

How do I plot a distribution in Seaborn? (With examples)

Authored by
stats writer

November 21, 2025

RECOMMENDED CITATION

stats writer (2025). *How do I plot a distribution in Seaborn? (With examples)*.

PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=99169>

As an indispensable tool within the modern data science ecosystem, Seaborn stands out as a powerful Python library. It builds upon Matplotlib, offering a high-level interface specifically designed for generating statistically rigorous and visually appealing graphics. Analyzing the structure and shape of a dataset is fundamental to any statistical inquiry, and this is typically achieved through plotting the data's distribution. Seaborn simplifies this process immensely, especially through the use of the versatile **displot()** function, which is the recommended successor to the older `distplot()` function for plotting univariate distributions.

The primary purpose of a distribution plot is to visualize how a set of data points is spread across a range of values. This visualization helps analysts quickly identify key characteristics such as the central tendency, variance, skewness, and the presence of multimodal peaks. The **displot()** function is exceptionally flexible, allowing users to choose between several different plot types, including histograms, kernel density estimates (KDEs), and empirical cumulative distribution functions (ECDFs). By default, **displot()** often generates a frequency-based plot, such as a histogram, that maps data values to their respective counts or normalized frequencies.

Customization is a core strength of Seaborn. The **displot()** function accepts numerous parameters that allow fine-tuning of the output, ensuring the visualization accurately reflects the underlying data dynamics. These parameters include controls over the number of bins in a histogram, the bandwidth smoothing used in a KDE, and aesthetic choices like color palettes and axis ranges. For instance, if you wanted to plot a distribution of ages with a bin size of 5 years and a dark blue color, you might use the following syntax: `sns.displot(ages, bins=5, color='darkblue')`. This immediate customization capability ensures that your statistical graphics are not just attractive, but also highly informative and tailored to the specific context of your analysis.

Core Methods for Distribution Visualization

To effectively visualize the distribution of numerical values within your dataset using Seaborn, you primarily rely on the highly adaptable **displot()** function. Below, we detail the three fundamental approaches available that cater to different analytical needs regarding data shape and frequency representation:

Method 1: Plot Distribution Using Histogram (Focuses on raw counts or frequencies within discrete bins.)

Method 2: Plot Distribution Using Density Curve (Focuses on a smoothed, continuous probability density estimate.)

Method 3: Plot Distribution Using Histogram & Density Curve (Combines both frequency counts and continuous density estimates for comprehensive insight.)

You can use the following methods to plot a distribution of values in Python using the data

visualization library:

Method 1: Plot Distribution Using Histogram

```
sns.displot(data)
```

Method 2: Plot Distribution Using Density Curve

```
sns.displot(data, kind='kde')
```

Method 3: Plot Distribution Using Histogram & Density Curve

```
sns.displot(data, kde=True)
```

The following examples show how to use each method in practice, providing detailed code implementations and resulting visualizations.

Visualizing Data Distribution Using a Histogram (Method 1)

The simplest and perhaps most common way to display a univariate distribution is through a histogram. A histogram groups numerical data into "bins" and counts how many data points fall into each bin. The height of the bar in the plot then corresponds directly to the frequency or count of observations within that bin's range. When the **displot()** function is called in Seaborn without specifying the `kind` argument, it defaults to creating this type of frequency visualization, providing an immediate sense of the data's inherent shape and location.

The following example utilizes the **displot()** function to visualize the spread of values contained within a synthetically generated NumPy array. We first ensure reproducibility by setting a random seed, then create 1,000 data points sampled from a normal distribution centered around a mean of 10. This foundational step is crucial for ensuring that statistical experiments can be verified and repeated reliably across different environments.

The accompanying code block demonstrates the setup and the initial call to **displot()**, resulting in a standard histogram plot. The axis interpretation is straightforward: the horizontal x-axis maps the range of values in the distribution, while the vertical y-axis indicates the raw count (frequency) of data points observed within each designated bin.

