Learning to Visualize Data: A Step-by-Step Guide to Creating Relative Frequency Histograms with Matplotlib

Authored by

Mohammed looti

October 30, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Learning to Visualize Data: A Step-by-Step Guide to Creating Relative Frequency Histograms with Matplotlib*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=6164

Understanding Relative Frequency Histograms

A **relative frequency histogram** is a powerful graphical tool that visually represents the proportion of occurrences of values within specific intervals, or **bins**, in a **dataset**. Unlike a standard **frequency histogram** which shows raw counts, a relative frequency histogram displays these counts as fractions or percentages of the total number of observations. This makes it particularly useful for comparing distributions across datasets of different sizes.

This visualization method offers immediate insights into the underlying distribution of data, highlighting the relative prominence of certain value ranges. It helps in understanding patterns and making comparisons more intuitive, providing a clear picture of how data points are distributed proportionally.

Essential Matplotlib Syntax for Relative Frequency Histograms

To create a relative frequency histogram in **Matplotlib**, the popular plotting library in **Python**, you need to leverage its **`hist()` function**. The key to transforming a standard frequency histogram into a relative one lies in utilizing the **`weights` parameter** within this function.

The following syntax snippet demonstrates the core components required. It involves importing necessary libraries like **Numpy** for numerical operations and Matplotlib for plotting, setting up a plotting area, and then calling the `hist()` function with the appropriate `weights` argument.

```
import matplotlib.pyplot as plt
import numpy as np

#define plotting area
fig = plt.figure()
ax = fig.add_subplot(111)

#create relative frequency histogram
ax.hist(data, edgecolor='black', weights=np.ones_like(data) / len(data))
```

The `weights` parameter is crucial here; by passing an array of ones divided by the total number of data points, we effectively normalize the counts, making the y-axis represent proportions rather than raw frequencies. The `edgecolor='black'` argument simply adds borders to the histogram bars for better visual separation.

Example: Creating a Relative Frequency Histogram in Matplotlib

Let's walk through a practical example to illustrate how to implement the syntax discussed above.

We will use a simple dataset to demonstrate the process of generating both a standard frequency histogram and then converting it into a relative frequency histogram using Matplotlib. This step-by-step approach will clarify the differences and the impact of the `weights` parameter.

Our example will begin by defining a sample dataset and then visualizing its distribution. Following this, we will adjust the plotting parameters to display relative frequencies, offering a comparative view of the data's distribution.

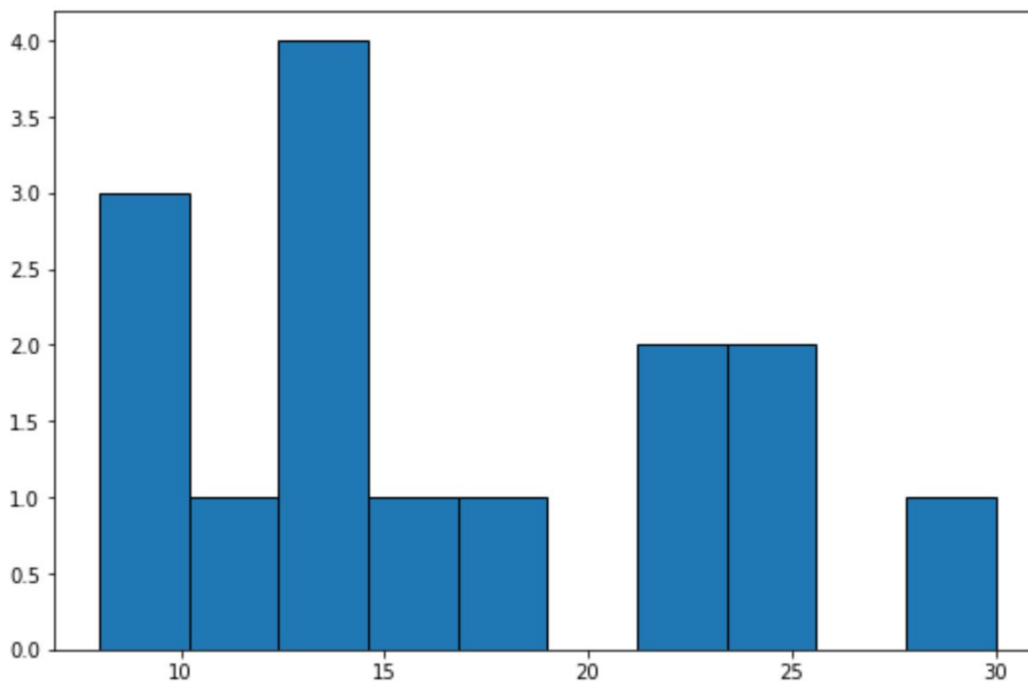
Step 1: Visualizing a Standard Frequency Histogram

