

How to Make Heatmaps with Seaborn (With Examples)

Authored by
stats writer

December 12, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Make Heatmaps with Seaborn (With Examples)*.

PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=107222>

A **heatmap** is a powerful graphical representation technique used in **data visualization**. It employs different shades of color intensity to represent the magnitude of data values within a two-dimensional matrix. This method allows viewers to quickly identify patterns, correlations, and outliers in large datasets, making complex information instantly accessible and interpretable.

This comprehensive tutorial is designed to guide you through the process of creating effective and customized heatmaps using **Seaborn**, a high-level statistical **Python** visualization library. Seaborn is built upon **Matplotlib** and excels at providing aesthetically pleasing and informative statistical graphics. We will utilize the built-in "flights" dataset to demonstrate the key functionalities of the `heatmap()` function.

Preparing the Data for Heatmap Visualization

Before we can generate our visualization, the data must be correctly structured. Heatmaps typically require data to be in a matrix format where the rows and columns represent categorical variables, and the cell values represent the measurement (or frequency) associated with that intersection. We begin by importing the necessary libraries and loading the sample data provided by **Seaborn**.

Import Seaborn (and Matplotlib, which is often implicitly required for figure control)

```
import seaborn as sns
```

```
# Load the standard "flights" dataset included with Seaborn
```

```
data = sns.load_dataset("flights")
```

```
# Use the .pivot function to restructure the data into a matrix suitable for a heatmap.
```

```
data = data.pivot("month", "year", "passengers")
```

```
# View the resulting structure (first five rows)
```

```
data.head()
```

The use of the **pivot table** function is essential here. The original 'flights' data is structured in a long format, which is not suitable for a heatmap visualization that requires a grid layout. The `pivot()` method transforms the data into the necessary wide format, assigning 'month' to the index (rows), 'year' to the columns, and 'passengers' as the cell values. This prepares the two-dimensional array required by the `heatmap()` function.

year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
month												
January	112	115	145	171	196	204	242	284	315	340	360	417
February	118	126	150	180	196	188	233	277	301	318	342	391
March	132	141	178	193	236	235	267	317	356	362	406	419
April	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472

