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, which creates a **basic histogram in one line,** once the normal boilerplate imports are done (see the following figure):

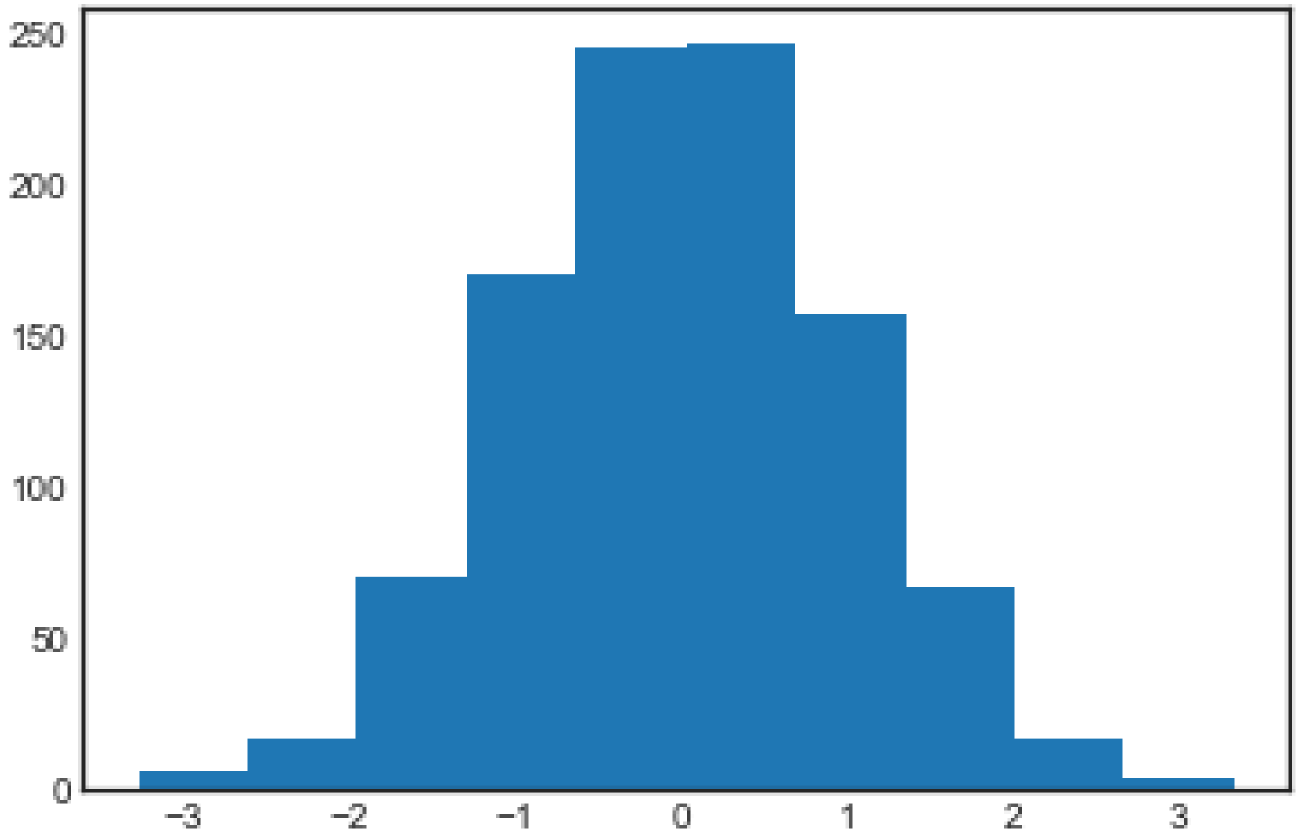
In [ ]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')

