Learning to Visualize Data Distributions with Seaborn in Python

Authored by

Mohammed looti

February 9, 2026

Effectively performing **data visualization** is a crucial and non-negotiable step in the data science pipeline, allowing analysts to uncover underlying patterns, assess data quality, and understand the intrinsic characteristics of a dataset. When working in **Python**, the **Seaborn** library stands out as an indispensable tool, offering powerful and highly intuitive functions for creating compelling statistical graphics. This comprehensive guide is designed to walk you through the various methods available in Seaborn to plot the **distribution** of values within your data, enabling you to gain profound insights into its shape, spread, and central tendency.

The Role of Distribution Plots in Data Analysis

A **distribution** fundamentally describes the range of all possible values for a given variable and the frequency with which each value occurs. Visualizing these distributions is a cornerstone of **exploratory data analysis (EDA)**. By observing the distribution, we can quickly identify critical features such as the presence of outliers, measure data variability, and make educated assumptions about the underlying data generation process before moving on to inferential statistics.

Seaborn simplifies this process significantly with its versatile `displot()` function. This function is a figure-level interface--meaning it manages the entire figure layout--and is specifically engineered for plotting univariate distributions. It provides a unified and cohesive approach to visualizing how a single column of data is spread across its range.

The `displot()` function serves as a flexible wrapper capable of producing several distinct types of distribution plots. Depending on the arguments provided, it can generate traditional histograms, smoothed kernel density estimates, or even empirical cumulative distribution functions. We will now delve into the three primary and most common methods for visualizing univariate distributions using this exceptionally powerful function.

Method 1: Plot Distribution Using a Histogram

The **histogram** is arguably the most recognized method for visualizing the frequency distribution of a single variable. This technique works by partitioning the entire range of observed values into a consecutive series of intervals, commonly referred to as **bins**. The height of the resulting bar in the plot then corresponds directly to the count or frequency of data points that fall within that specific bin.

When utilizing Seaborn's `displot()` function, omitting the optional `kind` argument will automatically instruct the library to generate a histogram. This default behavior makes it highly efficient for quickly grasping the overall shape of your data, identifying the number of modes (peaks), and detecting signs of asymmetry or skewness in the distribution.

sns.displot(data)

Method 2: Plot Distribution Using a Density Curve (KDE)

In contrast to the discrete, stepped nature of a histogram, a **density curve** provides a smooth, continuous estimate of the underlying probability density function of the data. This smooth curve is generated using **Kernel Density Estimation (KDE)**, which is a powerful non-parametric technique used to estimate the density function of a random variable. KDE is especially valuable when working with continuous data, as it minimizes the visualization noise caused by arbitrary bin choices in histograms.

To instruct Seaborn to plot a density curve instead of a histogram, you must explicitly set the `kind` parameter to the value `'kde'` within the **`displot()`** function call. The resulting plot is a line graph where the vertical axis (y-axis) represents the estimated probability density. The curve smoothly illustrates where the data points are concentrated, providing a clear visual representation of the distribution's theoretical shape.

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

Method 3: Plot Distribution Using Both Histogram & Density Curve

For the most complete and nuanced understanding of your data's distribution, combining the histogram's raw frequency counts with the KDE curve's smoothed estimate is often the preferred method. This dual approach allows the viewer to simultaneously assess the actual data counts within specific bins and visualize the estimated underlying statistical trend, offering a robust visual summary.

Seaborn facilitates this combined plot with great simplicity. By setting the Boolean parameter `kde` to `True` within the **`displot()`** function, the histogram will be generated by default, and the Kernel Density Estimate curve will be gracefully overlaid onto the bars. This highly insightful visualization technique is effective for both deep analytical review and clear communication of distribution characteristics to a non-technical audience.

sns.displot(data, kde=True)

Practical Implementation: Visualizing a Normal Distribution

To illustrate these methods, we will work through practical examples using **Seaborn** alongside the powerful array manipulation library, **NumPy**. For consistency and reproducibility across all

examples, we will generate a synthetic dataset consisting of 1000 data points drawn from a standard **normal distribution**, specifically centered with a mean of 10. We will initialize the random number generator using `numpy.random.seed(1)` to ensure that your results perfectly match the visuals provided here.