The following code shows how to plot the distribution of values in a NumPy array using the **displot()** function in seaborn:

```
import seaborn as sns
```

```
import numpy as np
```

```
#make this example reproducible
```

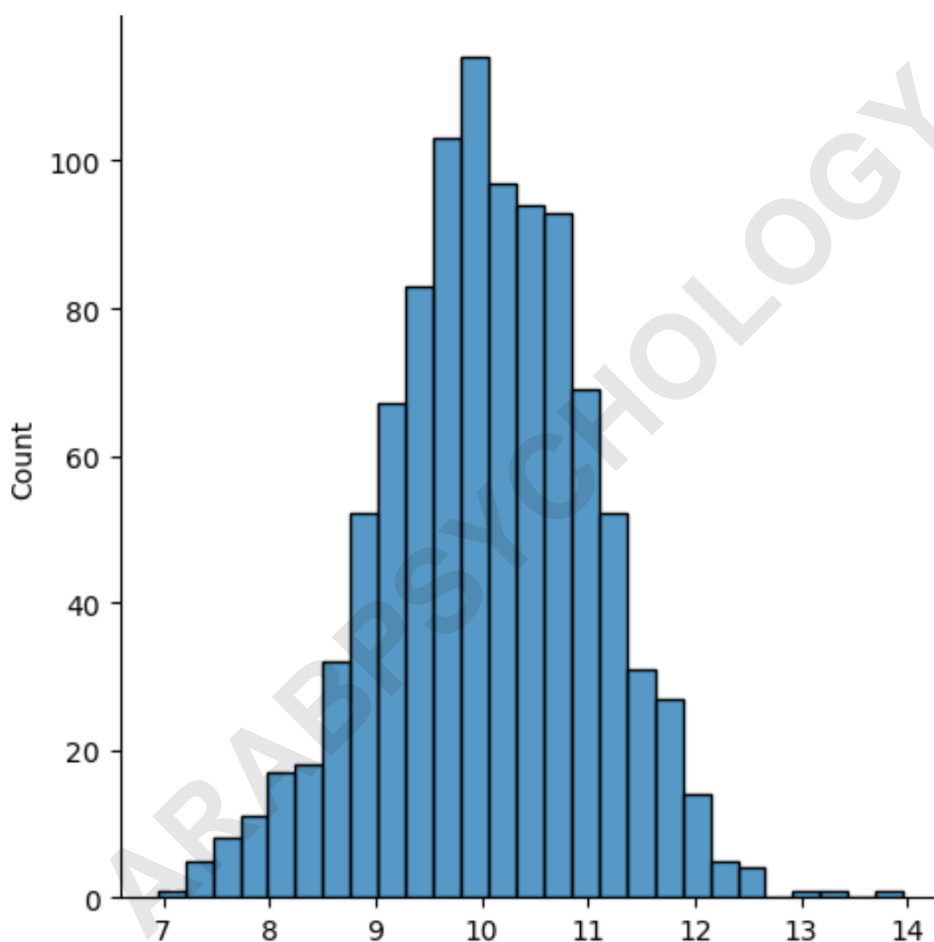
```
np.random.seed(1)
```

```
#create array of 1000 values that follow a normal distribution with mean of 10
```

```
data = np.random.normal(size=1000, loc=10)
```

```
#create histogram to visualize distribution of values
```

```
sns.displot(data)
```



As illustrated by the output, the x-axis precisely displays the range of data values present in the distribution, and the y-axis quantifies the count or frequency of observations falling within the boundary of each plotted bar.

Customizing Histograms: Mastering the `bins` Argument

A crucial decision when plotting a histogram involves selecting the appropriate number of bins. The bin size directly influences the visual interpretation of the data; too few bins can smooth out significant details, masking underlying patterns, while too many bins can introduce visual noise, making the overall distribution shape difficult to discern. Seaborn provides intelligent default bin selections, but often, statistical clarity requires manual adjustment using the `bins` argument within the **`displot()`** function.

The `bins` argument allows the user to explicitly define how many intervals the data should be partitioned into. For instance, setting `bins=10` instructs Seaborn to divide the entire data range into 10 equally sized segments before calculating the frequency counts for each segment. This granular control over the aggregation level is essential for highlighting features specific to certain subsets of the data, especially when dealing with skewed or multi-modal distributions.

