically rigorous. A key advantage of this package is its capacity to seamlessly integrate advanced features such as **hierarchical clustering**. This automated process groups data entities based on computed similarity, thereby revealing underlying structures and relationships that might otherwise be obscured within raw data tables.

This comprehensive guide is meticulously structured to provide a detailed introduction to the core functionalities of the `pheatmap()` function. We will begin by establishing the fundamental principles of data preparation necessary for this specific visualization method. Following the data setup, we will walk through the steps required to generate a basic heatmap using default settings, and then progress toward more advanced customization techniques. These enhancements will include integrating detailed numerical cell labels and applying custom **color palettes**. By diligently mastering the concepts presented here, you will gain the proficiency required to generate informative, visually compelling, and analytically sound heatmaps tailored precisely to your unique research and reporting requirements.

Structuring and Preparing Data for Heatmap Generation

The generation of any meaningful heatmap is predicated on the prerequisite that the input data is correctly organized into a structure that the `pheatmap()` function can effectively process. Specifically, the data must be configured as a numerical **matrix**. In typical analytical scenarios, rows within this matrix represent individual observations (such as distinct genes or samples), and columns represent variables or conditions (such as experimental treatments or sequential time points). For the pedagogical purposes of this tutorial, rather than depending on an external, potentially complex dataset, we will generate a synthetic dataset. This standard statistical practice allows us to precisely control the characteristics of the data and deliberately introduce specific patterns, ensuring that the resulting heatmaps clearly illustrate the function's capabilities and features.