Example 1: Plot Distribution Using a Histogram

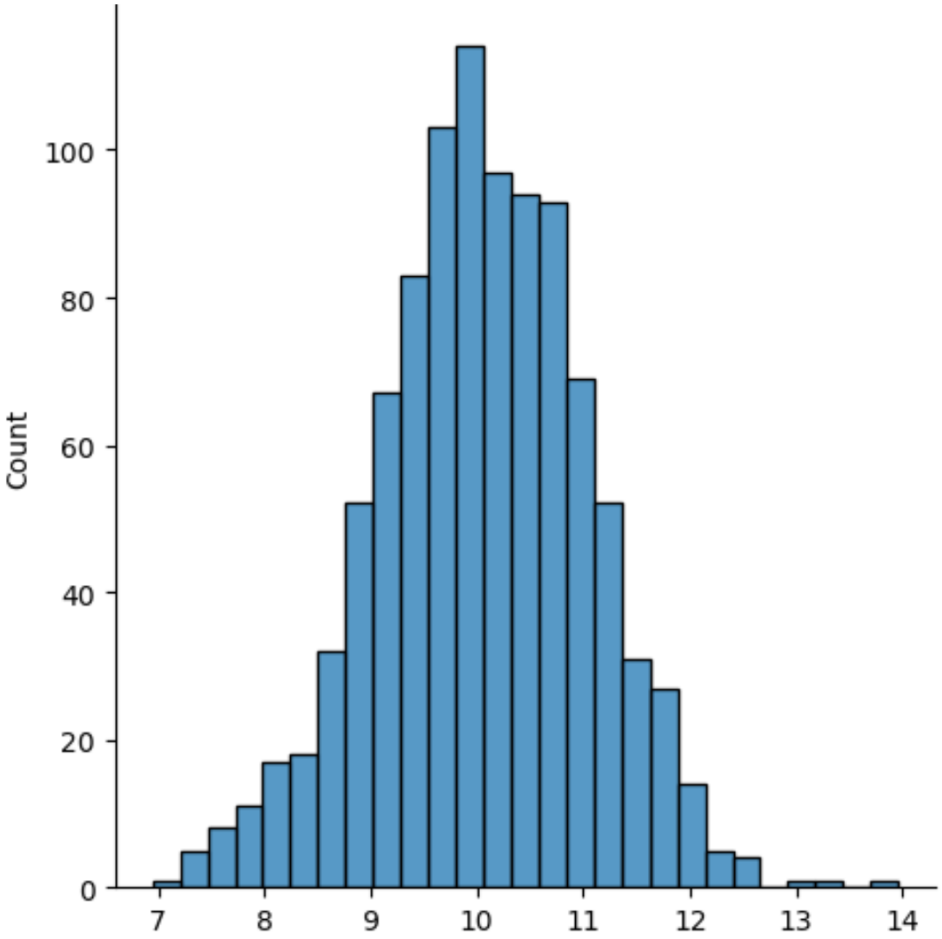
This first implementation demonstrates the creation of the default histogram plot. The code snippet below sets up the necessary libraries, generates the simulated data, and then calls **`sns.displot(data)`** to produce the visual representation of the frequency distribution.

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)



In this resulting plot, the **x-axis** represents the measured values in our distribution, while the **y-axis** displays the count or absolute frequency of data points located within each defined bin. This visualization clearly exhibits the classic bell shape expected of a normal distribution, centered around the value of 10.