rng = np.random.default_rng(1701)
data = rng.normal(size=1000)
```

In [ ]:

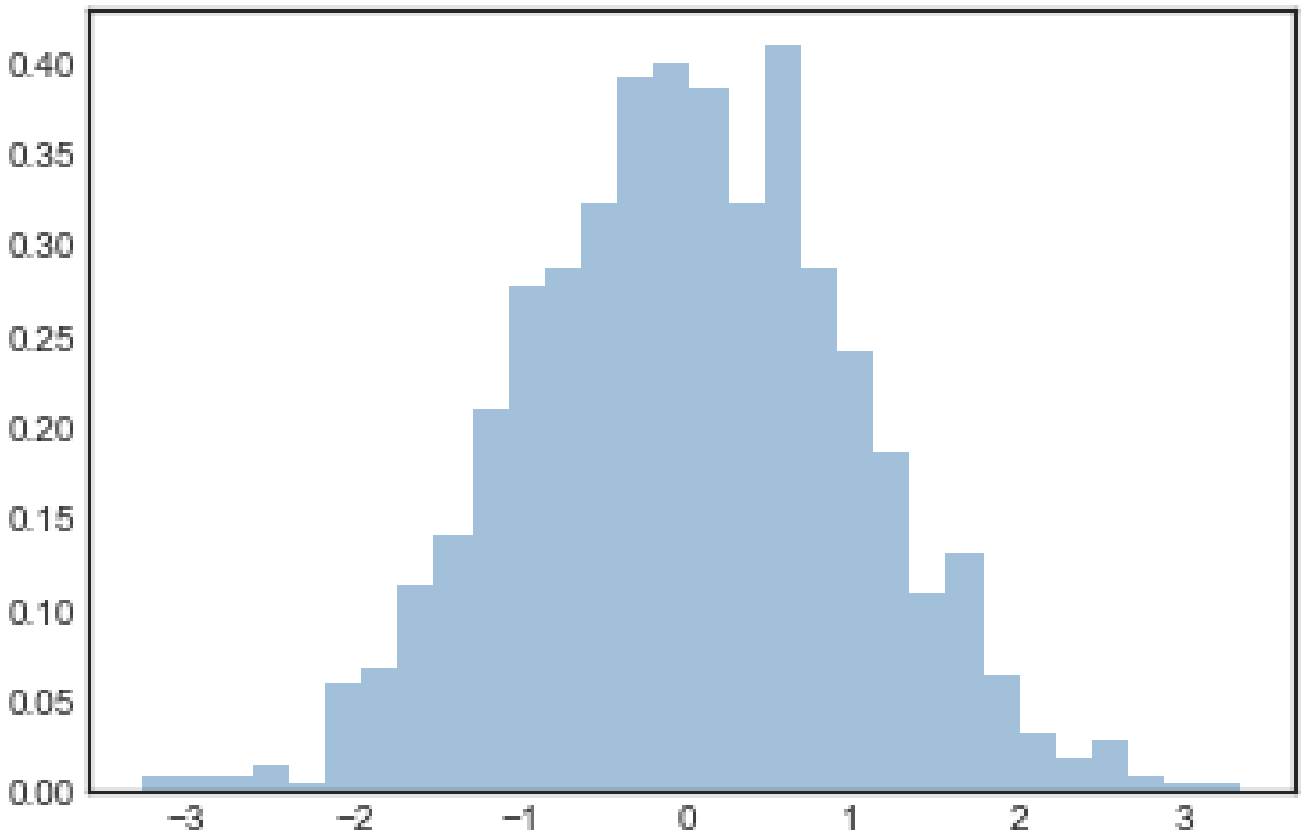
```
plt.hist(data);
```



The `hist` function has many **options** to tune both the calculation and the display;

here's an **example of a more customized histogram,** shown in the following figure:

