options available in these functions, refer to their docstrings. If you are interested in three-dimensional visualizations of this type of data, see "Three-Dimensional Plotting in Matplotlib" on page 290.

Histograms, Binnings, and Density

A simple histogram can be a great first step in understanding a dataset. Earlier, we saw a preview of Matplotlib's histogram function (see "Comparisons, Masks, and Boolean Logic" on page 70), which creates a basic histogram in one line, once the normal boilerplate imports are done (Figure 4-35):

```
In[1]: %matplotlib inline
       import numpy as np
       import matplotlib.pyplot as plt
       plt.style.use('seaborn-white')

       data = np.random.randn(1000)

In[2]: plt.hist(data);
```

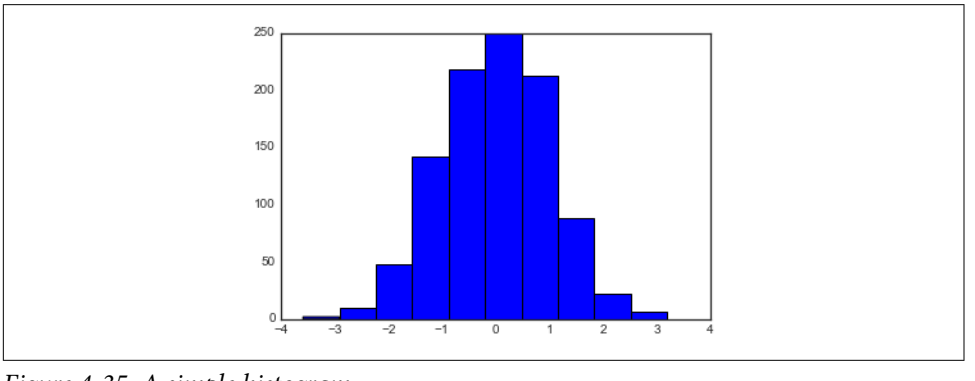

Figure 4-35. A simple histogram

The `hist()` function has many options to tune both the calculation and the display; here's an example of a more customized histogram (Figure 4-36):

```
In[3]: plt.hist(data, bins=30, normed=True, alpha=0.5,
                histtype='stepfilled', color='steelblue',
                edgecolor='none');
```

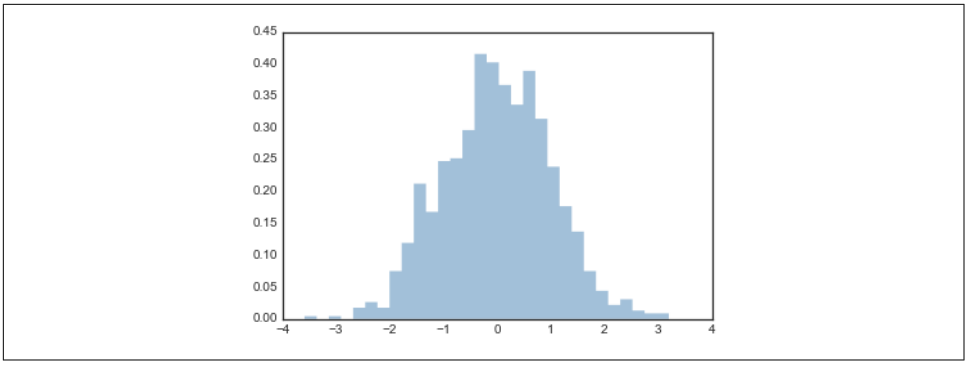

Figure 4-36. A customized histogram

The `plt.hist` docstring has more information on other customization options available. I find this combination of `histtype='stepfilled'` along with some transparency `alpha` to be very useful when comparing histograms of several distributions (Figure 4-37):

```
In[4]: x1 = np.random.normal(0, 0.8, 1000)
       x2 = np.random.normal(-2, 1, 1000)
       x3 = np.random.normal(3, 2, 1000)

       kwargs = dict(histtype='stepfilled', alpha=0.3, normed=True, bins=40)

       plt.hist(x1, **kwargs)
       plt.hist(x2, **kwargs)
       plt.hist(x3, **kwargs);
```

