Visualizing the distribution of a dataset

When dealing with a set of data, often the first thing you'll want to do is get a sense for how the variables are distributed. This chapter of the tutorial will give a brief introduction to some of the tools in seborn for examining univariate and bivariate distributions. You may also want to look at the categorical plots (categorical.html#categorical-tutorial) chapter for examples of functions that make it easy to compare the distribution of a variable across levels of other variables.

```
%matplotlib inline
```

```
import numpy as np
import pandas as pd
from scipy import stats, integrate
import matplotlib.pyplot as plt
```

```
import seaborn as sns
sns.set(color_codes=True)
```

```
np.random.seed(sum(map(ord, "distributions")))
```

Plotting univariate distributions

The most convenient way to take a quick look at a univariate distribution in seaborn is the `distplot()` (../generated/seaborn.distplot.html#seaborn.distplot) function. By default, this will draw a histogram (https://en.wikipedia.org/wiki/Histogram) and fit a kernel density estimate (https://en.wikipedia.org/wiki/Kernel_density_estimation) (KDE).

```
x = np.random.normal(size=100)
sns.distplot(x);
```
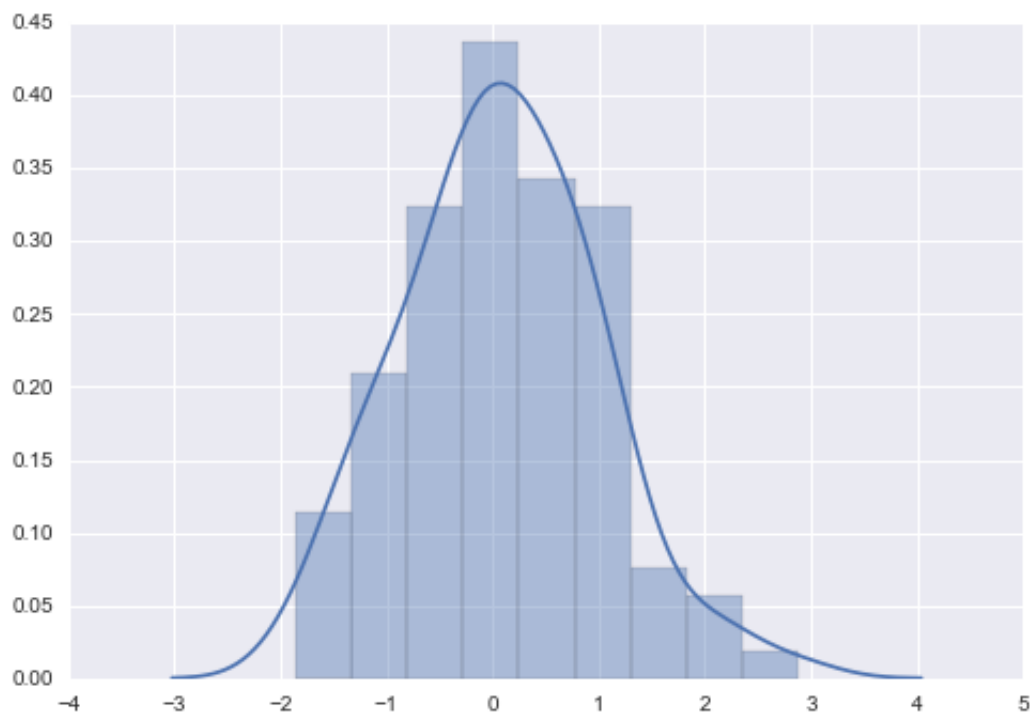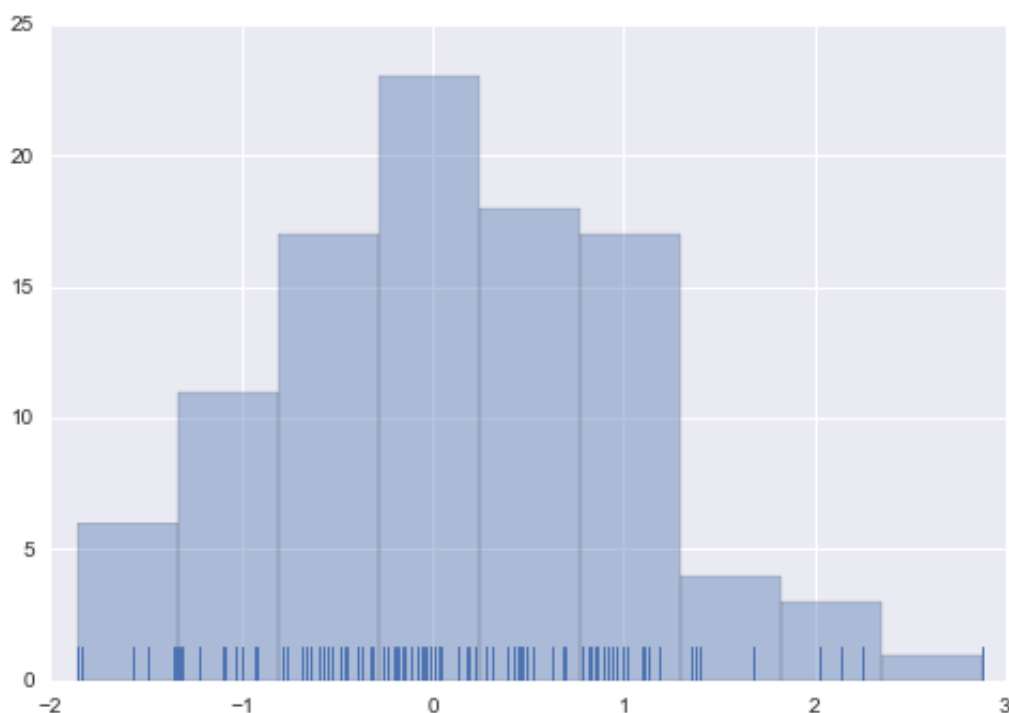
Histograms

Histograms are likely familiar, and a `hist` function already exists in matplotlib. A histogram represents the distribution of data by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin.

