

How can I implement equal frequency binning in Python?

Authored by
stats writer

April 16, 2024

RECOMMENDED CITATION

stats writer (2024). *How can I implement equal frequency binning in Python?*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=136121>

Equal frequency binning is a data preprocessing technique used to group continuous numerical data into equal-sized bins. This method is useful for handling skewed distributions and reducing the impact of outliers on the data analysis process. To implement equal frequency binning in Python, one can use the pandas library to first sort the data in ascending order. Then, the `cut()` function can be used to divide the data into desired number of bins, with the parameter "q" specifying the number of bins. This function will automatically assign data points to bins based on their frequency. Finally, the `pd.get_dummies()` function can be used to convert the bins into categorical variables for further analysis.

Equal Frequency Binning in Python

In statistics, binning is the process of placing numerical values into *bins*.

The most common form of binning is known as equal-width binning, in which we divide a dataset into k bins of equal width.

A less commonly used form of binning is known as equal-frequency binning, in which we divide a dataset into k bins that all have an equal number of frequencies.

This tutorial explains how to perform equal frequency binning in python.

Equal Frequency Binning in Python

Suppose we have a dataset that contains 100 values:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
#create data
```

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

```
data = np.random.randn(100)
```

```
#view first 5 values
```

```
data
```

```
array()
```

Equal-Width Binning:

If we create a histogram to display these values, Python will use equal-width binning by default:

```
#create histogram with equal-width bins
```

```
n, bins, patches = plt.hist(data, edgecolor='black')
```

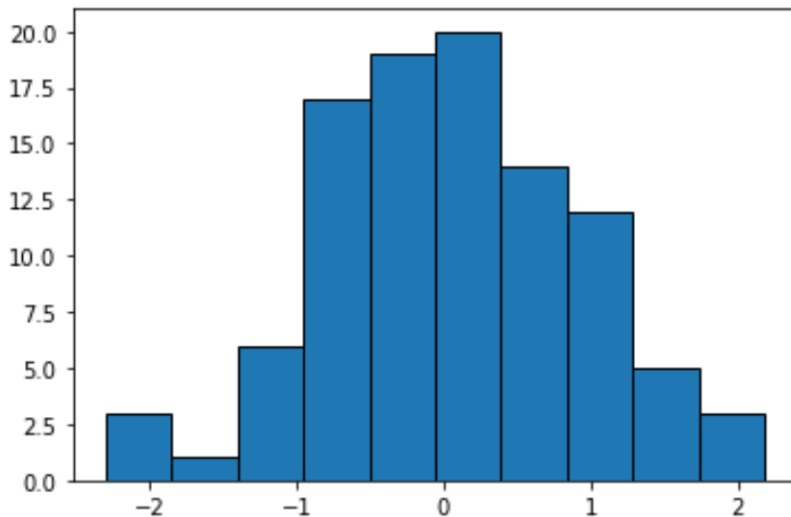
```
plt.show()
```

```
#display bin boundaries and frequency per bin
```

```
bins, n
```

```
(array(),
```

```
array())
```



Each bin has an equal width of approximately .4487, but each bin doesn't contain an equal amount of observations. For example:

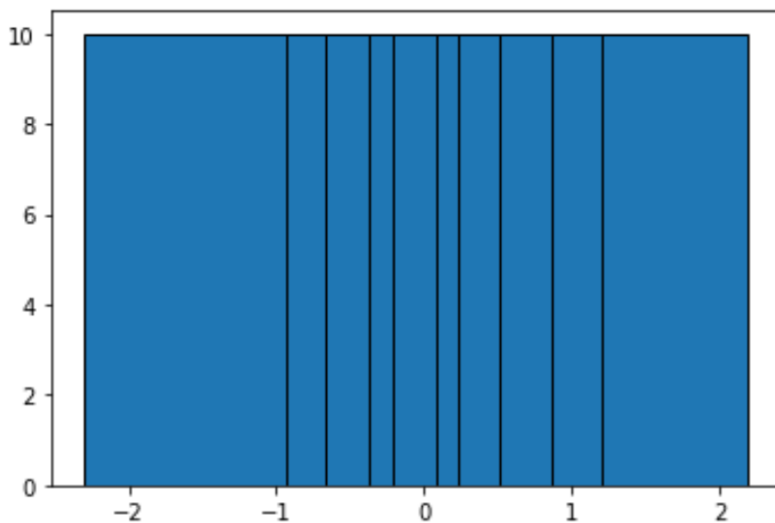
The first bin extends from -2.3015387 to -1.8528279 and contains 3 observations. The second bin extends from -1.8528279 to -1.40411588 and contains 1 observation. The third bin extends from -1.40411588 to -0.95540447 and contains 6 observations.

And so on.

Equal-Frequency Binning:

To create bins that contain an equal number of observations, we can use the following function:

```
#define function to calculate equal-frequency bins  
def equalObs(x, nbin):  
    nlen = len(x)  
    return np.interp(np.linspace(0, nlen, nbin + 1),  
                    np.arange(nlen),  
                    np.sort(x))  
  
#create histogram with equal-frequency bins  
n, bins, patches = plt.hist(data, equalObs(data, 10),  
                             edgecolor='black')  
plt.show()  
  
#display bin boundaries and frequency per bin  
bins, n  
  
(array(),  
array())
```



Each bin doesn't have an equal width, but each bin does contain an equal amount of observations. For example:

The first bin extends from -2.3015387 to -0.93576943 and contains 10 observations. The second bin extends from -0.93576943 to -0.67124613 and contains 10 observations. The third bin extends from -0.67124613 to -0.37528495 and contains 10 observations.

And so on.

We can see from the histogram that each bin is clearly not the same width, but each bin does contain the same amount of observations which is confirmed by the fact that each bin height is equal.