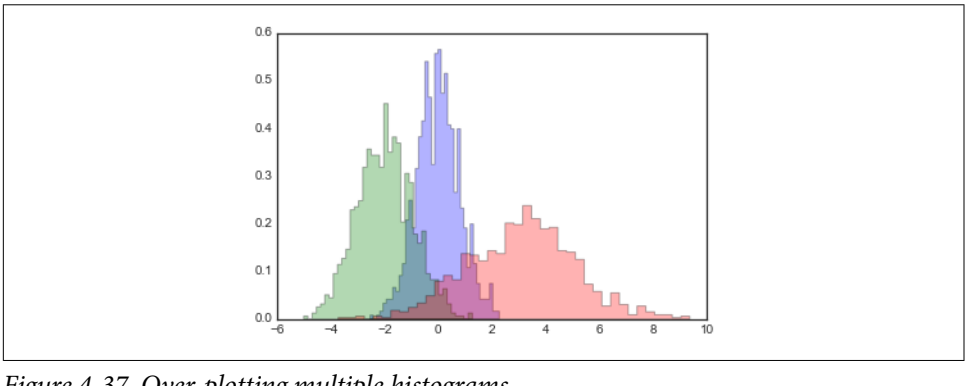

Figure 4-37. Over-plotting multiple histograms

If you would like to simply compute the histogram (that is, count the number of points in a given bin) and not display it, the `np.histogram()` function is available:

```
In[5]: counts, bin_edges = np.histogram(data, bins=5)
       print(counts)
```

```
[ 12 190 468 301  29]
```

Two-Dimensional Histograms and Binnings

Just as we create histograms in one dimension by dividing the number line into bins, we can also create histograms in two dimensions by dividing points among two-dimensional bins. We'll take a brief look at several ways to do this here. We'll start by defining some data—an `x` and `y` array drawn from a multivariate Gaussian distribution:

```
In[6]: mean = [0, 0]
       cov = [[1, 1], [1, 2]]
       x, y = np.random.multivariate_normal(mean, cov, 10000).T
```

plt.hist2d: Two-dimensional histogram

One straightforward way to plot a two-dimensional histogram is to use Matplotlib's `plt.hist2d` function (Figure 4-38):

```
In[12]: plt.hist2d(x, y, bins=30, cmap='Blues')
        cb = plt.colorbar()
        cb.set_label('counts in bin')
```

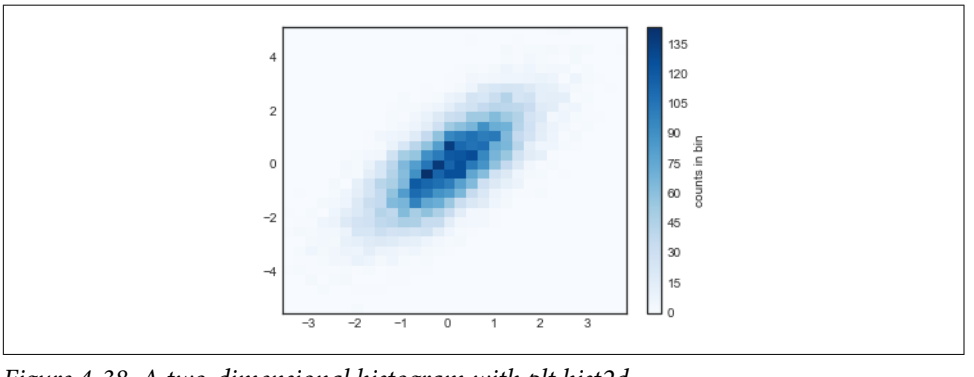

Figure 4-38. A two-dimensional histogram with plt.hist2d

Just as with `plt.hist`, `plt.hist2d` has a number of extra options to fine-tune the plot and the binning, which are nicely outlined in the function docstring. Further, just as `plt.hist` has a counterpart in `np.histogram`, `plt.hist2d` has a counterpart in `np.histogram2d`, which can be used as follows:

```
In[8]: counts, xedges, yedges = np.histogram2d(x, y, bins=30)
```

For the generalization of this histogram binning in dimensions higher than two, see the `np.histogramdd` function.