To illustrate this, let's remove the density curve and add a rug plot, which draws a small vertical tick at each observation. You can make the rug plot itself with the `rugplot()` (../generated/seaborn.rugplot.html#seaborn.rugplot) function, but it is also available in `distplot()` (../generated/seaborn.distplot.html#seaborn.distplot):

```
sns.distplot(x, kde=False, rug=True);
```

When drawing histograms, the main choice you have is the number of bins to use and where to place them. `distplot()` (../generated/seaborn.distplot.html#seaborn.distplot) uses a simple rule to make a good guess for what the right number is by default, but trying more or fewer bins might reveal other features in the data:
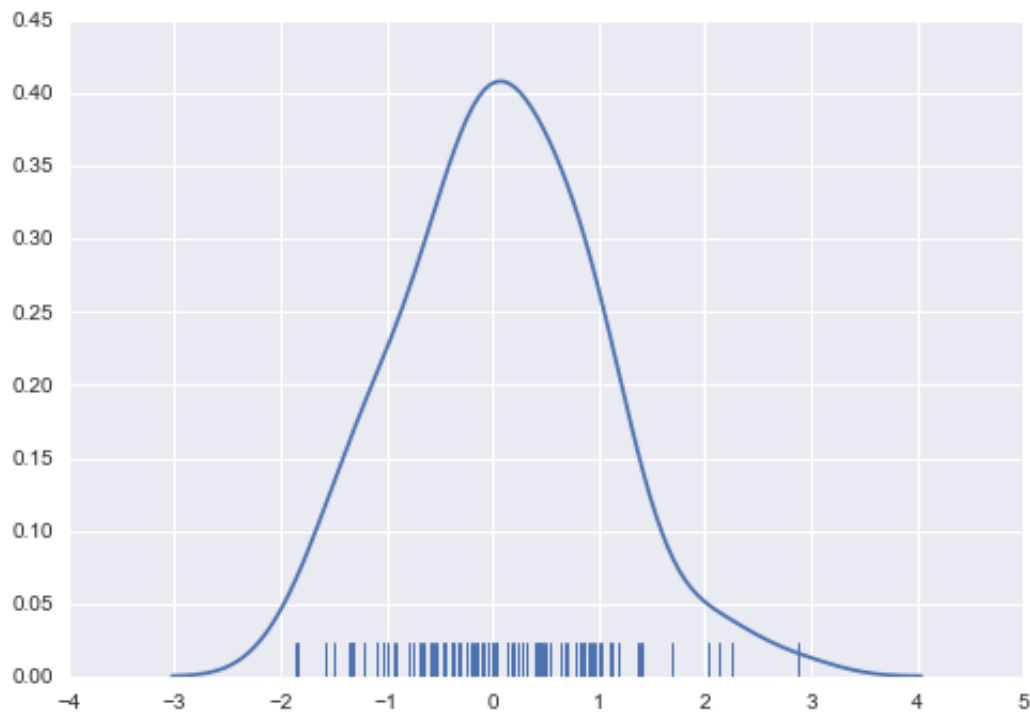
```
sns.distplot(x, bins=20, kde=False, rug=True);
```
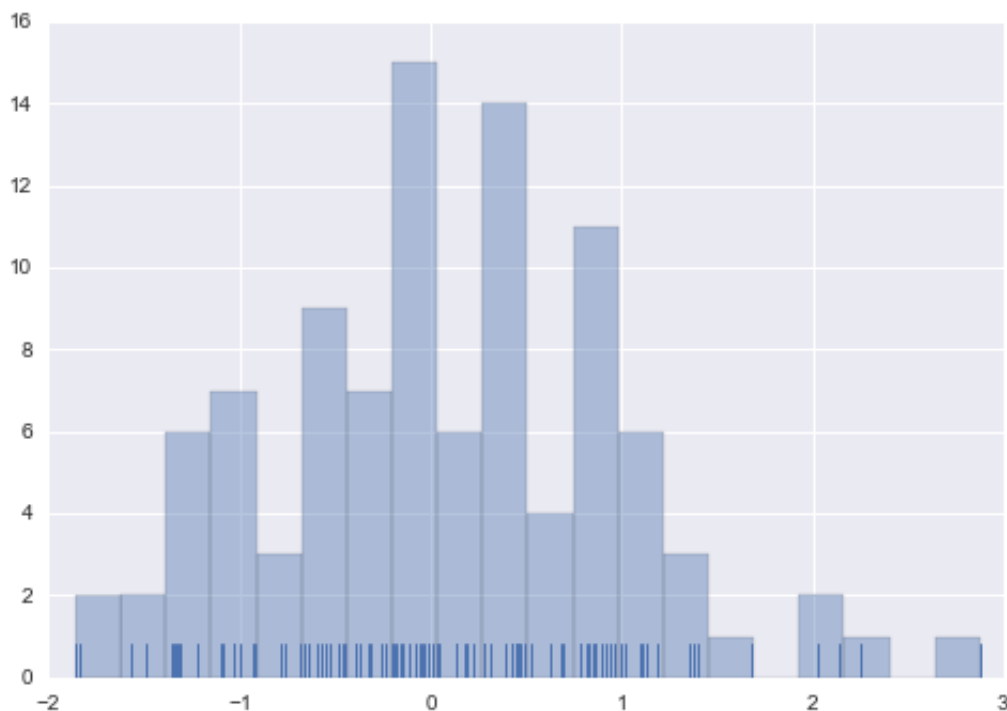

Kernel density estimaton

The kernel density estimate may be less familiar, but it can be a useful tool for plotting the shape of a distribution. Like the histogram, the KDE plots encodes the density of observations on one axis with height along the other axis:

```
sns.distplot(x, hist=False, rug=True);
```