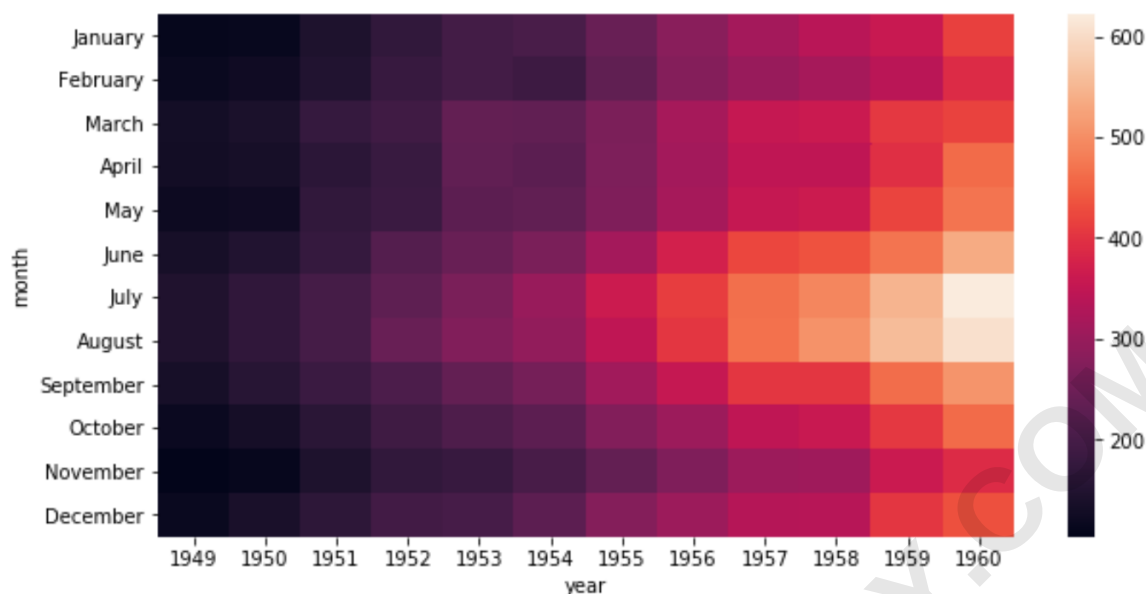
Generating the Basic Heatmap

Once the data is correctly structured as a matrix, generating the initial **heatmap** is exceptionally straightforward using the `sns.heatmap()` function. This function takes the restructured DataFrame as its primary argument and automatically determines the color intensity scale based on the range of passenger counts present in the data.

The core syntax for creating the most basic version of this visualization requires only a single line of code, demonstrating the efficiency and high-level abstraction provided by the **Seaborn** library.

`sns.heatmap(data)`

The resulting graphical output displays the year on the x-axis and the month on the y-axis. The intensity of the color in each cell represents the number of passengers traveling during that particular year-month combination. By default, lighter colors signify higher passenger counts, immediately highlighting peak travel seasons and annual trends. This basic visualization provides immediate insight into seasonality and long-term growth trends in air travel.



Controlling Figure Aesthetics: Adjusting Size

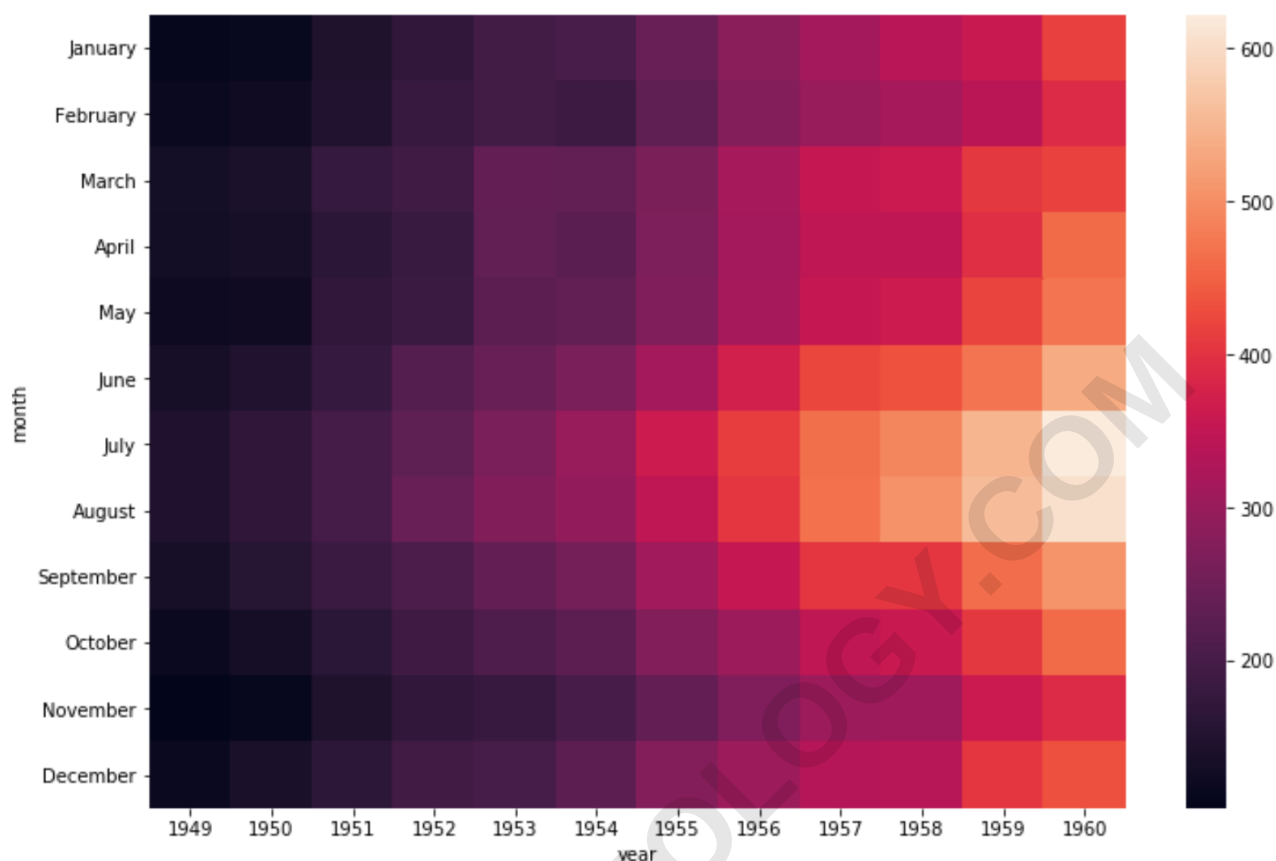
While the default **Seaborn** output is functionally correct, it often needs size adjustment for detailed reports or presentations. A crucial aspect of effective **data visualization** is ensuring the figure dimensions support readability, especially when dealing with complex datasets. Since Seaborn leverages the underlying capabilities of **Matplotlib**, we must use Matplotlib's `pyplot` module to define the overall figure size before plotting the heatmap.