The example below demonstrates the impact of specifying a defined number of bins. By explicitly setting `bins=10`, we observe a much coarser visualization compared to the default, which automatically calculated an optimal number based on the dataset size and variance. This illustrates the trade-off between detail and generalization when structuring a distribution plot for reporting or analysis.

To change the number of bins used in the histogram, you can specify a number using the **`bins`** argument:

```
import seaborn as sns
```

```
import numpy as np
```

```
#make this example reproducible
```

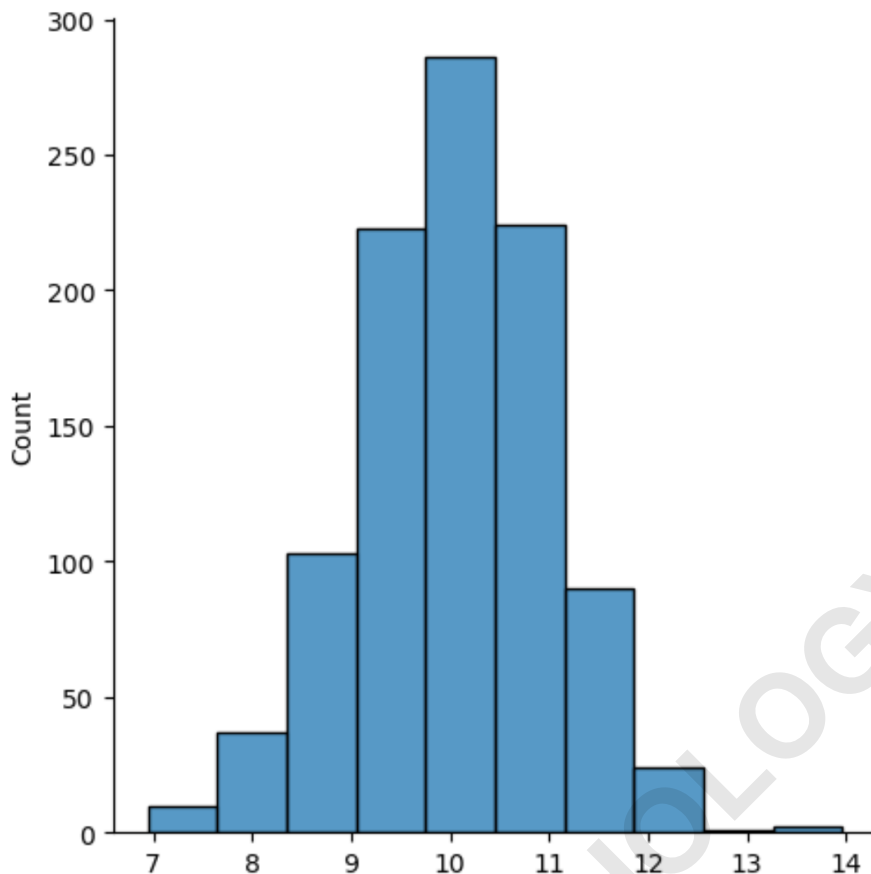
```
np.random.seed(1)
```

```
#create array of 1000 values that follow a normal distribution with mean of 10
```

```
data = np.random.normal(size=1000, loc=10)
```

```
#create histogram using 10 bins
```

```
sns.displot(data, bins=10)
```



Smoothing Distributions with a Density Curve (Method 2)

While histograms provide discrete counts, a Kernel Density Estimate (KDE) offers a smoother, continuous view of the distribution. KDE attempts to estimate the underlying probability density function of the data, essentially replacing the sharp steps of a histogram with a continuous curve. This method is particularly useful when the goal is to visualize the overall shape of the distribution without the visual dependency on bin selection.

To generate a density curve using the **displot()** function, we must set the `kind` parameter to `'kde'`. The resulting visualization presents the estimated probability density on the y-axis, rather than raw counts. This relative frequency interpretation means that the total area under the density curve is scaled to equal 1, making it a true probability distribution representation.