The following R code snippet is dedicated to constructing a synthetic **matrix** consisting of 20 rows and 5 columns. This setup mimics a common structure found in high-throughput experimental data, such as expression profiling studies. We have carefully engineered distinct blocks of high and low values within this matrix to guarantee that the visualization yields discernible and interpretable clusters when processed. To ensure the complete reproducibility of our analysis--a critical standard in all **R** programming and statistical practice--we initialize the random number generator using the `set.seed(1)` function. This guarantees that every time the code is executed, the identical sequence of random numbers and, consequently, the exact same data matrix is produced.

```
#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

When examining the code block, the function **`set.seed(1)`** anchors our process, ensuring complete consistency upon every execution. We generate 100 random numbers drawn from a standard normal distribution using **`rnorm(100)`**, which are subsequently reshaped into the required 20x5 **matrix** structure. The subsequent array manipulations are critical; they intentionally introduce differential values across specific subsets of rows and columns, successfully creating the anticipated clustering structures. Finally, we assign clear, descriptive labels--such as "Gene1" through "Gene20" for the rows and "T1" through "T5" for the columns. These labels are fundamental for clearly interpreting the resulting heatmap axes and effectively communicating the experimental findings to an audience.

Generating a Foundational Heatmap with Default Settings

The most direct and straightforward application of the `pheatmap()` function necessitates only the numerical data **matrix** as its primary input argument. By default, `pheatmap` is expertly engineered to simultaneously perform **hierarchical clustering** on both the rows and the columns of the supplied data. This automatic clustering capability is a highly valuable feature for exploratory **data analysis**, as it systematically reorders the data entities to place similar patterns or values adjacent to one another. This immediate reordering instantly exposes hidden correlations and inherent structures within the dataset without requiring any prior manual intervention or specification from the user.

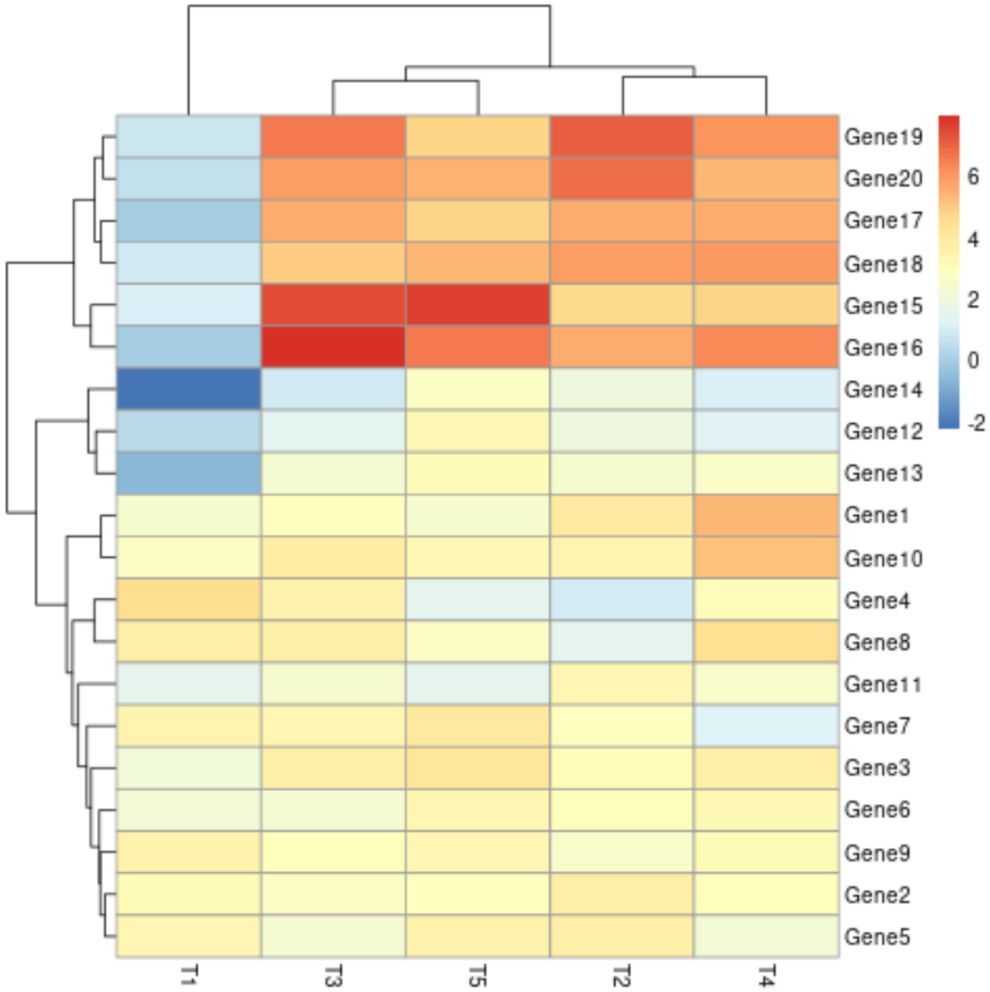
Beyond reordering the matrix, the function automatically generates a **dendrogram** for both the row and column dimensions. These dendrograms graphically summarize the outcomes of the clustering analysis, providing a visual representation of the proximity or similarity between individual rows and columns. In a dendrogram, shorter connecting branches signify a greater degree of similarity between the merged entities. Coupled with the clustering output, a clear color scale is also generated by default, which maps the numerical values within the matrix to a continuous color gradient. This scale is vital for translating the quantitative data into a qualitative visual assessment, allowing for the quick distinction between high-value and low-value regions of the map.