The `plt.figure()` function, utilizing the `figsize` argument, allows us to set the width and height of the canvas in inches. Increasing these dimensions ensures that the axis labels are not cramped and that the individual cells of the heatmap are large enough to visually differentiate the color gradients effectively.

```
# Import Matplotlib's pyplot module
import matplotlib.pyplot as plt
# Define the figure size (12 inches wide, 8 inches tall)
plt.figure(figsize = (12,8))

# Generate the heatmap within the resized figure
sns.heatmap(data)
```

The application of `figsize = (12, 8)` drastically improves the layout, providing more whitespace around the plot elements and clarifying the axis labels. This step is indispensable for high-quality visual outputs intended for formal review or publication.



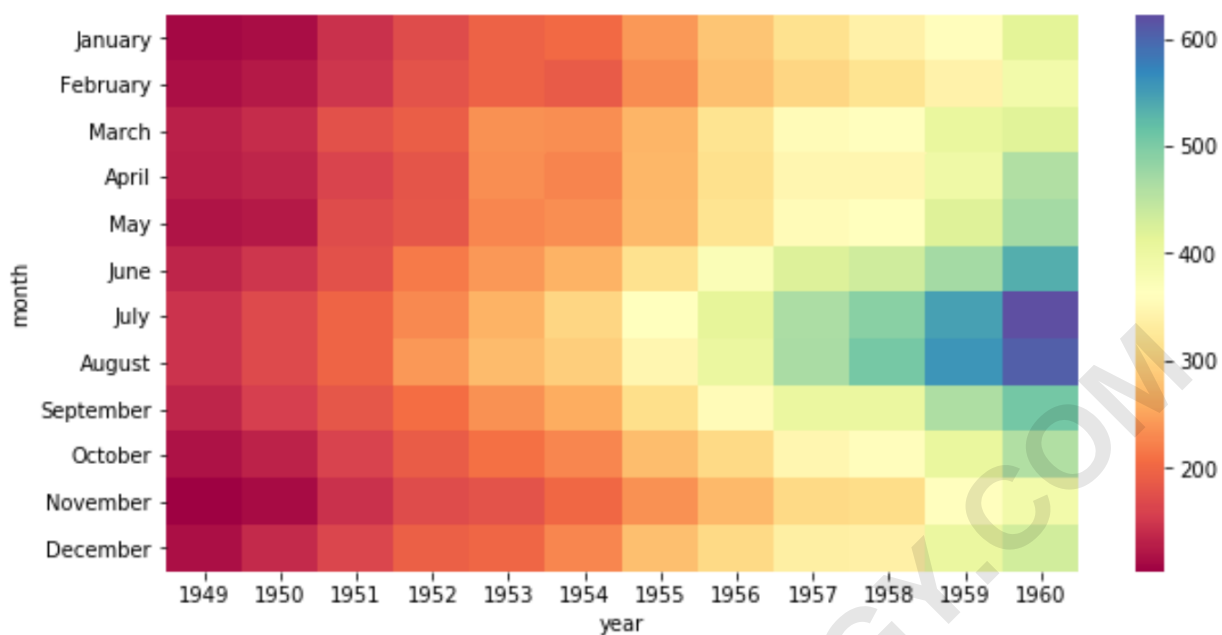
Customizing Color Schemes with the `cmap` Argument