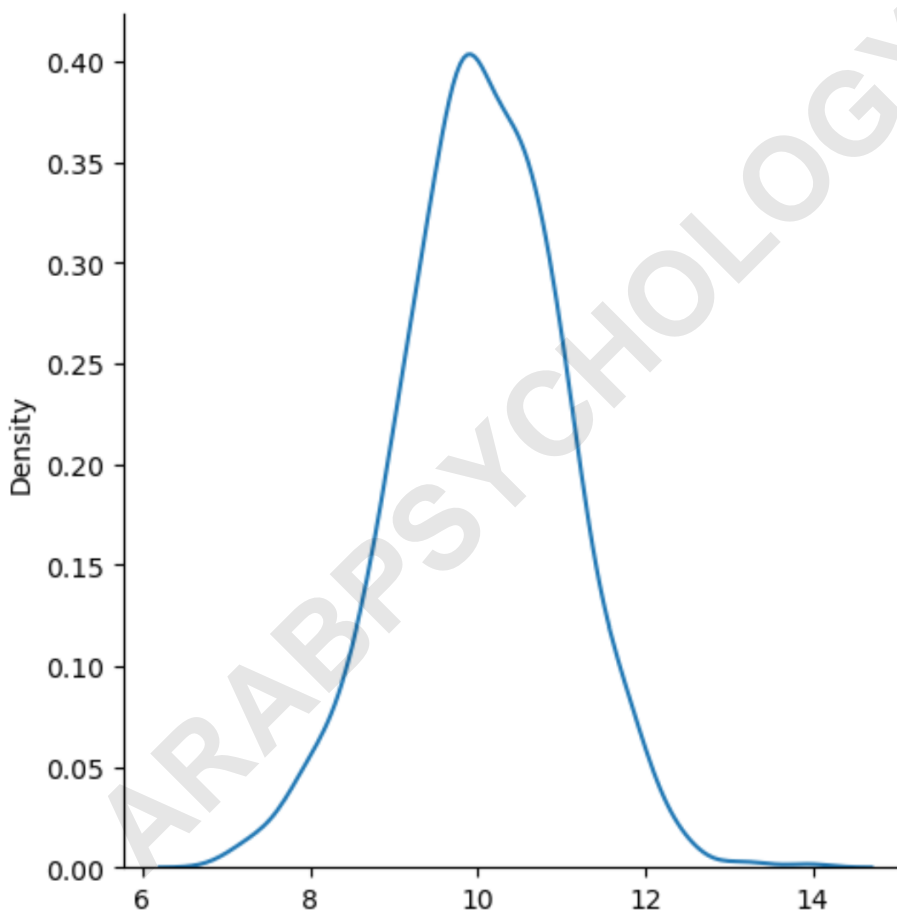
The smoothness of the KDE plot is controlled by the bandwidth parameter, although Seaborn typically handles this automatically using rules of thumb optimized for visual clarity. Analyzing the density curve allows for precise identification of modes (peaks) and rapid assessment of distribution characteristics such as symmetry or skewness, which might be obscured by the jagged nature of a histogram.

```
import seaborn as sns
import numpy as np

#make this example reproducible
np.random.seed(1)

#create array of 1000 values that follow a normal distribution with mean of 10
data = np.random.normal(size=1000, loc=10)

#create density curve to visualize distribution of values
sns.displot(data, kind='kde')
```



In this plot, the x-axis represents the data values, while the y-axis corresponds to the estimated relative frequency density of each value. Note that **kind='kde'** tells `seaborn` to use **kernel density estimation**, which produces a smooth curve that summarizes the distribution of values for a variable.

Combining Histogram and Density Curve for Comprehensive Analysis (Method 3)

For the most comprehensive visualization of a univariate distribution, data analysts often overlay a continuous density curve onto a discrete histogram. This combined approach leverages the strengths of both plot types: the histogram retains the raw frequency counts and shows exactly how the data aggregated into specific bins, while the density curve provides a generalized, smoothed view of the data's underlying probability distribution.

The **displot()** function makes this combination remarkably simple. Instead of setting the `kind` parameter, we merely need to set the `kde` parameter to `True`. When `kde=True`, Seaborn automatically generates the default histogram representation and then calculates and draws the corresponding KDE line on top of the bars. This hybrid visualization is particularly effective for presenting statistical findings, as it allows the audience to see the empirical evidence (the counts in the bars) while simultaneously guiding their eye toward the theoretical or idealized distribution shape (the smooth curve).

