How to Add a Table to Seaborn Plot (With Example)

Authored by

stats writer

November 21, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Add a Table to Seaborn Plot (With Example)*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=99165

ARABPSYCHOLOGY.COM

Integrating tabular data directly into statistical visualizations is a powerful technique in data visualization. While Seaborn excels at creating aesthetically pleasing and informative statistical graphics, it often relies on its underlying library, Matplotlib, for fine-grained control and complex additions like embedding data tables. The solution lies in leveraging the `matplotlib.table` function. This versatile function accepts an array of values, transforming them into a formatted table complete with designated row and column labels, which can then be overlaid or positioned adjacent to the plot. For instance, when visualizing complex datasets, adding the raw data table--whether it complements a Seaborn `Heatmap` or a `scatterplot`--significantly enhances the clarity and interpretability of the graphic. The process involves generating the visualization (e.g., using `seaborn.heatmap()`) and subsequently calling the `matplotlib.table()` function, providing the necessary data array. Furthermore, the resulting table is highly customizable, allowing users to modify cell aesthetics such as color, font size, and text alignment to match the overall visual theme of the plot.

The most straightforward and professional method for integrating a numerical table into a Seaborn visualization is by utilizing the dedicated **`table()`** function available within the core Matplotlib library. Because Seaborn is built upon the foundational capabilities of Matplotlib, these two libraries work synergistically, allowing us to generate sophisticated statistical plots while retaining the precise control over figure elements that Matplotlib provides.

The following comprehensive guide provides a practical, step-by-step demonstration of how to implement this technique, showing how to seamlessly blend graphical output with the precise detail of the underlying tabular data.

Example: Integrating Tabular Data into a Seaborn Scatterplot

Bridging Data Visualization and Tabular Data

In the realm of data analysis, effective communication often requires showing both the trend (the plot) and the underlying source material (the table). This combination is particularly valuable when presenting results to a non-technical audience or when specific data points need immediate cross-referencing. By leveraging Matplotlib capabilities, we can ensure that our Seaborn statistical plots are self-contained and fully informative. This integration eliminates the need for viewers to look up separate data sources, maintaining focus on the visualization itself.

To demonstrate this functionality, we will begin by defining a simple pandas DataFrame. This data structure will represent information typical in a sports analytics context, tracking key metrics for individual players across various teams. This setup ensures that our example is grounded in a realistic scenario where context (the raw numbers) is vital to understanding the graphical

ARABPSYCHOLOGY.COM

representation (the relationship between points and assists).

Suppose we are working with the following pandas DataFrame, which contains crucial performance information for basketball players distributed across three distinct teams (A, B, and C):

import pandas as pd

#create DataFrame
df = pd.DataFrame({'team': ,
'points': ,
'assists': })

#view DataFrame
print(df)

team points assists
0 A 18 5
1 A 22 7
2 A 19 7
3 B 14 9
4 B 14 12
5 B 11 9
6 C 20 9
7 C 28 4
8 C 30 15

ARABPSYCHOLOGY.COM

Step-by-Step Implementation: Combining Plot and Table

With our DataFrame ready, the next step involves generating a statistical visualization using Seaborn and then appending the table using the `table()` function. We will use a `scatterplot` to visualize the relationship between 'assists' (x-axis) and 'points' (y-axis), differentiating the data points by 'team' using the `hue` parameter. This graphical representation immediately highlights potential team-specific clustering or performance variances.

The core functionality resides in the `plt.table()` call. This function requires several key arguments to correctly map the DataFrame contents onto the plot space. Specifically, `cellText` accepts the actual data values (obtained via `df.values`), `rowLabels` uses the index of the DataFrame (`df.index`), and `colLabels` takes the column names (`df.columns`). These elements collectively define the visual structure and content of the generated table.

Crucially, the **`bbox`** argument is initially set to position the table strategically beneath the primary plot area. The coordinates provided to **`bbox`** define a bounding box (left, bottom, width, height) in figure coordinates (ranging from 0 to 1). A setting like `(.2, -.7, 0.5, 0.5)` positions the table starting at 20% from the left, significantly below the plot (at -70% of the figure height relative to the plot's bottom), and gives it a moderate size.