It is important to note that the number of **bins** chosen for the histogram has a substantial impact on its visual appearance and subsequent interpretation. Using too few bins can smooth over crucial details, resulting in a loss of granularity, while using too many bins can create a noisy, jagged appearance. To demonstrate control over this parameter, the following code snippet explicitly adjusts the number of bins used in the visualization:

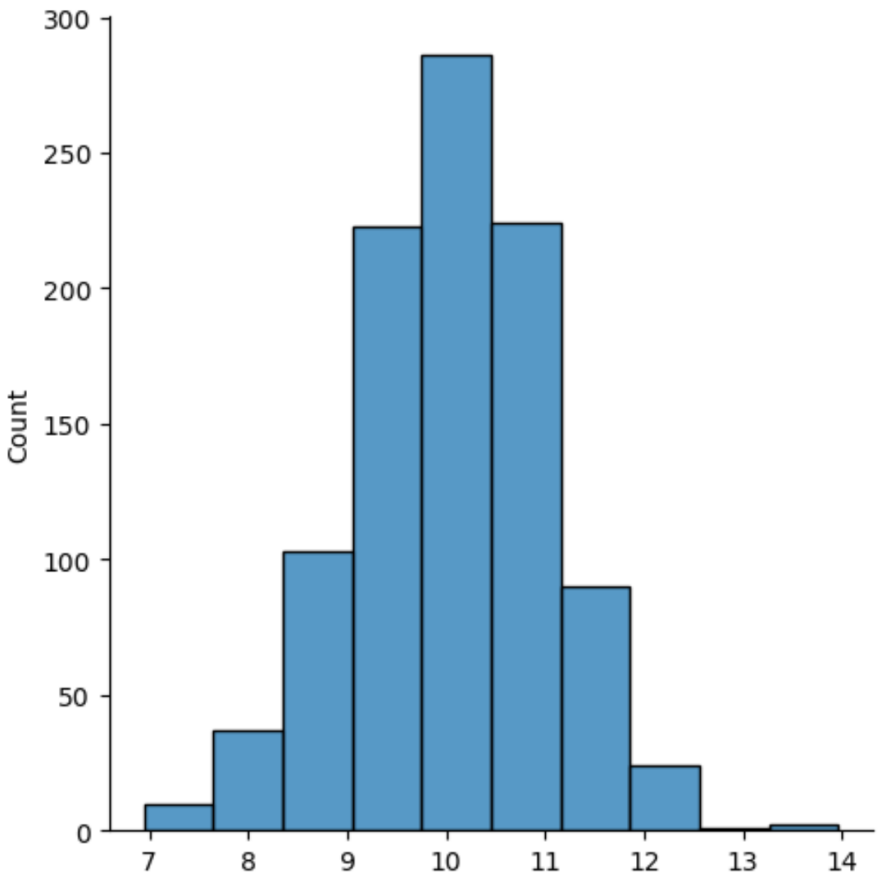
```
import seaborn as sns
import numpy as np

#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)
```



By observing the image above, notice how drastically changing the number of bins alters the granularity of the histogram, offering a different resolution on the data's frequency distribution. Determining the optimal number of bins frequently involves a degree of trial and error to achieve the most informative and visually balanced representation.

Example 2: Plot Distribution Using a Density Curve (KDE)

This second example focuses on generating a smooth **density curve**, demonstrating the power of **Kernel Density Estimation**. This visualization method is preferred when the primary goal is to infer the underlying continuous probability function, bypassing the inherent discretization caused by binning the data.