Drawing a KDE is more computationally involved than drawing a histogram. What happens is that each observation is first replaced with a normal (Gaussian) curve centered at that value:

```
x = np.random.normal(0, 1, size=30)
bandwidth = 1.06 * x.std() * x.size ** (-1 / 5.)
support = np.linspace(-4, 4, 200)

kernels = []
for x_i in x:

    kernel = stats.norm(x_i, bandwidth).pdf(support)
    kernels.append(kernel)
    plt.plot(support, kernel, color="r")

sns.rugplot(x, color=".2", linewidth=3);
```
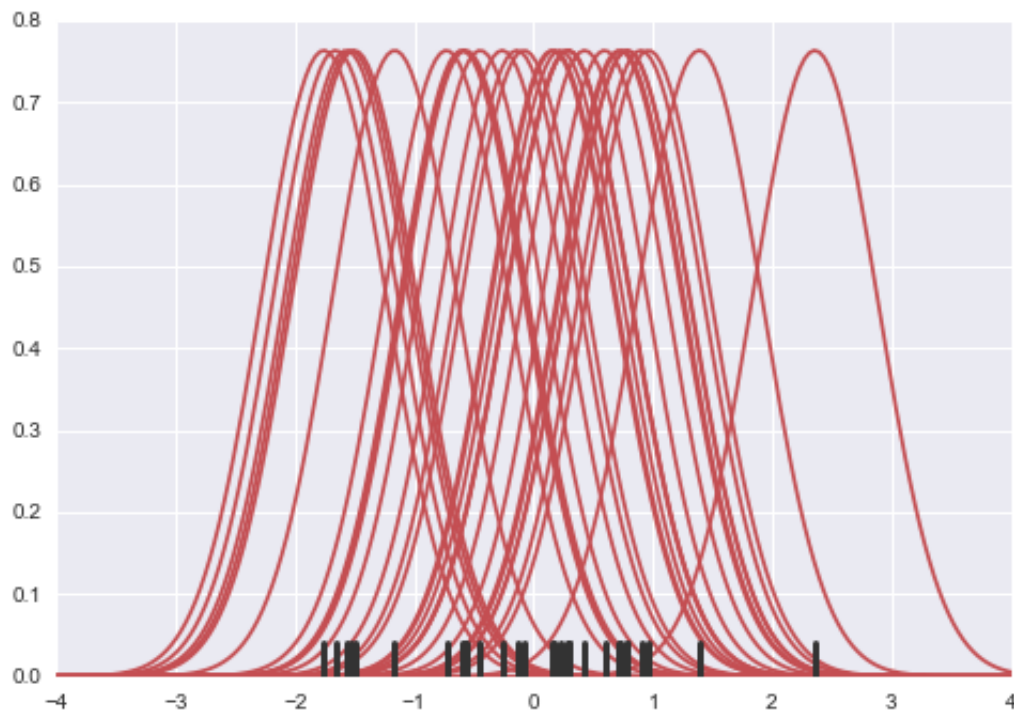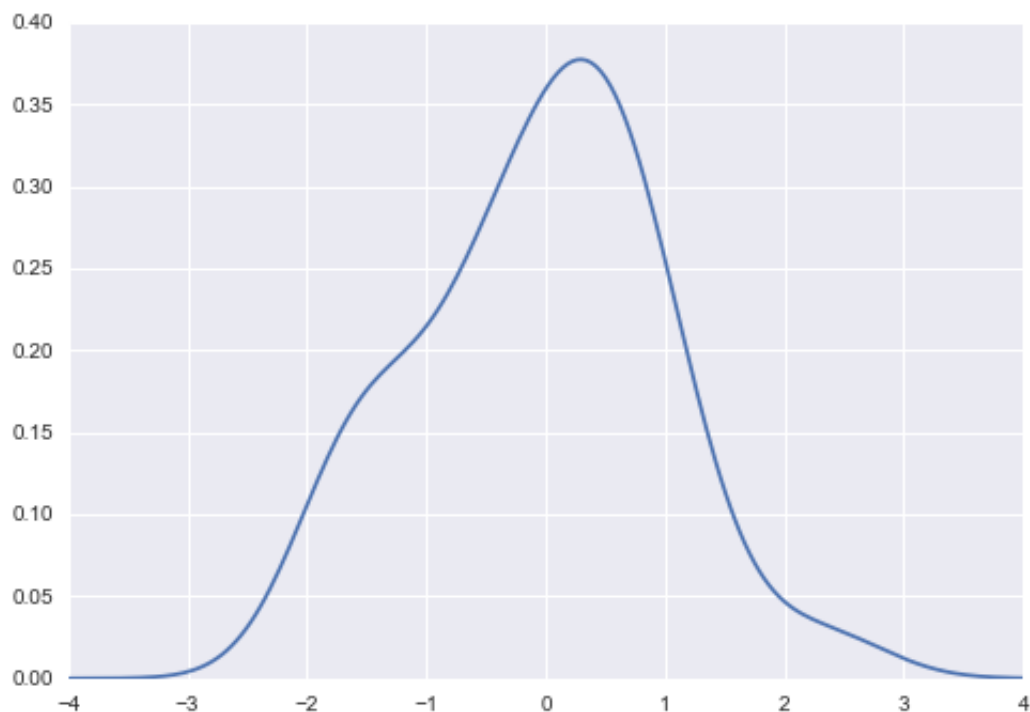
Next, these curves are summed to compute the value of the density at each point in the support grid. The resulting curve is then normalized so that the area under it is equal to 1:

```
density = np.sum(kernels, axis=0)
density /= integrate.trapz(density, support)
plt.plot(support, density);
```


We can see that if we use the `kdeplot()` (../generated/seaborn.kdeplot.html#seaborn.kdeplot) function in seaborn, we get the same curve. This function is used by `distplot()` (../generated/seaborn.distplot.html#seaborn.distplot), but it provides a more direct interface with easier access to other options when you just want the density estimate:

```
sns.kdeplot(x, shade=True);
```
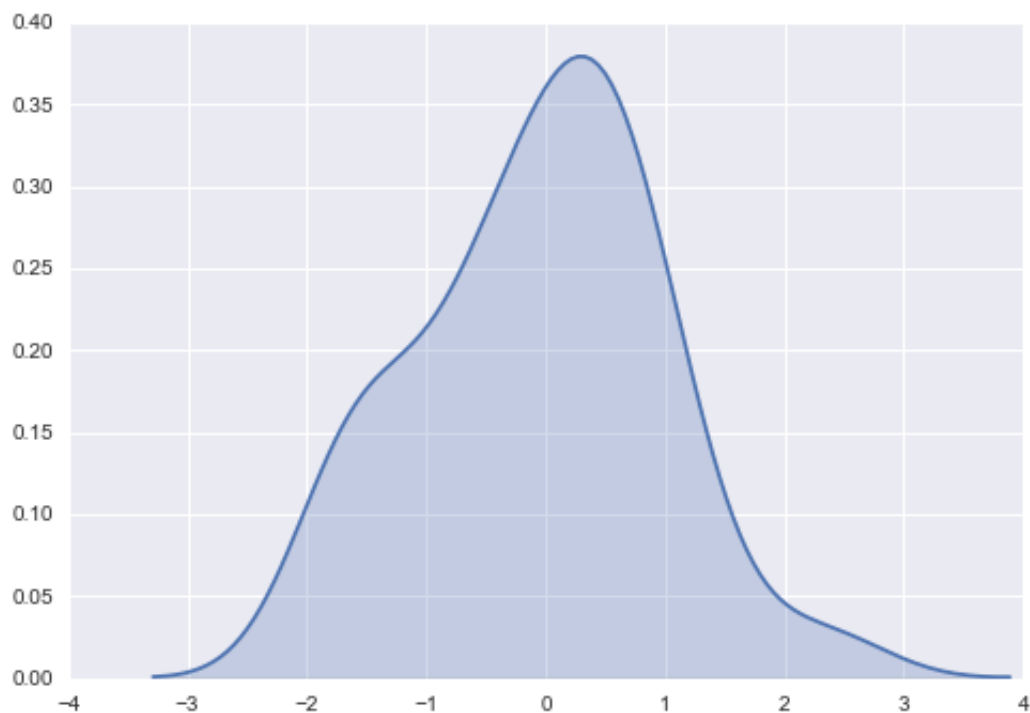
The bandwidth (bw) parameter of the KDE controls how tightly the estimation is fit to the data, much like the bin size in a histogram. It corresponds to the width of the kernels we plotted above. The default behavior tries to guess a good value using a common reference rule, but it may be helpful to try larger or smaller values:

```
sns.kdeplot(x)
sns.kdeplot(x, bw=.2, label="bw: 0.2")
sns.kdeplot(x, bw=2, label="bw: 2")
plt.legend();
```
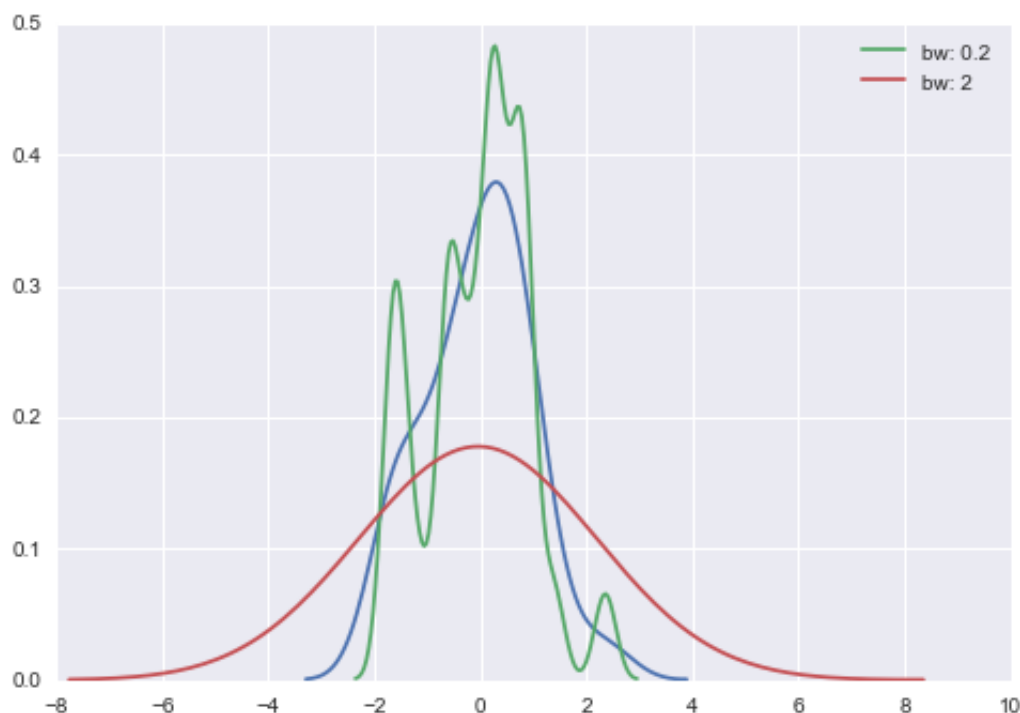