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



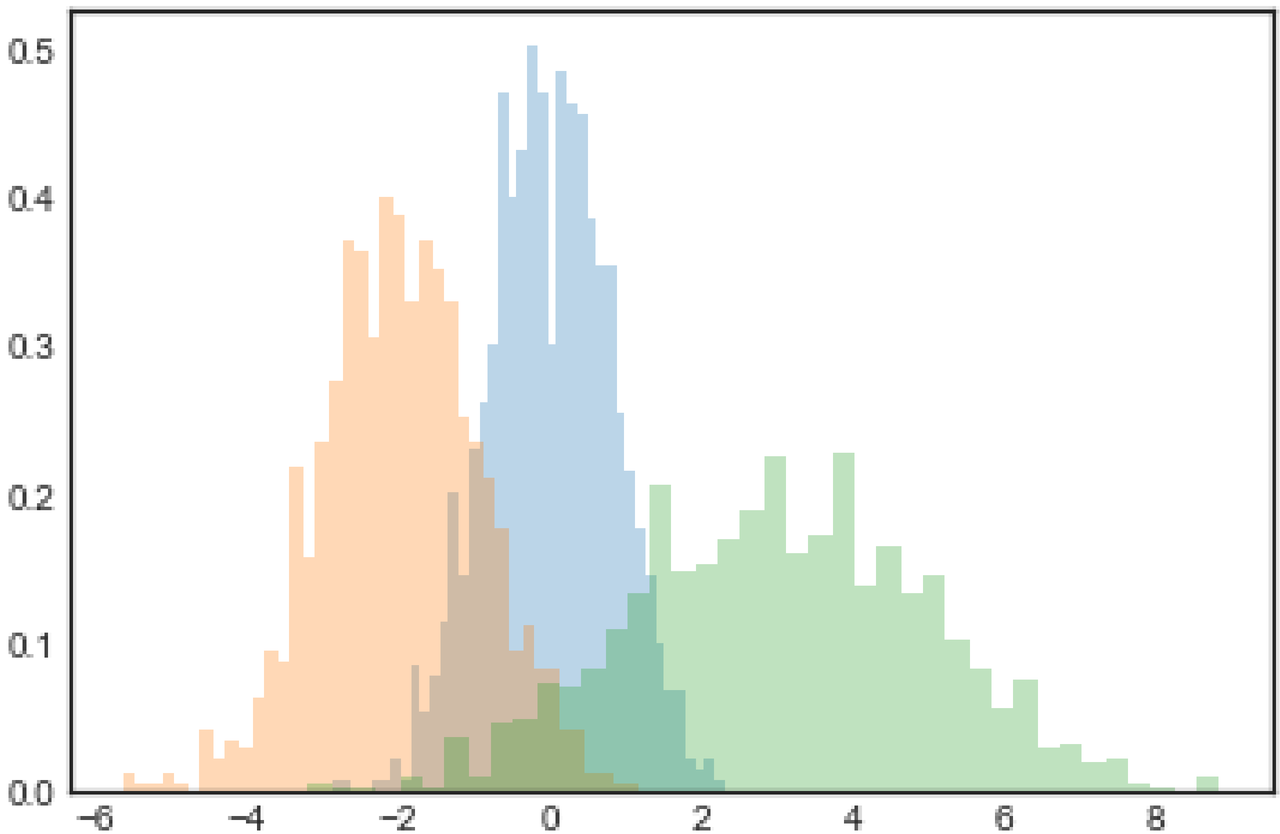
Combination of `histtype='stepfilled'` along with some transparency `alpha` is helpful when comparing histograms of several distributions (see the following figure):

In [ ]:

```
x1 = rng.normal(0, 0.8, 1000)
x2 = rng.normal(-2, 1, 1000)
x3 = rng.normal(3, 2, 1000)

kwargs = dict(histtype='stepfilled', alpha=0.3, density=True,

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



If you are **interested in computing,** but not displaying, the histogram, you can use the `np.histogram` function:

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

```
[ 23 241 491 224  21]
```

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:**

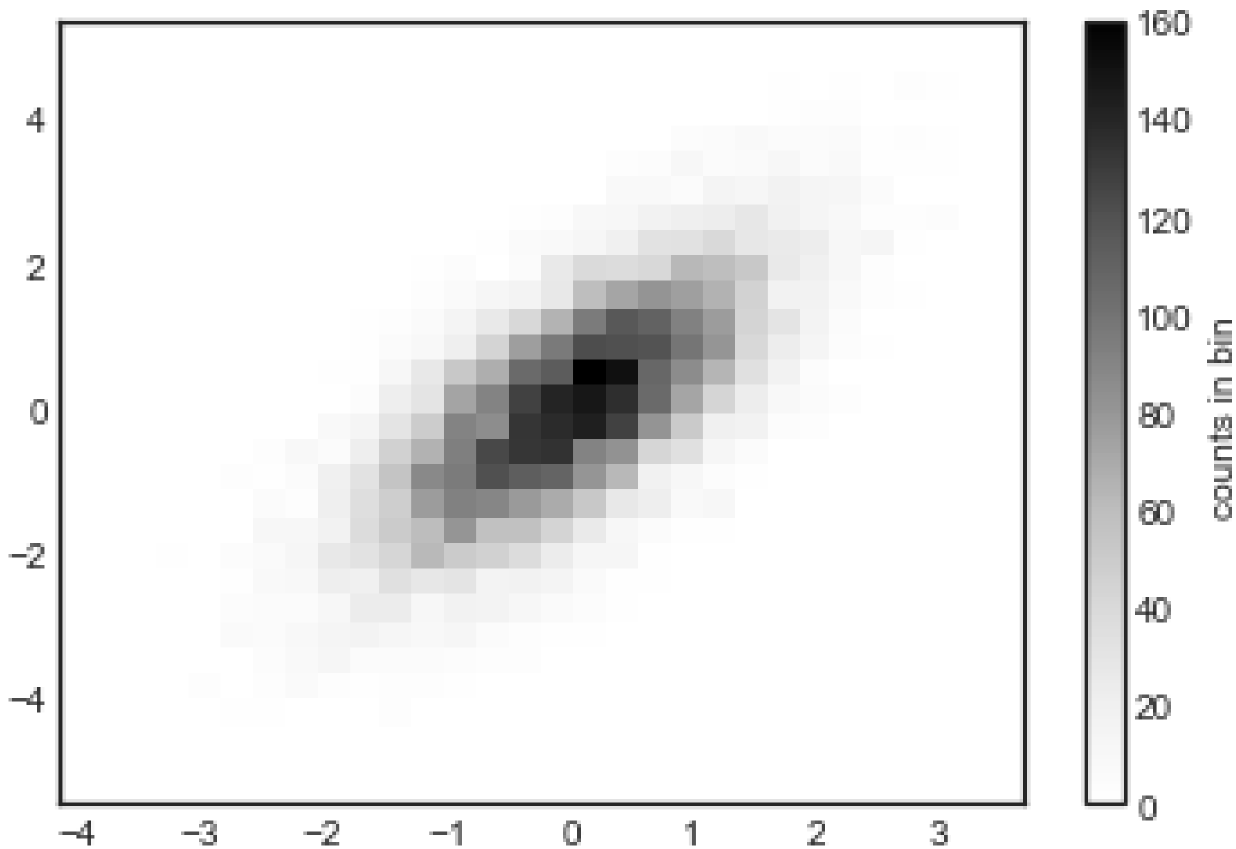
In [ ]:

```
mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = rng.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 (see the following figure):

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



Just like `plt.hist`, `plt.hist2d` has a number of **extra options** to fine-tune the plot and the binning.

Further, just as `plt.hist` has a **counterpart** in `np.histogram`, `plt.hist2d` has a counterpart in `np.histogram2d`:

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