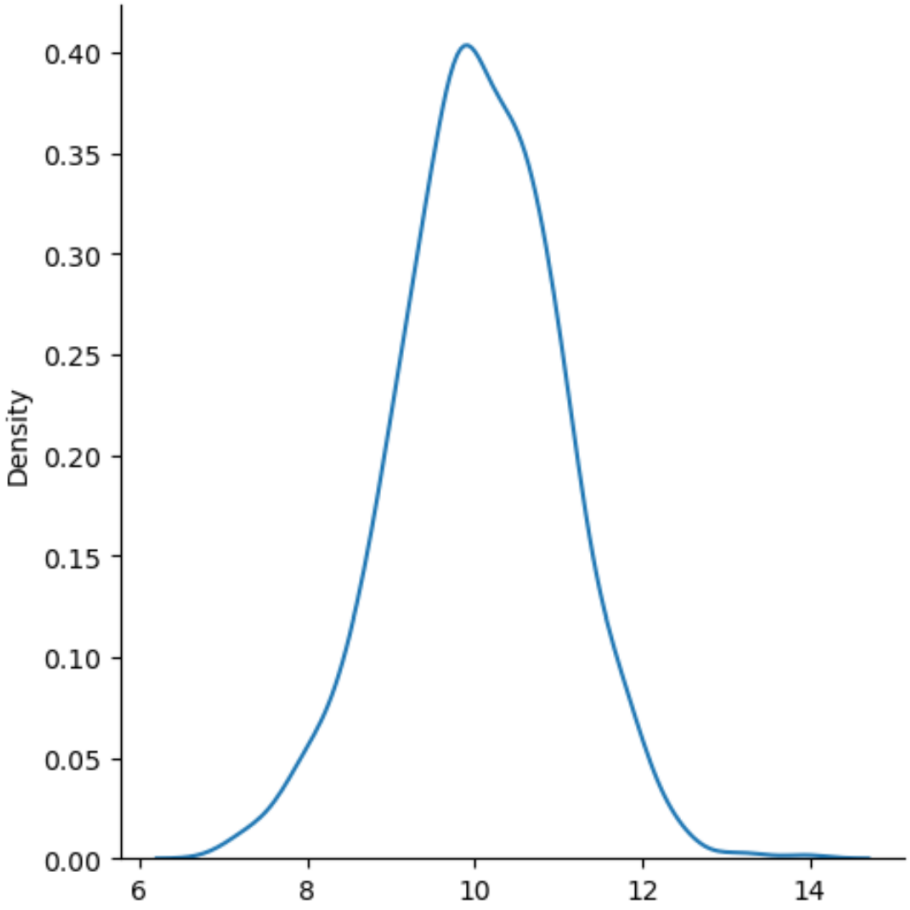
import seaborn as sns

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 density plot, the **x-axis** continues to represent the data values, but the **y-axis** now quantifies the estimated probability density. A key characteristic of KDE plots is that the total area positioned underneath the curve integrates to 1. The explicit use of the `kind='kde'` argument tells Seaborn to generate this non-parametric estimate, resulting in a smooth, elegant curve that effectively summarizes the central tendencies and spread of the continuous variable.

Example 3: Plot Distribution Using Histogram & Density Curve

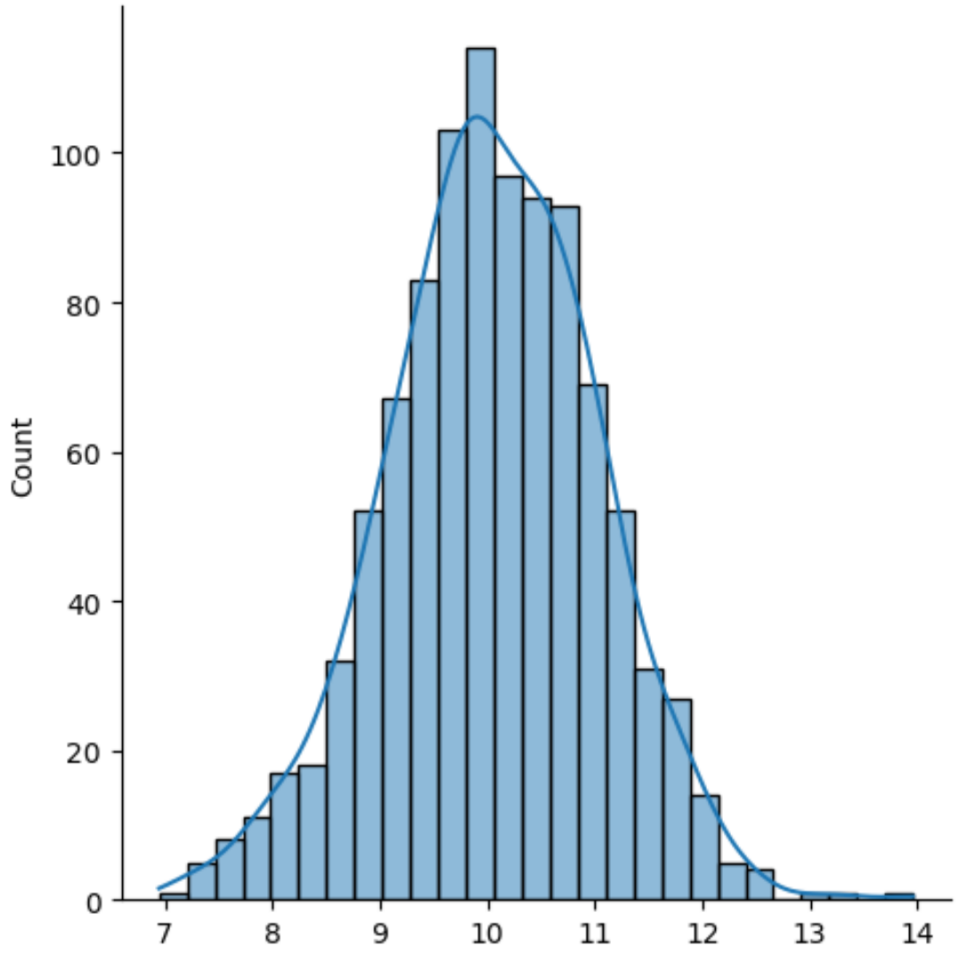
As previously discussed, achieving a comprehensive view often requires leveraging the strengths of both methods. This final practical example illustrates how to seamlessly overlay the smoothed density curve onto the binned histogram, creating a powerful composite visual summary of the data distribution.