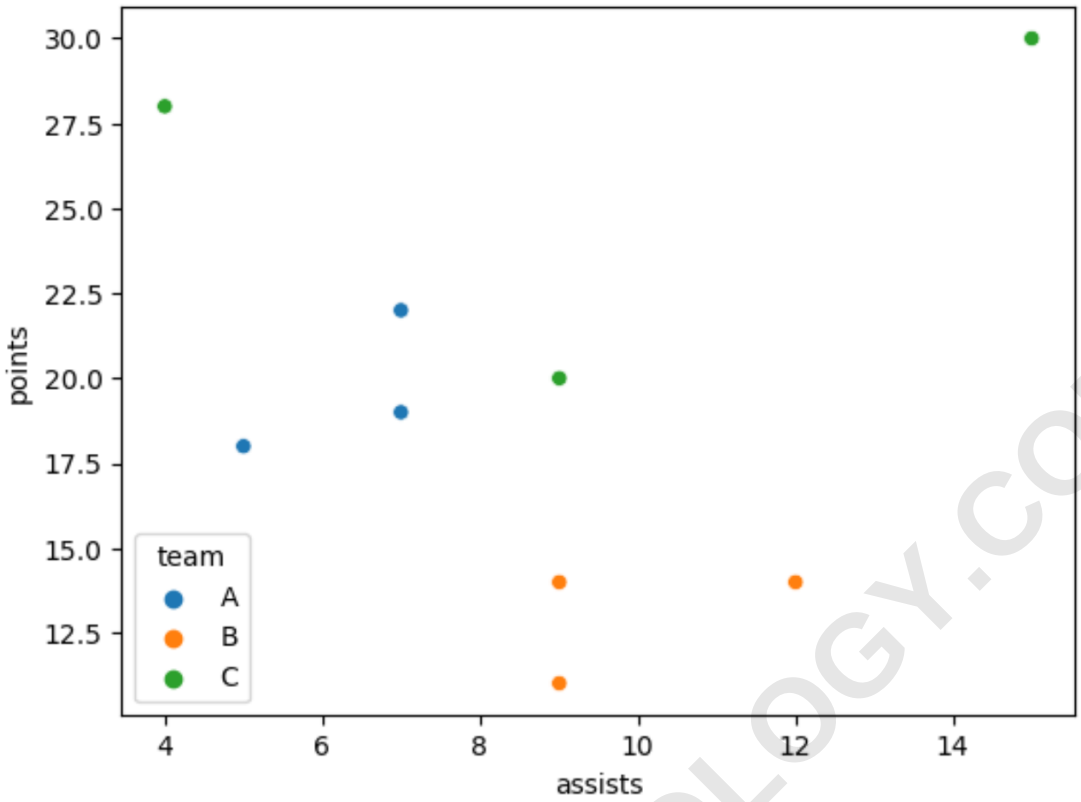
```
import seaborn as sns
import matplotlib.pyplot as plt

#create scatterplot of assists vs points
sns.scatterplot(data=df, x='assists', y='points', hue='team')

#add table below scatterplot
table = plt.table(cellText=df.values,
rowLabels=df.index,
colLabels=df.columns,
bbox=(.2, -.7, 0.5, 0.5))

#display final plot
plt.show()
```

ARABPSYCHOLOGY.COM



	team	points	assists
0	A	18	5
1	A	22	7
2	A	19	7
3	B	14	9
4	B	14	12
5	B	11	9
6	C	20	9
7	C	28	4
8	C	30	15

ARABPSYCHOLOGY.COM

Analyzing the Output and Initial Placement

As visible in the resulting image, the table successfully integrates with the scatterplot. The visualization itself occupies the upper portion, illustrating the distribution and grouping of points and assists, categorized by team. Directly beneath, the newly added table displays the precise raw data values that underpin the graphical elements. This juxtaposition is invaluable for validating outlier points or quickly confirming specific player statistics without cluttering the plot area itself.