Before diving into relative frequencies, it's beneficial to first understand how a standard frequency histogram is constructed. This provides a baseline for comparison and helps appreciate the transformation when relative frequencies are applied. The following code snippet will generate a basic frequency histogram from our sample data.

```
import numpy as np
import matplotlib.pyplot as plt

#define data values
data =

#create frequency histogram
fig = plt.figure()
ax = fig.add_subplot(111)
ax.hist(data, edgecolor='black')
```



In this standard **frequency histogram**, the **x-axis** represents the ranges of data values (the **bins**), while the **y-axis** indicates the absolute count, or frequency, of data points falling into each bin. For instance, if a bar reaches '4' on the y-axis, it means four data points from our dataset fall within that specific range on the x-axis. This visualization is fundamental for understanding the raw distribution of data.

Step 2: Generating the Relative Frequency Histogram

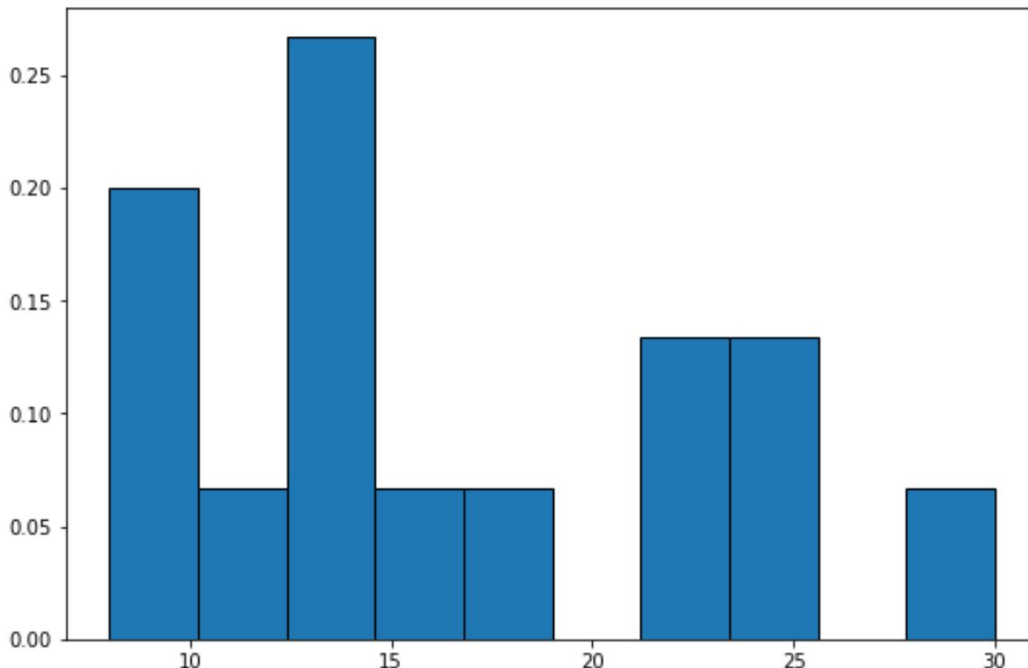
Now, let's modify our code to display **relative frequencies** on the y-axis, providing a clearer proportional view of our data distribution. This is achieved by introducing the `weights` parameter into the **`hist()` function** call. The `weights` parameter takes an array of the same length as our data, where each element is the reciprocal of the total number of data points.

The crucial adjustment is `weights=np.ones_like(data) / len(data)`. This expression creates an array where each element is `1 / total_number_of_data_points`. When passed to `hist()`, Matplotlib uses these weights to normalize the bin counts, effectively converting them into proportions.

import numpy as np
import matplotlib.pyplot as plt

#define data values
data =

```
#create relative frequency histogram
fig = plt.figure()
ax = fig.add_subplot(111)
ax.hist(data, edgecolor='black', weights=np.ones_like(data) / len(data))
```



As you can observe, the **y-axis** now displays **relative frequencies** instead of raw counts. For our example, there are **15 total values** in the dataset. If a bin originally had a frequency of **4**, it will now show a relative frequency of 4/15, which is approximately **0.2667**. This representation immediately tells us the proportion of data points falling into each bin relative to the entire dataset.