The choice of colormap (`cmap`) fundamentally influences how data patterns are perceived in a **heatmap**. Different color palettes are suited for different data types--sequential, diverging, or qualitative. For our sequential data (passenger counts ranging from low to high), while the default is acceptable, customizing the `cmap` allows us to achieve specific visual effects or ensure compliance with color standards.

The `cmap` argument accepts a string defining the desired colormap. We can experiment with various built-in Matplotlib/Seaborn colormaps. For instance, using "Spectral" provides a high-contrast diverging palette which emphasizes the difference between low and high values using a broad spectrum of colors.

```
sns.heatmap(data, cmap="Spectral")
```

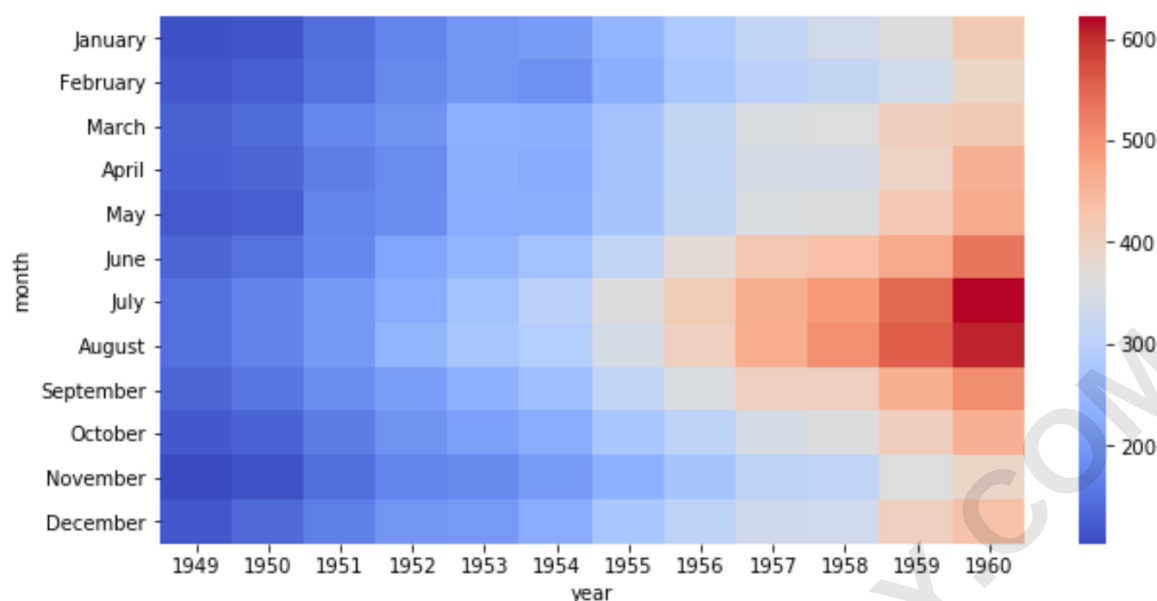
The resulting heatmap utilizing the "Spectral" colormap displays a dramatic shift in color relationships, offering a different interpretation of the data density compared to the default.