This initial placement, achieved using negative values in the **bbox** bottom parameter, effectively pushes the table outside the primary axis frame and into the figure margin. This separation is often desirable for aesthetic purposes, ensuring that the table does not overlap or distract from the core

visual message conveyed by the data visualization. The table is clearly labeled with column headers ('team', 'points', 'assists') and the default DataFrame indices.

Understanding the `matplotlib.table()` Parameters

The `table()` function is highly configurable. Understanding its core parameters is essential for mastery:

`cellText`: This is the primary input, accepting a sequence of sequences (like a list of lists or a NumPy array), representing the data to be displayed in the cells. Using `df.values` converts the pandas DataFrame into the required array format.

`rowLabels` and `colLabels`: These parameters specify the text labels for the rows and columns, respectively. Utilizing `df.index` and `df.columns` ensures that the table retains the descriptive metadata from the original data structure.

`bbox` (Bounding Box): This is arguably the most critical parameter for layout control. It defines the position and dimensions of the table within the figure coordinate system. It takes a tuple of four floats: `(left, bottom, width, height)`. These values are normalized to the figure dimensions (0.0 to 1.0).

Beyond these core inputs, the function supports advanced customization options, such as `cellColours`, `rowColours`, and specific properties to adjust font size, cell padding, and alignment. Leveraging these options allows developers to create visually integrated tables that conform perfectly to corporate branding or specific editorial styles, moving far beyond the default grayscale appearance.

Customizing Table Placement using the `bbox` Argument

The flexibility of the **`bbox`** argument allows developers to position the data table in virtually any location within the overall figure canvas. While placing the table below the plot (as demonstrated initially) is common, situations often arise where a side-by-side arrangement is preferred, especially when the plot itself is compact or when available space in the figure margin is wider than it is taller.

To reposition the table to the right side of the scatterplot, we must adjust the **`bbox`** coordinates significantly. We need a larger `left` value (to move it horizontally) and a positive `bottom` value (to align it vertically with the plot axes).

Consider the new **`bbox`** definition: `(1.1, .2, 0.5, 0.5)`.

The `left` value of **1.1** places the table's left edge slightly outside the standard plot area (which

ARABPSYCHOLOGY.COM

typically ends at 1.0). This ensures the table resides in the right margin without overlapping the scatterplot markers or axes.

The `bottom` value of **0.2** lifts the table vertically, aligning it appropriately relative to the scatterplot's y-axis.

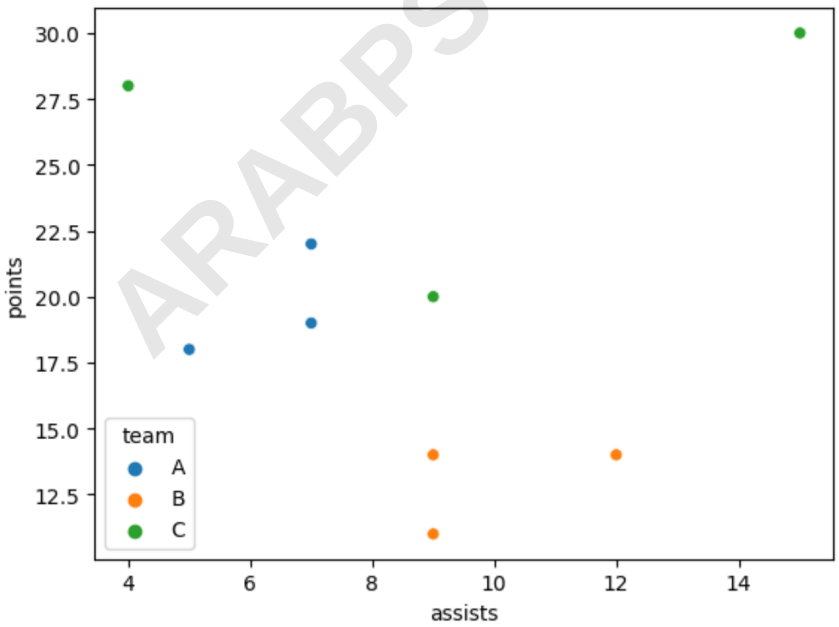
The `width` and `height` parameters (0.5, 0.5) control the physical size of the table, though the final size is constrained by the number of rows and columns.

```
import seaborn as sns
import matplotlib.pyplot as plt

#create scatterplot of assists vs points
sns.scatterplot(data=df, x='assists', y='points', hue='team')

#add table to the right of the scatterplot
table = plt.table(cellText=df.values,
rowLabels=df.index,
colLabels=df.columns,
bbox=(1.1, .2, 0.5, 0.5))

#display final plot
plt.show()
```