```
(30, 30)
```

plt.hexbin: Hexagonal binnings

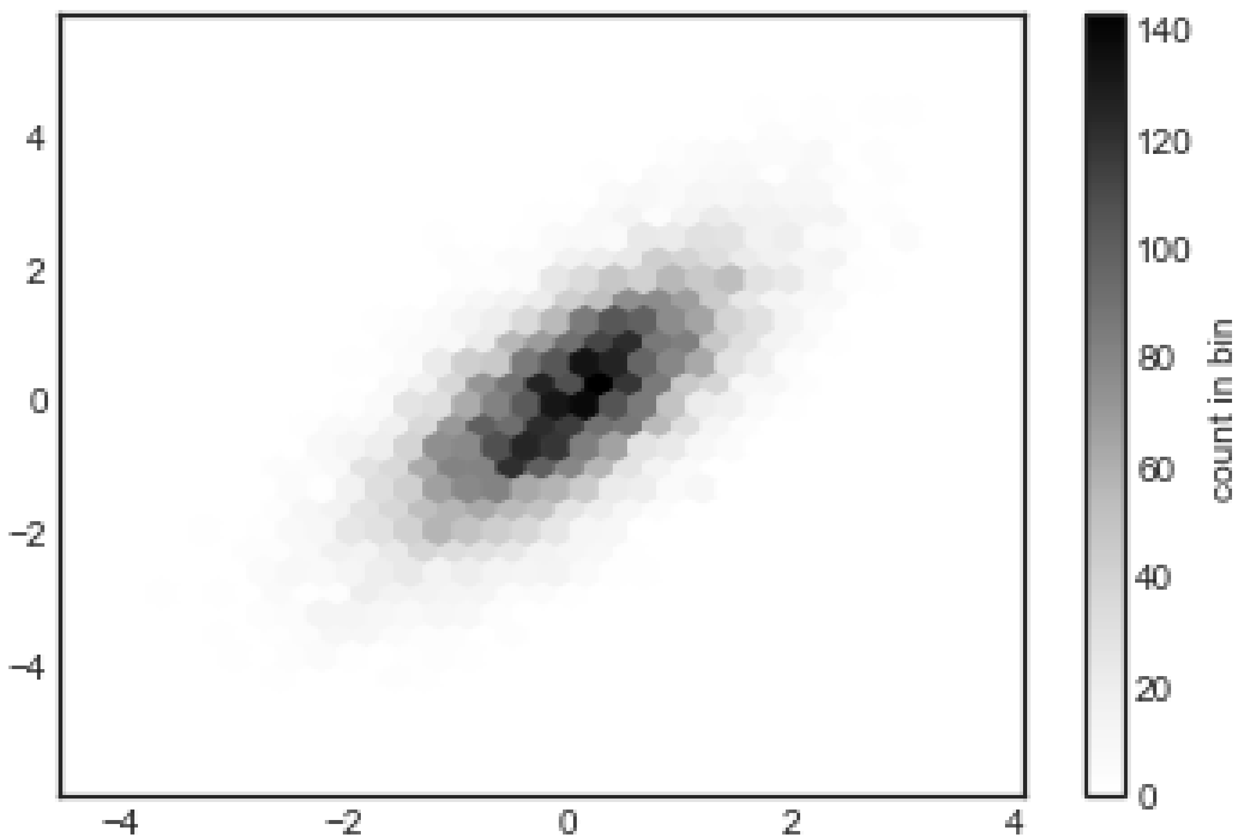
The two-dimensional histogram creates a **tesselation of squares** across the axes.

Another natural shape for such a tesselation is the regular **hexagon.**

For this purpose, Matplotlib provides the `plt.hexbin` routine,

which represents a two-dimensional dataset binned within a **grid of hexagons** (see the following figure):

In [ ]:
```
plt.hexbin(x, y, gridsize=30)
cb = plt.colorbar(label='count in bin')
```



Kernel density estimation

Another common method for estimating and representing densities in multiple dimensions is **kernel density estimation** (KDE).

KDE can be thought of as a way to "smear out" the points in space and add up the result to obtain a **smooth function.**