Enhancing Readability: Displaying Percentages

While relative frequencies (as decimals) are informative, presenting them as percentages often enhances readability and makes the graph more intuitive for a wider audience. **Matplotlib** provides a convenient way to achieve this using the **`PercentFormatter()` function** from its `ticker` module.

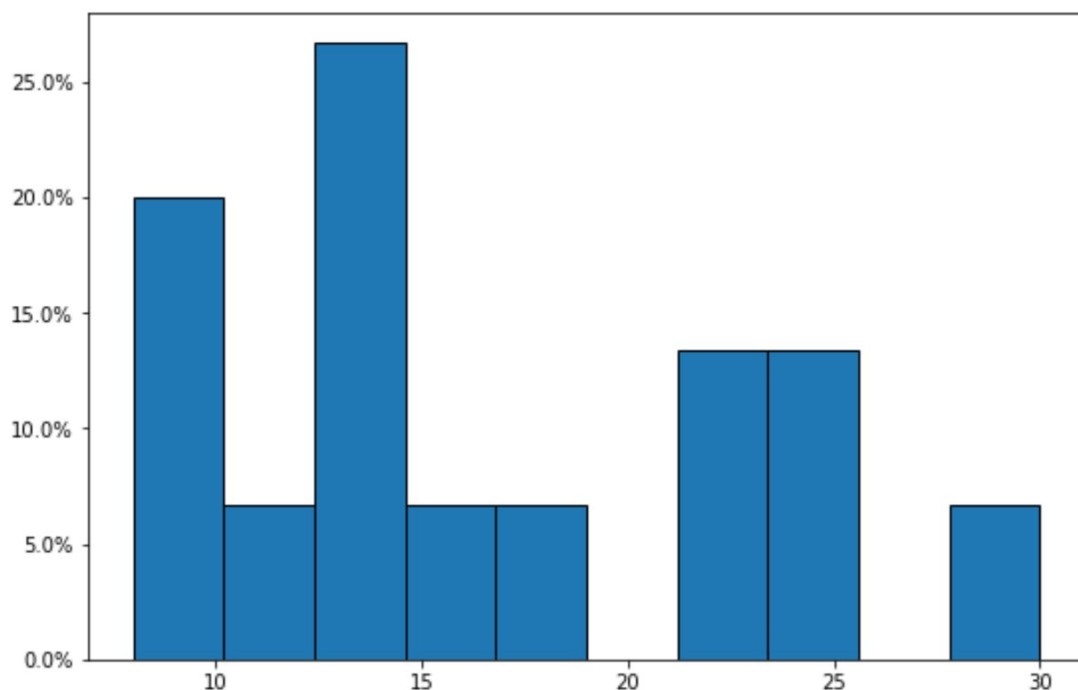
To display percentages on the y-axis, we need to make two small modifications: first, multiply our `weights` by 100 so that the values correspond to percentages, and then apply the `PercentFormatter()` to format the y-axis labels accordingly.

```
import numpy as np
import matplotlib.pyplot as plt
```

from matplotlib.ticker import PercentFormatter

```
#define data values
data =

#create relative frequency histogram with percentages on y-axis
fig = plt.figure()
ax = fig.add_subplot(111)
ax.hist(data, edgecolor='black', weights=np.ones_like(data)*100 / len(data))
ax.yaxis.set_major_formatter(PercentFormatter())
```



Notice how the **y-axis** now clearly displays the **relative frequencies** as percentages, making the interpretation of the data distribution even more straightforward. This final visualization is ideal for presentations and reports where clarity and immediate understanding are paramount.

Additional Resources for Data Visualization

Creating relative frequency histograms is just one of many ways to visualize data effectively using **Matplotlib**. For those interested in exploring further **data visualization** techniques, the library offers a vast array of plotting functionalities.

Consider delving into the following tutorials to expand your charting skills and create other common types of graphs in Matplotlib:

How to Create Stacked Bar Plots
How to Create Scatter Plots
How to Create Box Plots