	team	points	assists
0	A	18	5
1	A	22	7
2	A	19	7
3	B	14	9
4	B	14	12
5	B	11	9
6	C	20	9
7	C	28	4
8	C	30	15

ARABPSYCHOLOGY.COM

Interpreting Side-by-Side Visualizations

The result of adjusting the **`bbox`** parameters is a horizontally extended figure where the data table sits comfortably to the right of the scatterplot. This configuration is highly effective when maximizing screen real estate or when both the visual pattern and the specific numerical values must be viewed concurrently without excessive scrolling or eye movement. This practice is common in academic publications and detailed financial reports where data integrity and accessibility are paramount to sound analysis.

It is important to remember that the figure size (which can be controlled via `plt.figure(figsize=(width, height))` before plotting) will significantly influence how the normalized **`bbox`** coordinates translate into actual pixel placement. When positioning elements outside the primary axes (i.e., when using coordinate values greater than 1.0 or less than 0.0), manual adjustment of the figure size might be necessary to ensure the table is not clipped or rendered outside the viewing window.

Advanced Customization of Table Aesthetics

Beyond simple positioning, the visual attributes of the table cells can be deeply customized using additional arguments in the `table()` function. Achieving an integrated and professional look often requires matching the table's color scheme and typography to the Seaborn plot's theme.

Specific properties that can be adjusted include:

Cell Colors (`cellColours`): Allows for the application of background colors to specific cells, rows, or columns, useful for highlighting specific data entries or differentiating headers.

Font Properties (Cell objects): Once the table object is created (e.g., `table = plt.table(...)`), individual cell objects can be accessed and modified to change font size, weight, and color, enhancing readability, especially for small tables.

Alignment (`cellLoc` and `rowLoc`): Controls how text is aligned within the table cells (e.g., 'center', 'left', 'right'), which is critical for numerical precision.

Experimentation with the **`bbox`** values is highly encouraged to achieve the perfect layout. Since figure coordinates are relative, slight changes (e.g., changing `1.1` to `1.05` or `-.7` to `-.65`) can fine-tune the table's final position, ensuring it complements the data visualization without intrusion.

Conclusion: Best Practices for Combined Visualizations

The methodology of combining Seaborn statistical plots with Matplotlib's table functionality represents a robust approach to data communication. This synergy leverages Seaborn's ability to

ARABPSYCHOLOGY.COM

produce high-level statistical summaries (like distributions, relationships, and correlations) while retaining the foundational clarity provided by the raw numerical data. By mastering the **`bbox`** parameter, users gain complete control over the layout, ensuring the resulting figures are not just visually appealing but also maximally informative.

Always ensure that when placing tables outside the primary axes, you utilize the **`bbox`** argument to precisely manage placement. Referencing the official documentation for the `Matplotlib table()` function is always the best practice for exploring the full range of customization options available, guaranteeing valid and reproducible results in any data analysis pipeline.

Feel free to continue experimenting with the positional parameters and aesthetic arguments to achieve the exact visual integration required for your specific reporting needs.

Note: You can find the complete documentation for the Matplotlib **`table()`** function here.

ARABPSYCHOLOGY.COM