A second example involves using "coolwarm," a quintessential diverging colormap often used when data centers around a meaningful zero or average point. Even when applied to purely sequential data, "coolwarm" can effectively delineate high extremes (warm colors) and low extremes (cool colors).

`sns.heatmap(data, cmap="coolwarm")`

This variation highlights the versatility of the `cmap` parameter. Users are encouraged to refer to the official [Matplotlib](#) documentation for a comprehensive gallery of available colormaps to ensure the chosen palette is suitable for the data type and the intended audience.



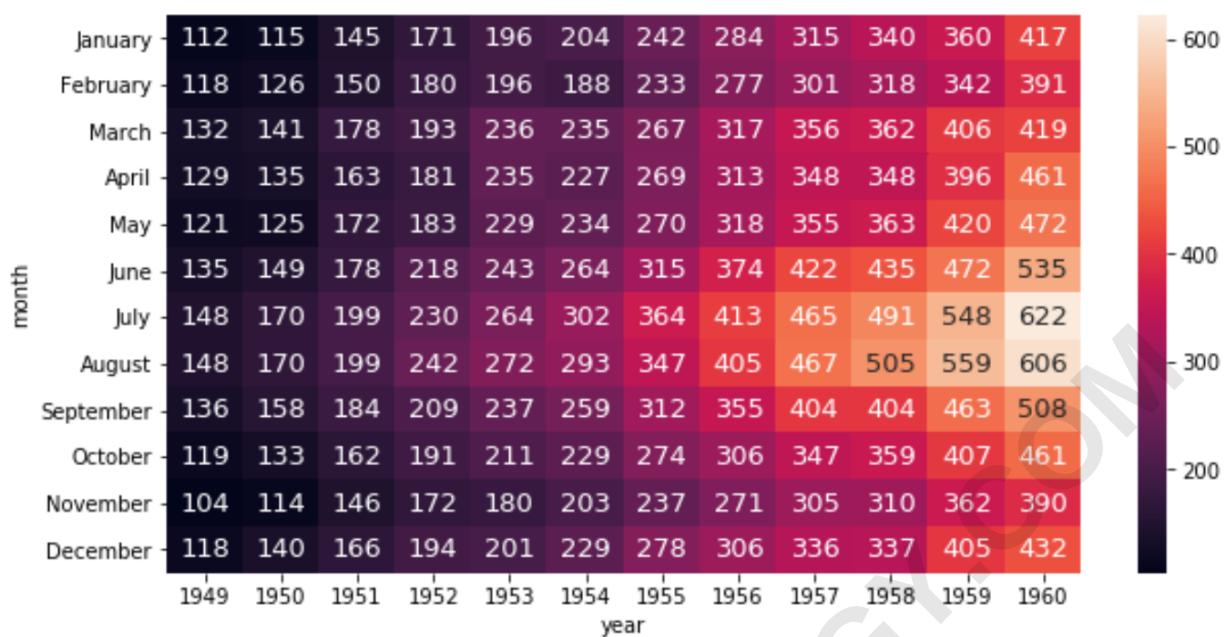
Enhancing Readability: Annotating Cells with Data Values

While color gradients provide an intuitive sense of magnitude, providing the exact numerical value for each cell is often necessary for precise analysis. **Seaborn** facilitates this through the `annot` argument. Setting `annot=True` instructs the function to display the data values directly inside the cells.

For numerical accuracy, especially when dealing with integers like passenger counts, the `fmt` argument is used to apply formatting. Using `fmt="d"` ensures that the values are displayed as standard integers (decimal). Additionally, to improve the visual prominence of these numbers, we can customize the text properties using `annot_kws`, which takes a dictionary of arguments passed to the text rendering engine. We specify `{"size": 13}` to increase the font size.

```
sns.heatmap(data, annot=True, fmt="d", annot_kws={"size":13})
```

The inclusion of annotations transforms the **heatmap** from a purely exploratory visualization tool into a detailed data table overlaid with color coding. This combination is highly effective, allowing for both rapid identification of high-value regions and precise data retrieval.