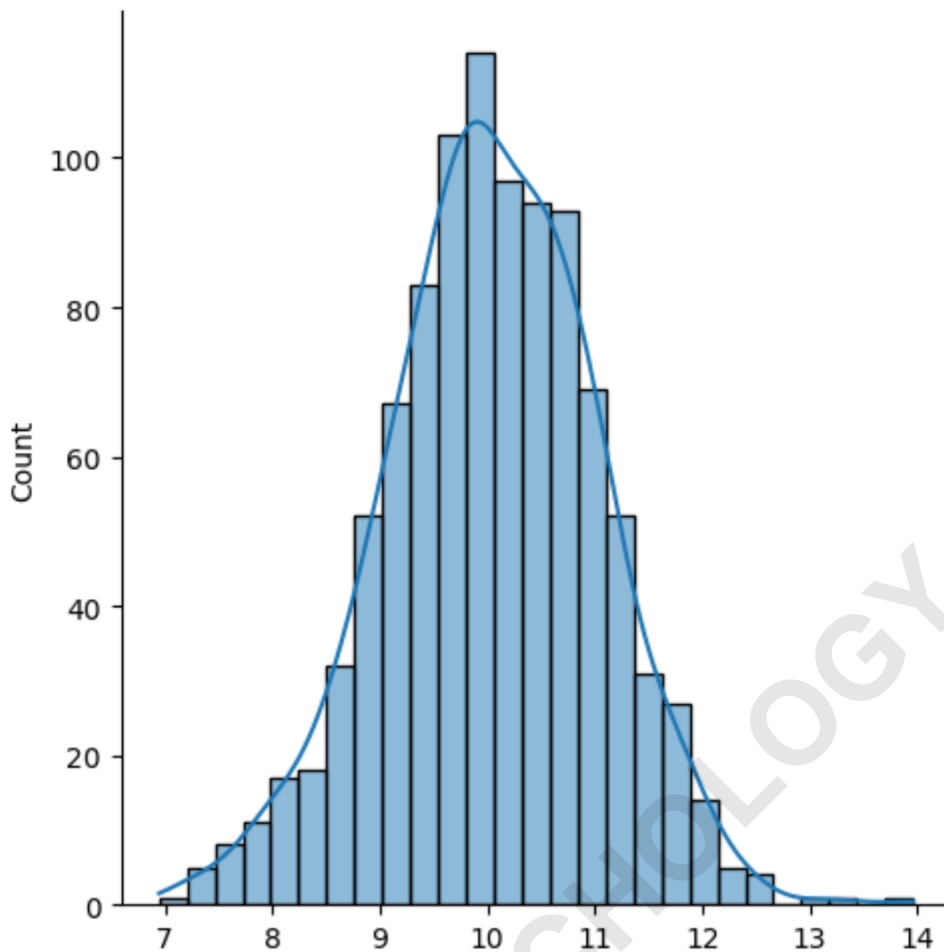
The following code shows how to plot the distribution of values in a NumPy array using a histogram with a density curve overlaid:

```
import seaborn as sns
import numpy as np

#make this example reproducible
np.random.seed(1)

#create array of 1000 values that follow a normal distribution with mean of 10
data = np.random.normal(size=1000, loc=10)

#create histogram with density curve overlaid to visualize distribution of values
sns.displot(data, kde=True)
```



The result is a compelling visualization featuring a histogram with a density curve overlaid.

Further Exploration and Resources

Mastering the **displot()** function is essential for anyone conducting statistical analysis in Python. Its flexibility allows developers and analysts to switch seamlessly between frequency-based histograms and probability-based density estimations, catering to various analytical needs. Furthermore, **displot()** integrates seamlessly with other features of Seaborn, supporting faceted plots and conditional distributions across multiple variables.

We encourage thorough examination of the official documentation for advanced customization options, including control over elements like rug plots, bandwidth selection, and normalization techniques, all of which enhance the fidelity of the final visualization.

Note: You can find the complete documentation for the seaborn **displot()** function [here](#).

Related Seaborn Tutorials

The following tutorials explain how to perform other common tasks using seaborn:

ARABPSYCHOLOGY.COM