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 resultant plot clearly displays a traditional **histogram** with a smooth **density curve** overlaid. This synergy allows for a direct visual assessment of how well the estimated KDE curve fits the observed frequency counts in the histogram bins. It is an extremely effective visualization for simultaneously highlighting both localized frequency variations and the overarching global trend of the distribution.

Decoding Your Visualizations: Key Interpretive Concepts

Generating a distribution plot is only the first step; extracting meaningful insights requires knowing how to interpret the resulting shapes and features. Here are the key elements you should analyze in any distribution plot:

Shape and Skewness: Carefully observe the symmetry of the plot. Is the distribution symmetric (like the normal distribution example)? Is it skewed left (meaning the long tail extends to the left, indicating most values are concentrated on the right)? Or is it skewed right (tail extends to the right)? Skewness informs you about the concentration of the bulk of the data.

Central Tendency: Identify the peak(s) of the distribution, which represent the most frequently occurring values. In a single-peaked (unimodal) distribution, this peak provides a strong visual indicator of the mean, median, and mode.
Spread and Variability: Assess the width of the distribution. A wide, flat plot signifies high variability, suggesting the data points are widely dispersed. Conversely, a narrow, tall distribution indicates low variability, meaning the data points are tightly clustered around the central tendency.
Outliers: Actively look for isolated bars (in a histogram) or small, separate bumps far from the main body of the distribution. These potential anomalies might represent genuine outliers, measurement errors, or perhaps an interesting sub-population within your dataset.
Modality: Determine the number of distinct peaks in the plot. A single peak is unimodal, two peaks are bimodal, and more than two are multimodal. Bimodal distributions often signal that the dataset combines two distinct, underlying groups or processes.

These visual cues are fundamentally important for summarizing the characteristics of your dataset and guiding subsequent statistical modeling and hypothesis testing.

Conclusion and Next Steps

The **`displot()`** function in **Seaborn** is an essential and remarkably versatile utility for visualizing univariate data **distributions** within the **Python** data science ecosystem. Whether your analysis requires the precise frequency counts provided by a **histogram**, the smooth, inferential estimate of a **Kernel Density Estimate**, or the superior insight of a combined plot, **`displot()`** offers an intuitive path to achieve robust visualization goals. By mastering these three core techniques, you are well-equipped to unlock deeper understandings of your data's underlying statistical structure.

For analysts seeking further detail and advanced customization options, the complete documentation for the **Seaborn `displot()` function** remains the definitive resource.

Additional Resources for Seaborn Mastery

To further expand your proficiency in **Seaborn** and statistical plotting, we recommend exploring these related tutorials that detail how to perform other common data visualization tasks critical to data analysis:

How to Create a Scatter Plot in Seaborn
How to Create a Box Plot in Seaborn
How to Create a Bar Plot in Seaborn