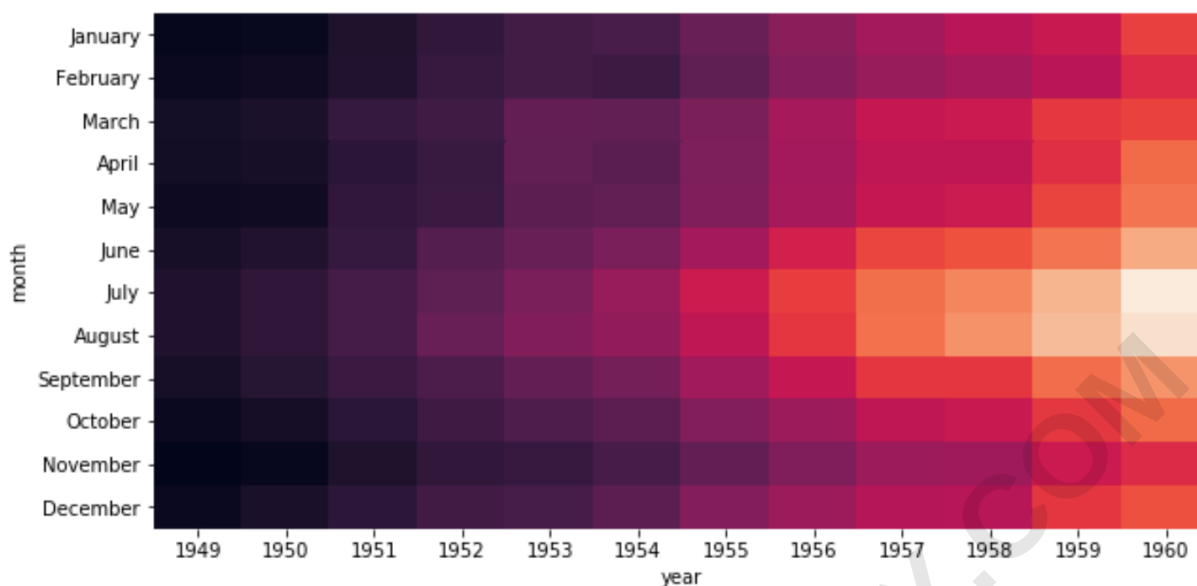
Refining the Display: Modifying or Removing the Colorbar

The colorbar automatically generated by **Seaborn** serves as a key to interpreting the color-to-value mapping. However, in instances where the figure space is constrained, or if the cells are fully annotated, the colorbar may become redundant. The `cbar` argument provides the control necessary to manage this element of the visualization.

By setting the `cbar` argument to `False`, we instruct the `heatmap()` function to suppress the generation and display of the colorbar, resulting in a cleaner, more compact visual output.

`sns.heatmap(data, cbar=False)`

Removing the colorbar, as demonstrated below, prioritizes the visibility of the data matrix itself. This is often desirable when the heatmap is part of a series of plots or when the exact range of values is already known or labeled within the plot itself, such as through annotations.



Conclusion and Further Exploration

This tutorial has demonstrated the fundamental steps required to create, customize, and refine heatmaps using the **Python Seaborn** library, utilizing both Seaborn's specialized functions and **Matplotlib**'s general figure controls. Key parameters such as `cmap`, `annot`, `fmt`, and `cbar` allow for meticulous control over the final visualization.

Mastery of heatmaps is invaluable for statistical analysis, providing a clear visual language for correlation matrices, feature importance scores, and time-series data. To further advance your skills, consider exploring advanced features such as masking specific data points, changing the orientation of tick labels, or implementing hierarchical clustering within your heatmaps.

You can find more detailed examples and documentation on advanced statistical plotting techniques by consulting the official **Seaborn** tutorials.