One extremely quick and simple **KDE implementation** exists in the `scipy.stats` package.

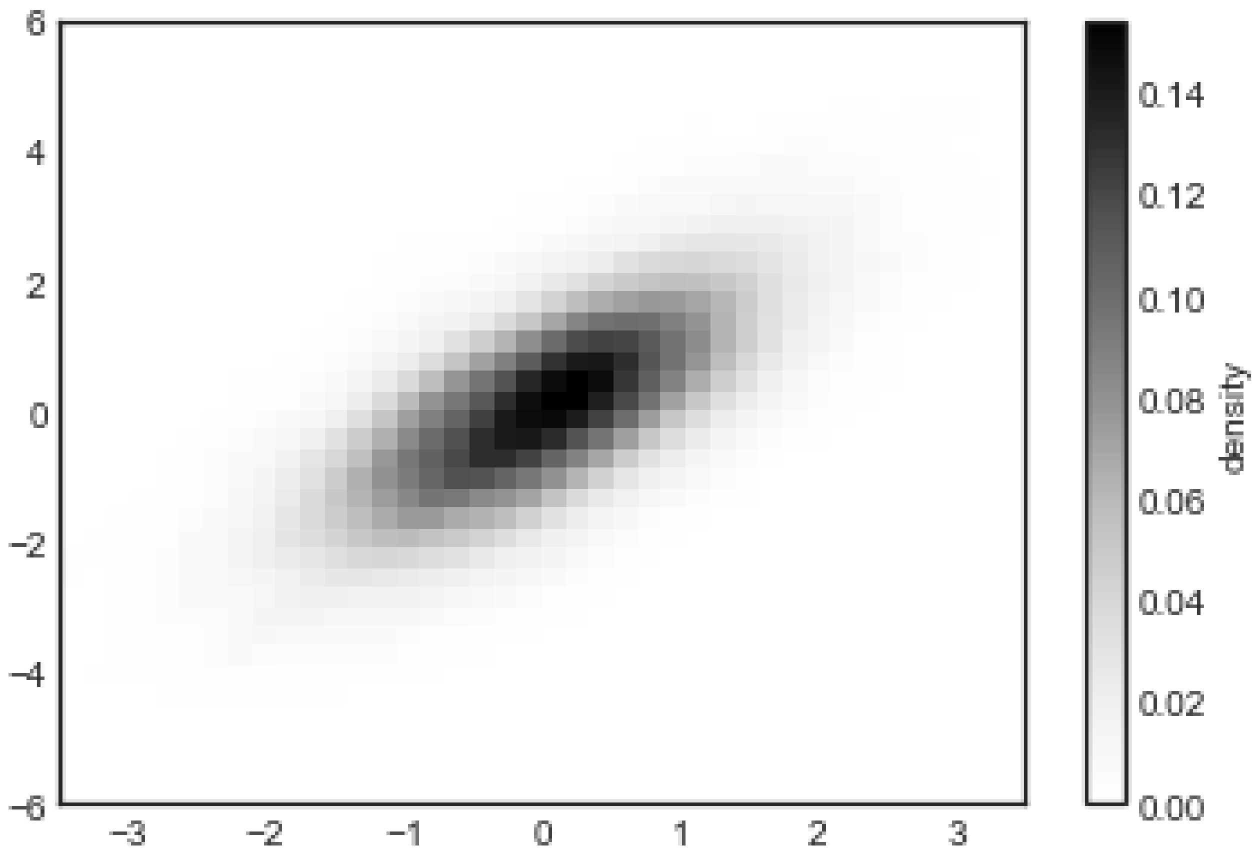
Here is a quick example of using KDE (see the following figure):

In [ ]:
```
from scipy.stats import gaussian_kde

# fit an array of size [Ndim, Nsamples]
data = np.vstack([x, y])
kde = gaussian_kde(data)
```

```python
# evaluate on a regular grid
xgrid = np.linspace(-3.5, 3.5, 40)
ygrid = np.linspace(-6, 6, 40)
Xgrid, Ygrid = np.meshgrid(xgrid, ygrid)
Z = kde.evaluate(np.vstack([Xgrid.ravel(), Ygrid.ravel()]))

# Plot the result as an image
plt.imshow(Z.reshape(Xgrid.shape),
           origin='lower', aspect='auto',
           extent=[-3.5, 3.5, -6, 6])
cb = plt.colorbar()
cb.set_label("density")
```



6
4
2
0
−2
−4
−6
−3
−2
−1
0
1
2
3
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
density