As you can see above, the nature of the Gaussian KDE process means that estimation extends past the largest and smallest values in the dataset. It's possible to control how far past the extreme values the curve is drawn with the cut parameter; however, this only influences how the curve is drawn and not how it is fit:
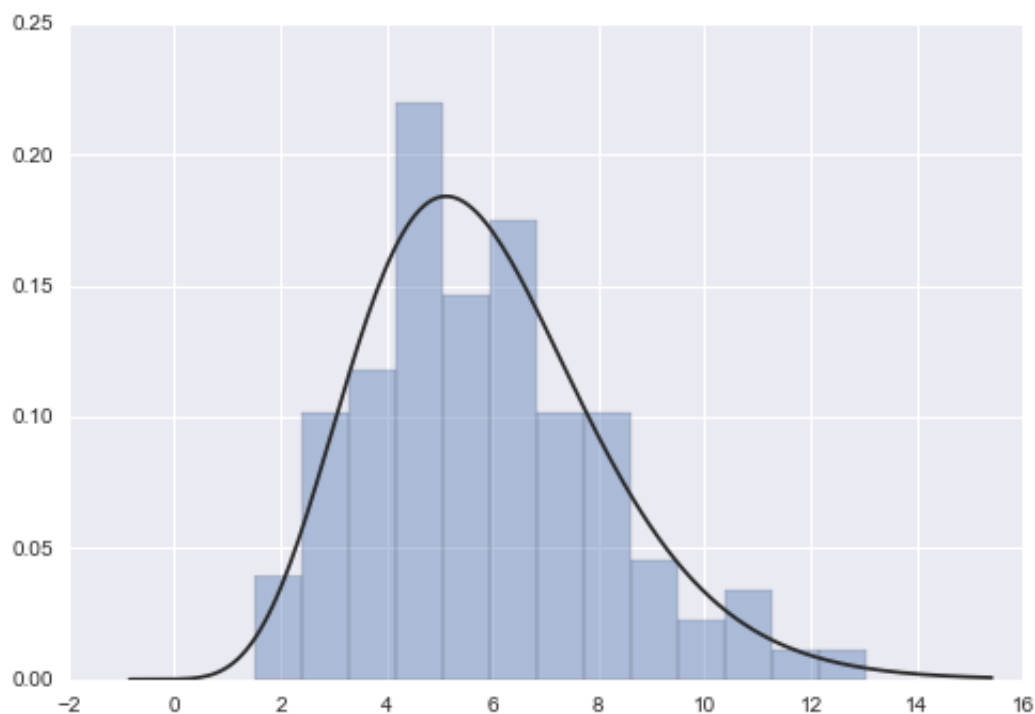
```
sns.kdeplot(x, shade=True, cut=0)
sns.rugplot(x);
```
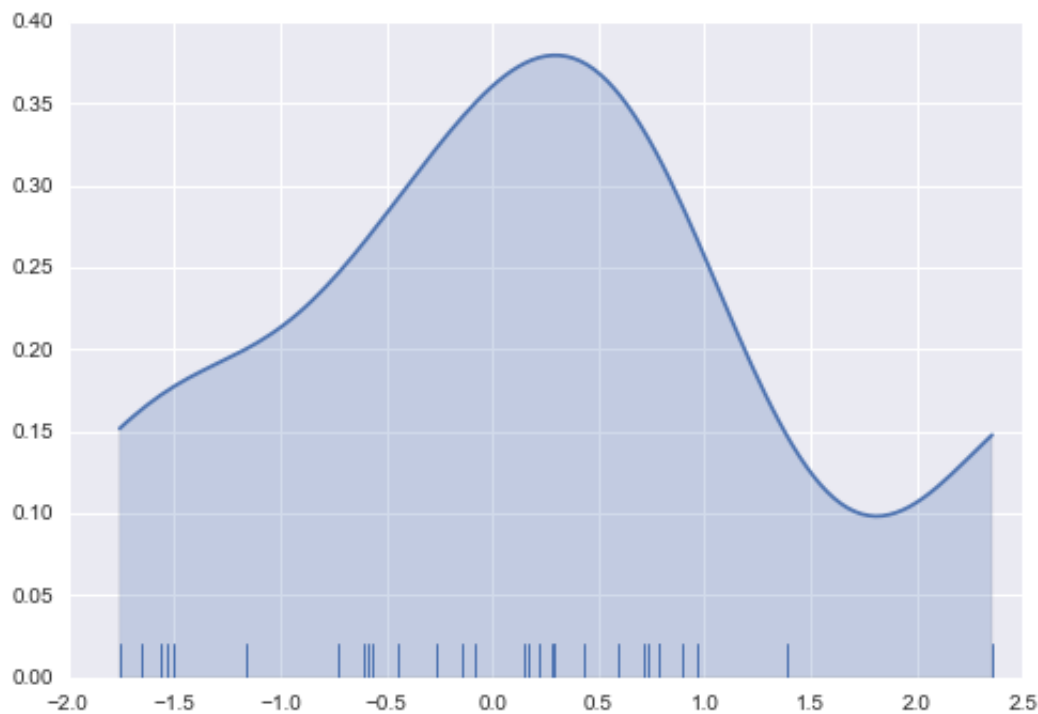