To generate this foundational heatmap using the prepared `data` matrix, the user simply needs to execute the following concise R commands after ensuring the requisite library is loaded into the session:

library(pheatmap)

#create basic heatmap
pheatmap(data)

As visually confirmed by the image provided below, the resulting heatmap immediately offers a comprehensive visual summary of the synthetic data. The clustered structure is clearly evident, accurately reflecting the patterns we deliberately introduced into the data matrix. The default **color scheme**, which frequently employs a diverging structure (typically blue for low values, white for mid-range, and red for high values), effectively highlights regions of extremes. This initial, robust visualization represents an essential starting point for any in-depth exploratory data analysis effort.



Enhancing Clarity: Incorporating Numerical Cell Labels

While the strategic use of color gradients in a heatmap is excellent for providing a macro-level overview of global trends and underlying patterns, specific analytical situations often demand access to the precise numerical value contained within each cell for detailed verification or presentation purposes. For smaller datasets, or when the visualization is intended to serve a highly quantitative reporting function, overlaying these exact numerical values directly onto the heatmap cells becomes highly desirable. The `pheatmap()` function is specifically designed to accommodate this need elegantly, offering a straightforward parameter to toggle this crucial feature on or off.

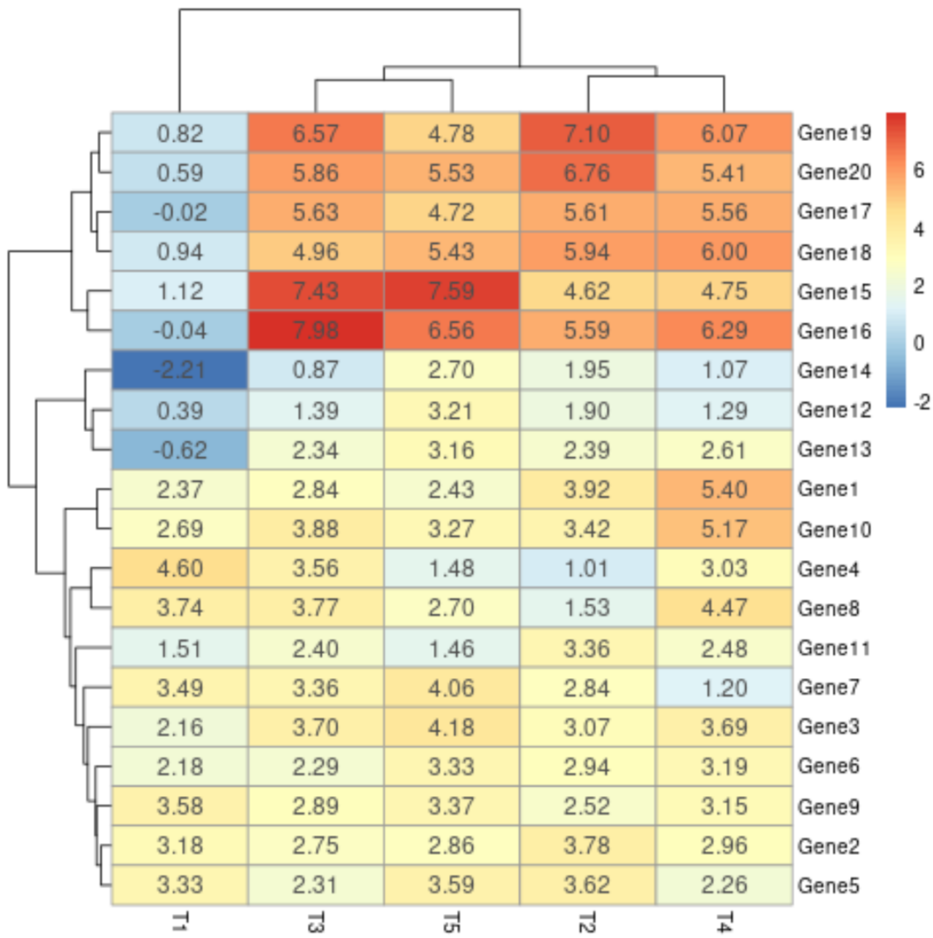
To activate the display of numerical values within the grid cells, the user must set the `display_numbers` argument to its Boolean state of `TRUE`. This command instantly overlays the data points onto the visual field. Furthermore, to safeguard the legibility and aesthetic quality of the visualization--a particular concern when dealing with plots destined for various output sizes--the `fontsize_number` argument provides granular control over the size of the overlaid text. This dual control mechanism ensures that the numerical details are presented clearly without the common risk of overcrowding the plot or rendering the text too small to read, successfully balancing visual appeal with informational precision.

The following R code provides a clear demonstration of how to implement these arguments, activating the display of numbers and setting a slightly larger font size than the default to ensure maximum readability across different platforms:

```
library(pheatmap)

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

The resulting **heatmap**, presented below, clearly showcases the impact of these parameter settings. With `fontsize_number` set to 12, each cell now contains its corresponding numerical value, facilitating easy cross-referencing between the visual intensity (color) and the exact data magnitude. This feature proves invaluable when the analytical goal is to pinpoint specific data points or when the matrix size remains manageable. It is important to recall that the default `fontsize_number` is typically **8**; consequently, strategic adjustment of this parameter is key to optimizing the visualization for specific output formats, such as high-resolution publication figures or projection-ready presentation slides.



Customizing Color Palettes for Optimal Interpretability

The strategic selection of a **color scheme** represents arguably the single most critical decision in effective **data visualization**, as the chosen palette fundamentally dictates how quickly and accurately viewers can perceive patterns, magnitudes, and differences within the data. While `pheatmap()` does provide a robust and functionally sound default color gradient, custom manipulation of the colors allows the user to align the visualization precisely with the underlying data characteristics or adhere to recognized standards within a particular scientific domain. This customization ensures optimal interpretability of the continuous spectrum of values represented in the heatmap.

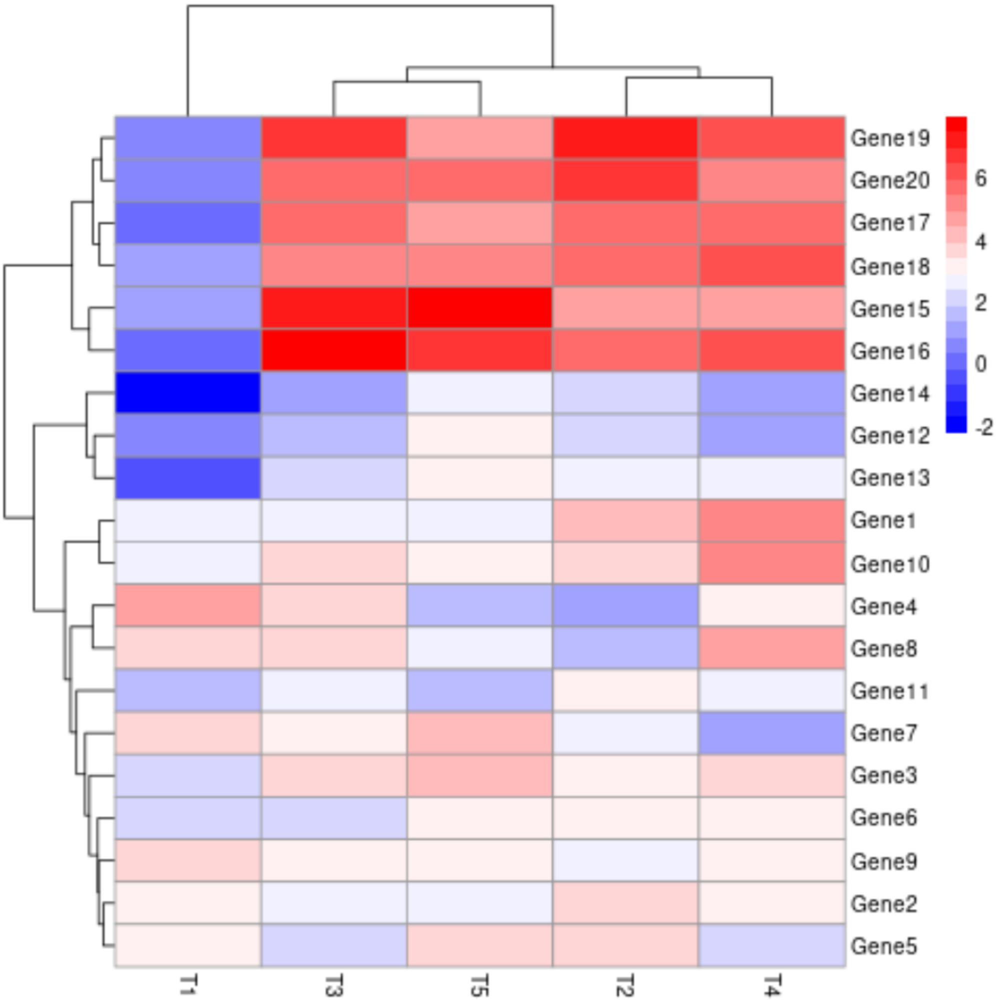
Custom colors are implemented via the `color` argument, which expects a vector of colors defining the desired gradient. The recommended and most effective method for generating this smooth transition is by utilizing the **`colorRampPalette()`** function, which is readily available within R's standard `grDevices` package. This powerful function allows the user to specify a set of anchor colors, between which it interpolates a continuous range of intermediate hues. For instance, datasets that inherently possess a natural midpoint--such as deviations from an average, a control group, or a zero point--are best visualized using a diverging color palette (e.g., transitioning from one color, through white, to a contrasting color), which effectively emphasizes deviations in two distinct directions.