Fitting parametric distributions

You can also use `distplot()` (../generated/seaborn.distplot.html#seaborn.distplot) to fit a
parametric distribution to a dataset and visually evaluate how closely it corresponds to the
observed data:

```
x = np.random.gamma(6, size=200)
sns.distplot(x, kde=False, fit=stats.gamma);
```

Plotting bivariate distributions

It can also be useful to visualize a bivariate distribution of two variables. The easiest way to do this in seaborn is to just the `jointplot()` (../generated/seaborn.jointplot.html#seaborn.jointplot) function, which creates a multi-panel figure that shows both the bivariate (or joint) relationship between two variables along with the univariate (or marginal) distribution of each on separate axes.

```
mean, cov = [0, 1], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
```

Scatterplots

The most familiar way to visualize a bivariate distribution is a scatterplot, where each observation is shown with point at the x and y values. This is analgous to a rug plot on two dimensions. You can draw a scatterplot with the matplotlib `plt.scatter` function, and it is also the default kind of plot shown by the `jointplot()` (../generated/seaborn.jointplot.html#seaborn.jointplot) function:

```
sns.jointplot(x="x", y="y", data=df);
```
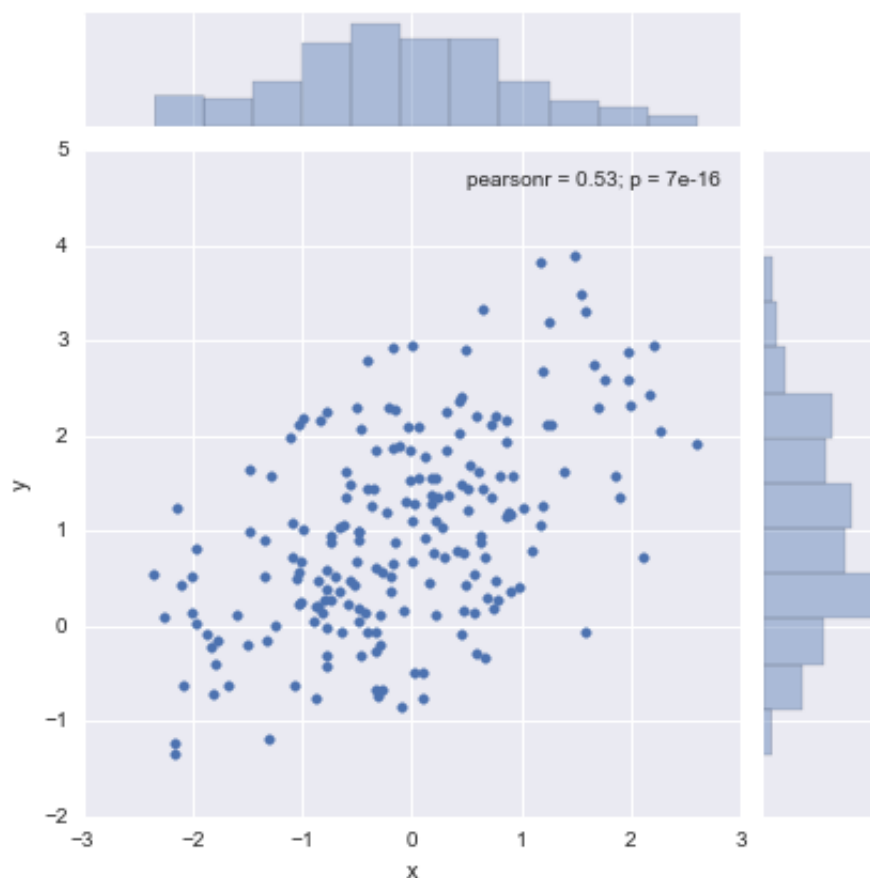

Hexbin plots

The bivariate analogue of a histogram is known as a "hexbin" plot, because it shows the counts of observations that fall within hexagonal bins. This plot works best with relatively large datasets. It's availible through the matplotlib `plt.hexbin` function and as a style in `jointplot()` (../generated/seaborn.jointplot.html#seaborn.jointplot). It looks best with a white background:

```
x, y = np.random.multivariate_normal(mean, cov, 1000).T
with sns.axes_style("white"):
    sns.jointplot(x=x, y=y, kind="hex", color="k");
```
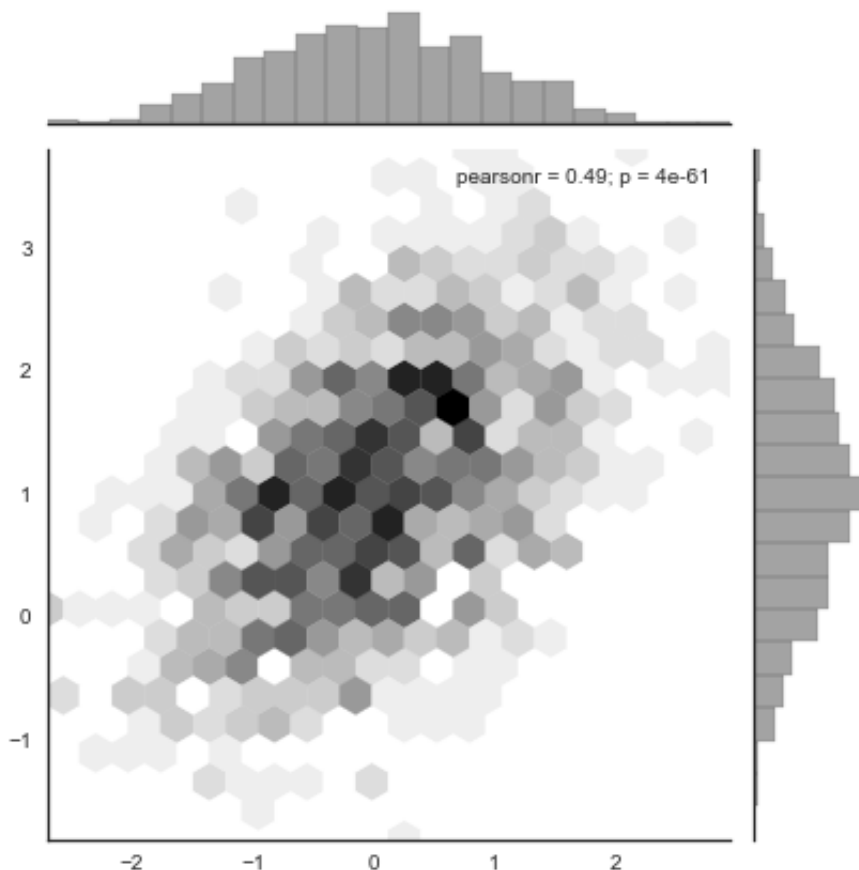

Kernel density estimation

It is also posible to use the kernel density estimation procedure described above to visualize a bivariate distribution. In seaborn, this kind of plot is shown with a contour plot and is available as a style in `jointplot()` (../generated/seaborn.jointplot.html#seaborn.jointplot):

```
sns.jointplot(x="x", y="y", data=df, kind="kde");
```
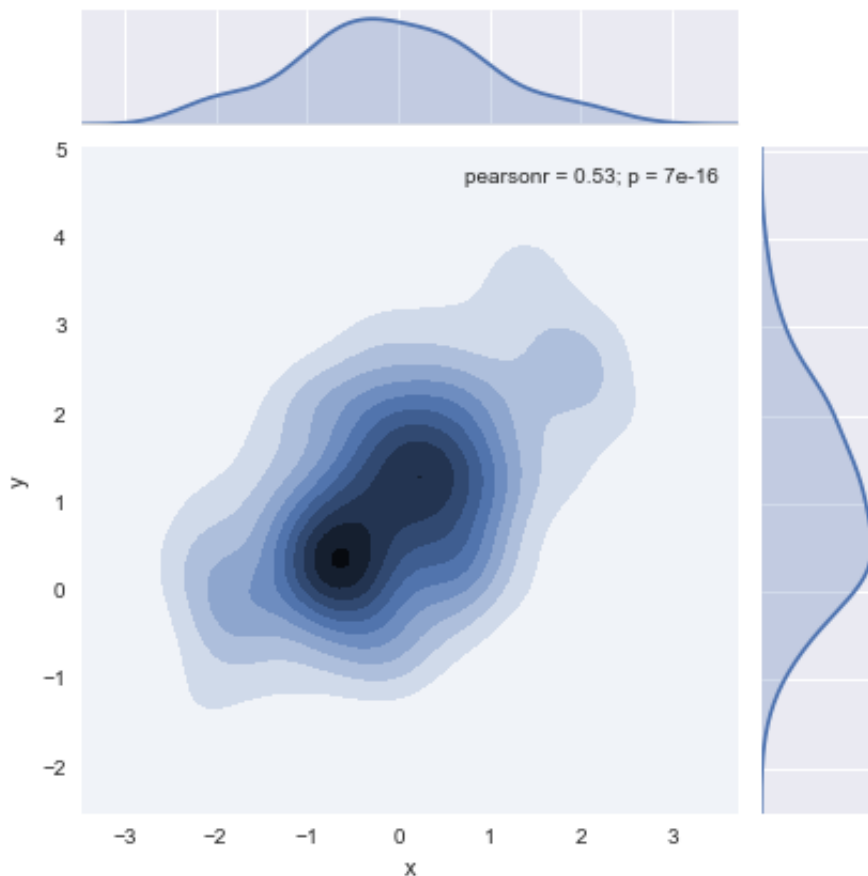
You can also draw a two-dimensional kernel density plot with the `kdeplot()` (../generated/seaborn.kdeplot.html#seaborn.kdeplot) function. This allows you to draw this kind of plot onto a specific (and possibly already existing) matplotlib axes, whereas the `jointplot()` (../generated/seaborn.jointplot.html#seaborn.jointplot) function manages its own figure:

```
f, ax = plt.subplots(figsize=(6, 6))
sns.kdeplot(df.x, df.y, ax=ax)
sns.rugplot(df.x, color="g", ax=ax)
sns.rugplot(df.y, vertical=True, ax=ax);
```
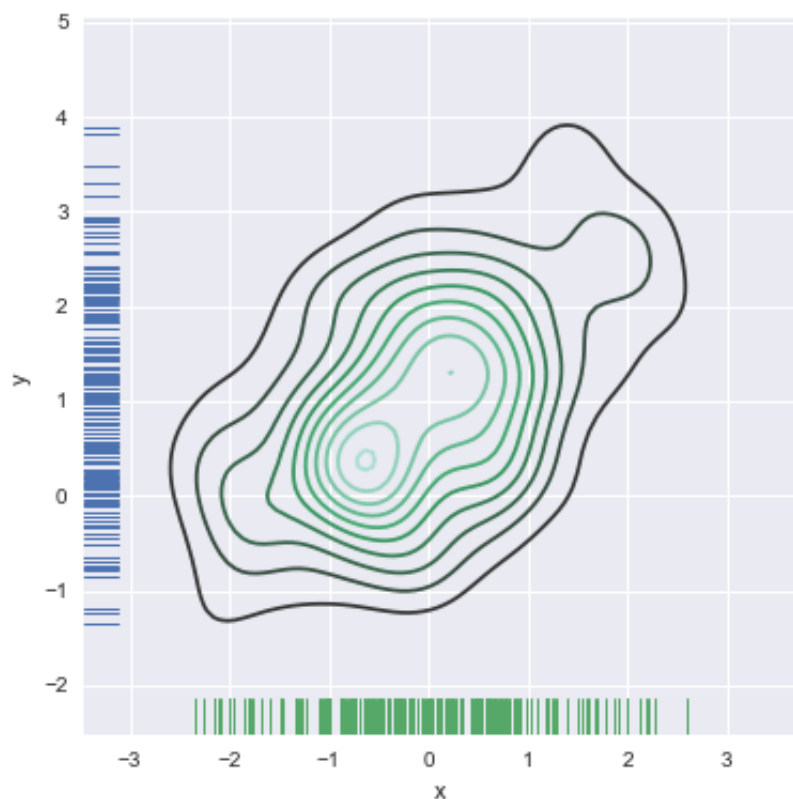
If you wish to show the bivariate density more continuously, you can simply increase the number of contour levels:

```
f, ax = plt.subplots(figsize=(6, 6))
cmap = sns.cubehelix_palette(as_cmap=True, dark=0, light=1, reverse=True)
sns.kdeplot(df.x, df.y, cmap=cmap, n_levels=60, shade=True);
```
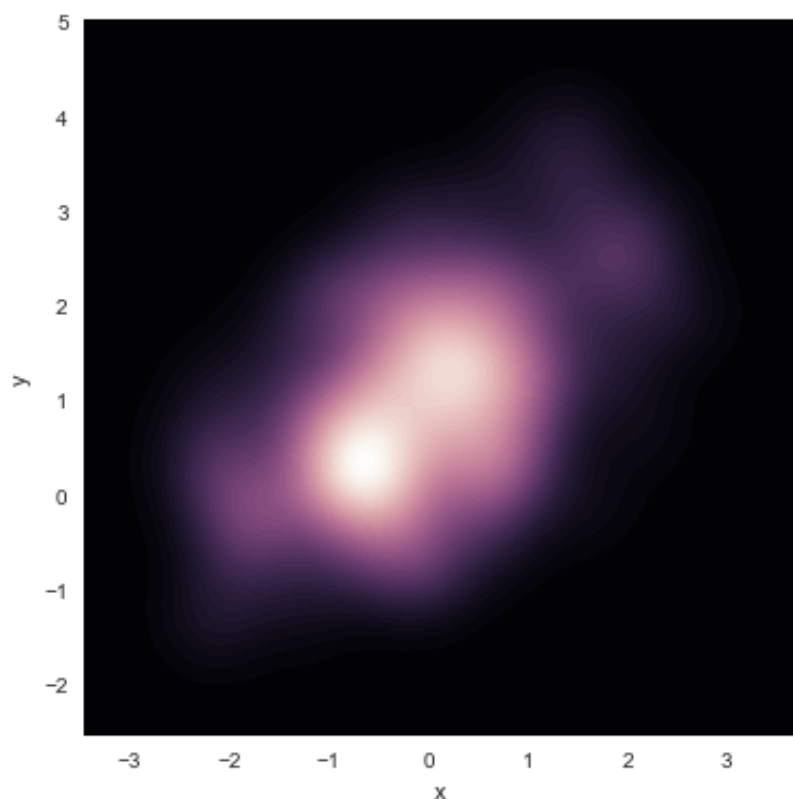