To powerfully illustrate this customization capability, we will redefine our heatmap to employ a classic and scientifically favored diverging palette: low values will be represented by **blue**, the median values will be marked by **white**, and high values will transition towards a saturated **red**. This structure is exceptionally effective for highlighting extremes and providing immediate visual context regarding the magnitude and direction of value shifts:

```
library(pheatmap)

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

The resulting visualization, depicted below, confirms how dramatically the custom **color scheme** enhances interpretability. The distinct **blue** and **red** extremes immediately draw the viewer's attention to the lowest and highest value regions, respectively, while the smooth transition through **white** clearly delineates the data's central tendency. This visual strategy significantly improves the ability to discern nuanced patterns and magnitudes within the data at a single glance, thereby maximizing the overall effectiveness and communicative power of the **heatmap**.



Advanced Customization Options for Fine-Tuning Heatmaps

The inherent power of the `pheatmap()` function extends far beyond simple color and label adjustments, providing a rich ecosystem of parameters specifically designed to fine-tune visualizations for complex analytical needs. These advanced options allow for precise control over fundamental aspects of the heatmap, ranging from the mathematical methodology used for **hierarchical clustering** to minute details of visual presentation, such as scale normalization and border appearance. Mastering these parameters is essential for transforming a basic graphical output into a sophisticated and precise analytical instrument capable of conveying detailed, high-level insights.

The following list summarizes some of the most critical and frequently utilized advanced customization options available within `pheatmap()`, along with their intended analytical utility and purpose:

`scale`: This pivotal parameter governs how data values are normalized prior to visualization.

Options include `"row"` (standardizing values within each row to have a mean of zero and standard deviation of one), `"column"` (applying the same scaling to each column), or `"none"` (the default). Scaling is often indispensable when comparing rows or columns whose inherent value ranges vary drastically, ensuring that relative changes, rather than absolute magnitudes, drive the clustering process.

`cluster_rows` and **`cluster_cols`**: These Boolean arguments (`TRUE` or `FALSE`) allow the user to independently enable or disable the automated clustering process for rows and columns. Disabling clustering is frequently necessary when the rows or columns must adhere to a predefined, biologically, or chronologically significant order, thus overriding the algorithmic grouping.

`clustering_distance_rows` and **`clustering_distance_cols`**: These parameters specify the metric used to calculate the distance or dissimilarity between data points during the hierarchical clustering calculation. Standard options include `"euclidean"` (the default), `"maximum"`, `"manhattan"`, `"canberra"`, `"binary"`, or `"minkowski"`. The careful choice of distance metric profoundly influences the resulting cluster structure and the visual configuration of the **dendrogram**.

`clustering_method`: This defines the agglomeration method--the specific mathematical rule dictating how nascent clusters are merged at each step of the hierarchy. Choices such as `"ward.D"`, `"complete"`, `"average"`, and `"single"` are available. Each method operates under different statistical assumptions regarding the distance between clusters, thus altering the final grouping outcome.

`show_rownames` and **`show_colnames`**: These Boolean arguments control the visibility of row and column labels. Hiding labels is particularly useful for very large matrices where the sheer number of labels would render the plot illegible or excessively cluttered.

`main`: This simple yet crucial parameter allows for the addition of a prominent main title to the heatmap, which is essential for providing immediate context and a succinct description of the visualization's content.

`border_color`: Used to set the color of the borders surrounding each cell. Setting this parameter to `NA` will remove the borders entirely, a common technique employed to achieve a smoother, more continuous visual effect, especially when visualizing continuously varying data.

Conclusion and Recommended Best Practices

The `pheatmap()` function within **R** represents a cornerstone tool for researchers and analysts engaged in **statistical computing** and advanced **data visualization**. Its robust capacity to rapidly generate insightful and visually striking heatmaps, combined with an extensive suite of customization options, solidifies its position as the preferred choice for exploring, summarizing, and reporting complex multivariate datasets. Whether the task involves a quick exploratory plot or a highly detailed figure destined for publication, `pheatmap()` offers the necessary flexibility and analytical depth to meet the demand.

To maximize the effectiveness and ensure the interpretability of your heatmaps, adherence to the following best practices is strongly recommended for all users:

Contextual Scaling: The decision to apply row, column, or no scaling must be grounded entirely in the analytical question being addressed. Use scaling judiciously when the objective is to compare relative changes (e.g., fold changes across samples) rather than absolute measurements.

Judicious Color Selection: Always select **color palettes** that are perceptually appropriate for your data type. Diverging palettes are ideal for data centered around a zero point, while sequential palettes are best suited for non-negative, ordered data. Furthermore, actively ensure your chosen palette is accessible to individuals with common forms of color blindness.

Validate Clustering Results: Always recognize that the results of **hierarchical clustering** are highly sensitive to the specific distance metric and agglomeration method chosen. These fundamental methodological choices must be explicitly justified by the underlying scientific context and the specific nature of the data being analyzed.

Prioritize Clarity: Ensure that all visual elements, including the main title, axis labels, and legend, are highly legible and informative. While cell labels can be exceptionally useful for small matrices, they should be consciously omitted if they introduce visual clutter to the overall plot.

Provide Interpretation: A heatmap should never stand in isolation. Always accompany the visualization with sufficient explanatory text that clearly defines the meaning of the rows, columns, and the color scale, effectively guiding the audience toward the key insights the visualization is intended to convey.

By faithfully applying these principles and leveraging the robust capabilities embedded within the `pheatmap()` package, you will be empowered to generate compelling visualizations that effectively uncover and communicate the intricate patterns hidden within your data, ultimately leading to a deeper understanding and more robust, data-driven decision-making processes.

Additional Resources for Continued Learning

To further refine your proficiency in R programming and sophisticated data visualization techniques, we strongly encourage continued exploration of the official documentation and advanced tutorials available online. These resources will provide the foundational knowledge necessary to confidently tackle complex analytical challenges and expand your repertoire of visualization skills within the powerful R environment.