The `jointplot()` (../generated/seaborn.jointplot.html#seaborn.jointplot) function uses a `JointGrid` (../generated/seaborn.JointGrid.html#seaborn.JointGrid) to manage the figure. For more flexibility, you may want to draw your figure by using `JointGrid`

(../generated/seaborn.JointGrid.html#seaborn.JointGrid) directly. `jointplot()`
(../generated/seaborn.jointplot.html#seaborn.jointplot) returns the `JointGrid`
(../generated/seaborn.JointGrid.html#seaborn.JointGrid) object after plotting, which you can use
to add more layers or to tweak other aspects of the visualization:

```
g = sns.jointplot(x="x", y="y", data=df, kind="kde", color="m")
g.plot_joint(plt.scatter, c="w", s=30, linewidth=1, marker="+")
g.ax_joint.collections[0].set_alpha(0)
g.set_axis_labels("$X$", "$Y$");
```
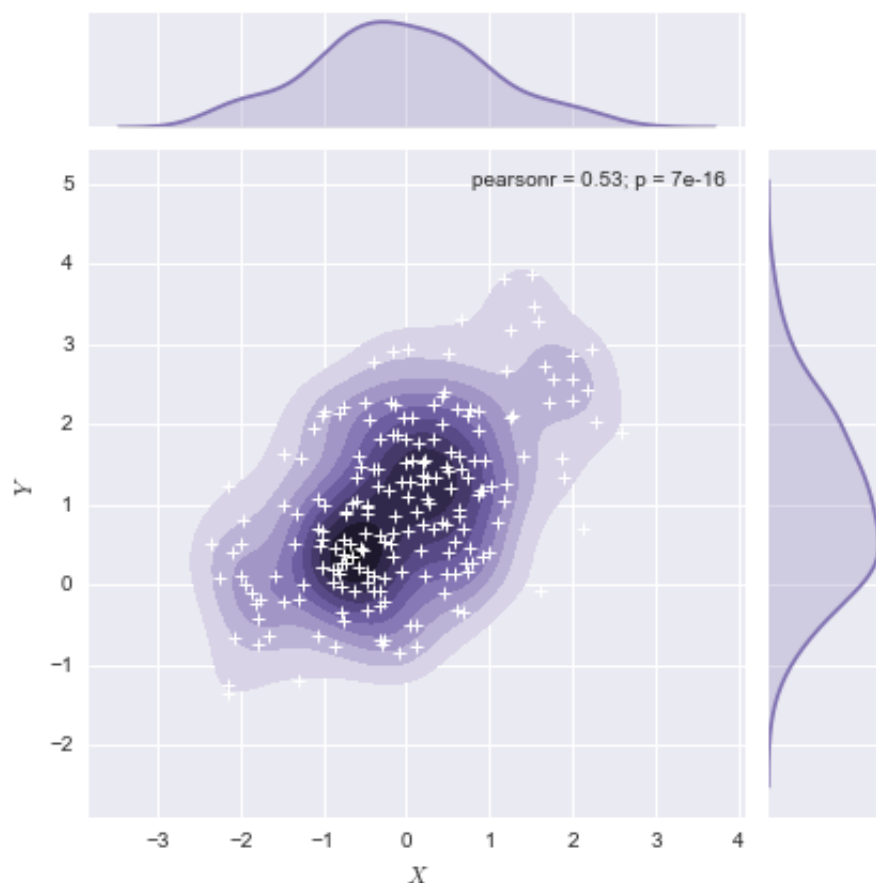

Visualizing pairwise relationships in a dataset

To plot multiple pairwise bivariate distributions in a dataset, you can use the `pairplot()`
(../generated/seaborn.pairplot.html#seaborn.pairplot) function. This creates a matrix of axes and
shows the relationship for each pair of columns in a DataFrame. by default, it also draws the
univariate distribution of each variable on the diagonal Axes:

```
iris = sns.load_dataset("iris")
sns.pairplot(iris);
```
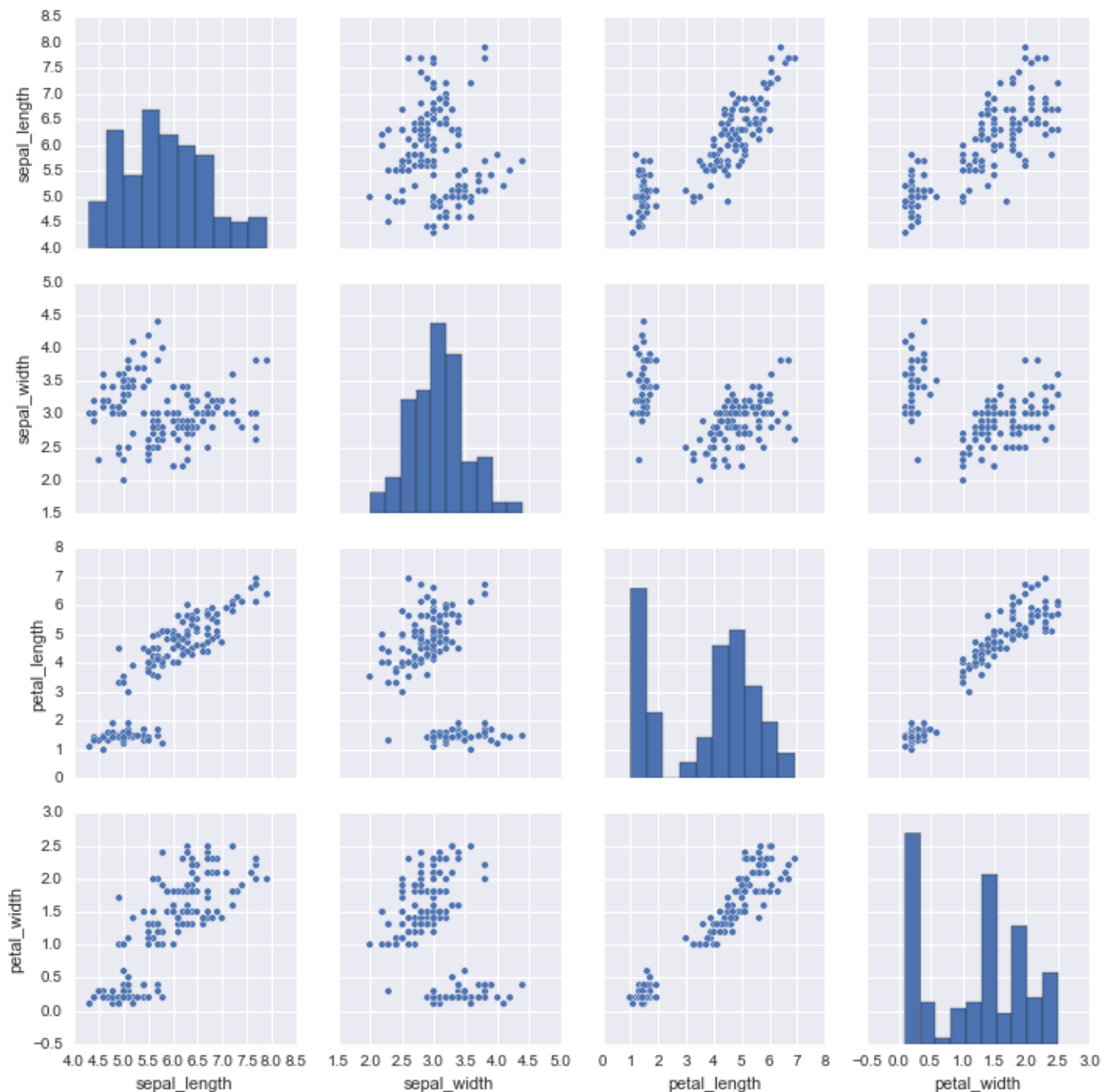
Much like the relationship between `jointplot()`
(../generated/seaborn.jointplot.html#seaborn.jointplot) and `JointGrid`
(../generated/seaborn.JointGrid.html#seaborn.JointGrid), the `pairplot()`
(../generated/seaborn.pairplot.html#seaborn.pairplot) function is built on top of a `PairGrid`
(../generated/seaborn.PairGrid.html#seaborn.PairGrid) object, which can be used directly for more
flexibility:

```
g = sns.PairGrid(iris)
g.map_diag(sns.kdeplot)
g.map_offdiag(sns.kdeplot, cmap="Blues_d", n_levels=6);
```

```
/Users/mwaskom/anaconda/lib/python2.7/site-packages/matplotlib/axes/_axes.py:519
: UserWarning: No labelled objects found. Use label='...' kwarg on individual pl
ots.
  warnings.warn("No labelled objects found. "
```
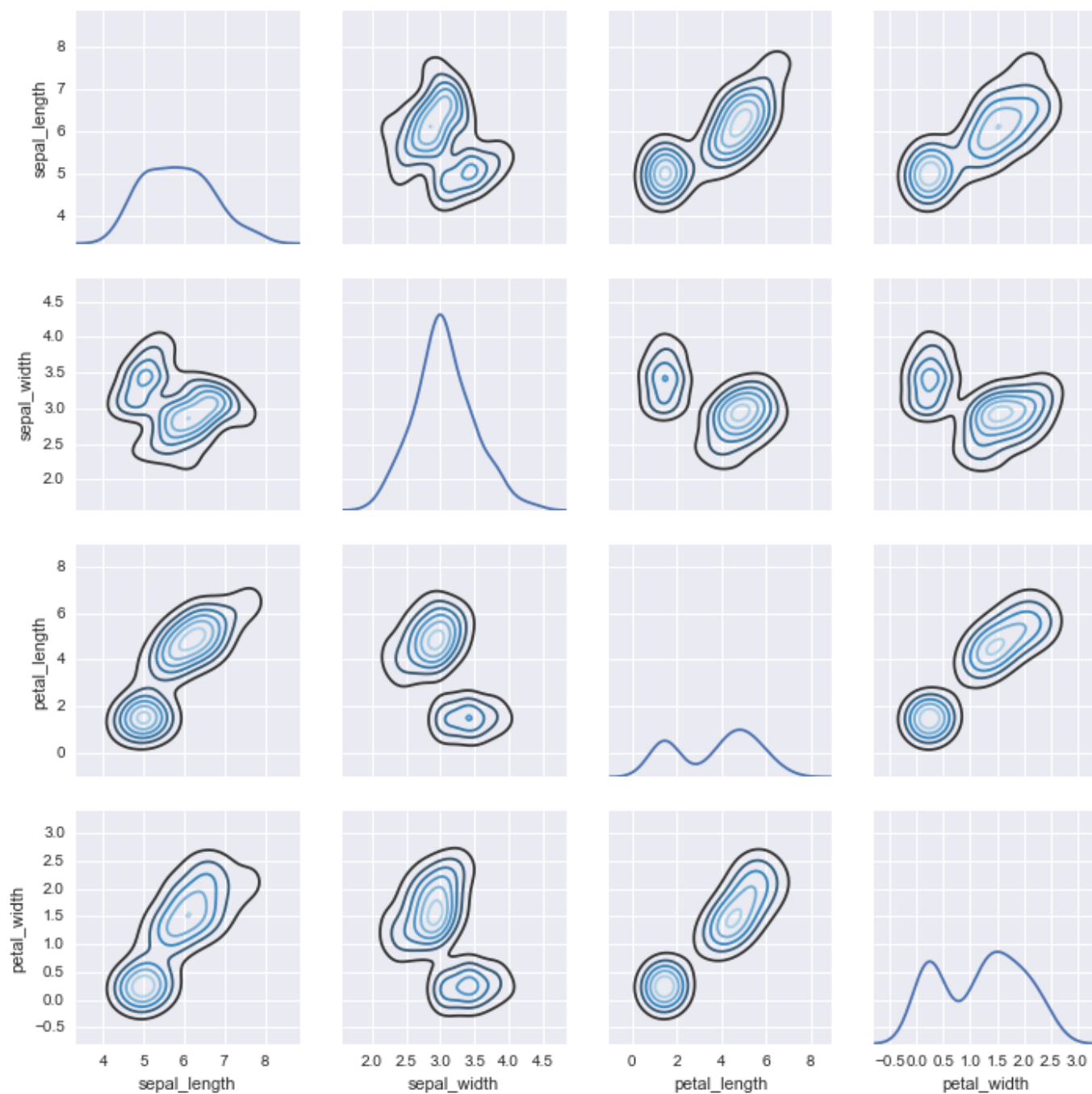
Source (../_sources/tutorial/distributions.txt)   Back to top

© Copyright 2012-2015, Michael Waskom.
Created using Sphinx (http://sphinx-doc.org/) 1.3.3.