

Visualization with Seaborn

Matplotlib is a useful tool, but it leaves much to be desired. There are several valid complaints about matplotlib that often come up:

- Matplotlib's defaults are not exactly the best choices. It was based off of MatLab circa 1999, and this shows.
- Matplotlib is relatively low-level. Doing sophisticated statistical visualization is possible, but often requires a *lot* of boilerplate code.
- Matplotlib is not designed for use with Pandas dataframes. In order to visualize data from a Pandas dataframe, you must extract each series and often concatenate these series' together into the right format.
- The answer to these problems is **Seaborn**. Seaborn provides an API on top of matplotlib which uses sane plot & color defaults, uses simple functions for common statistical plot types, and which integrates with the functionality provided by Pandas dataframes.

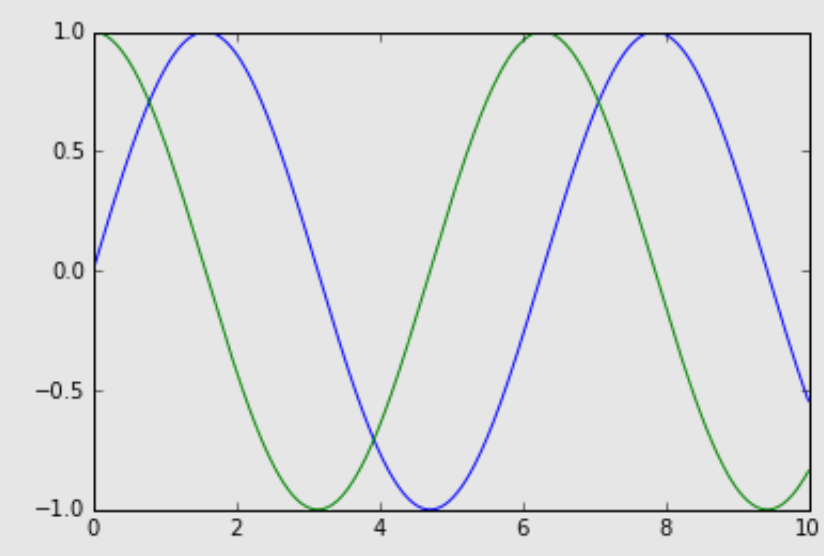
Let's take a look at Seaborn in action. We'll start by importing the key libraries we'll need.

```
1  from __future__ import print_function, division
2
3  %matplotlib inline
4  import matplotlib.pyplot as plt
5  import numpy as np
6  import pandas as pd
```

Run Again

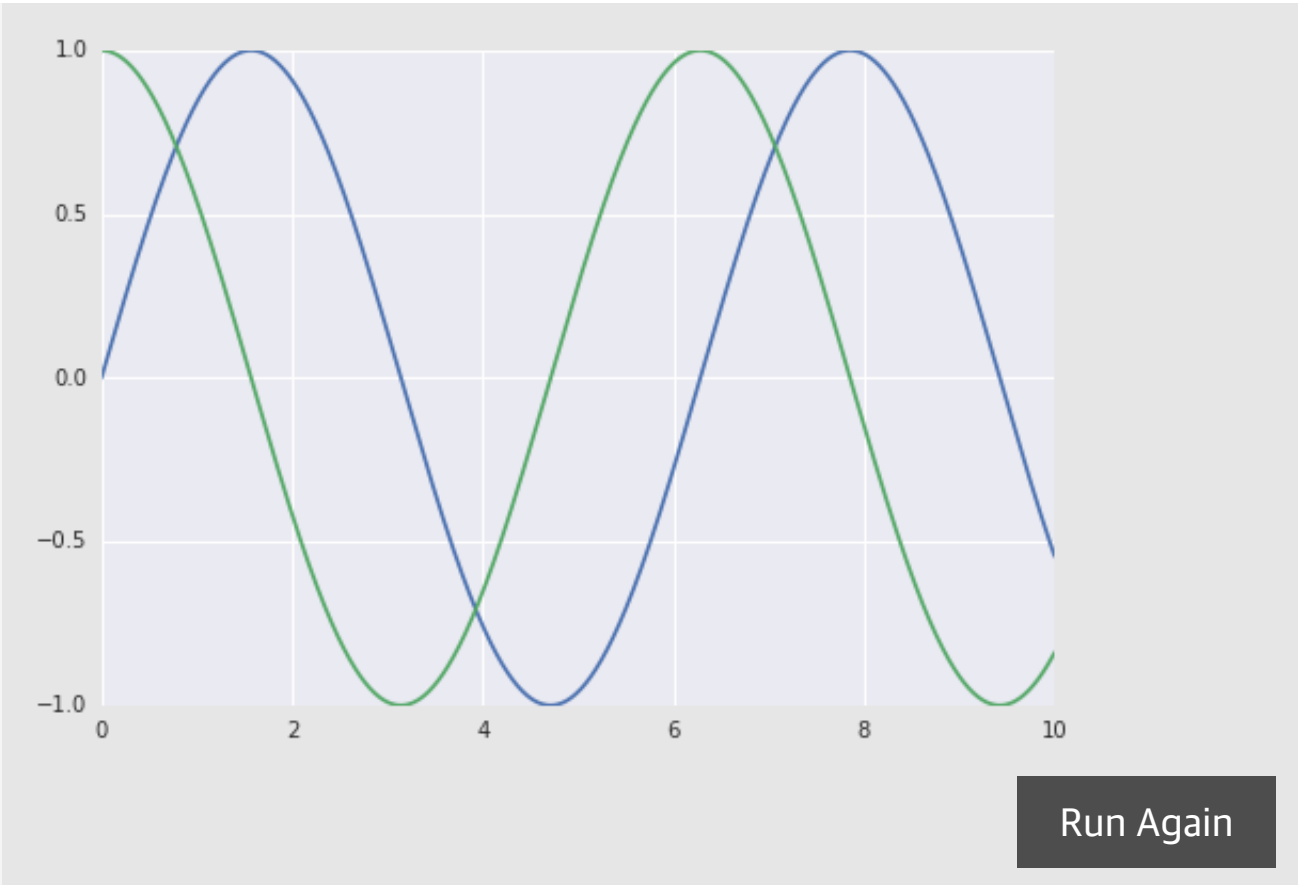
Then we import seaborn, which by convention is imported as `sns` . We can set the seaborn style as the default matplotlib style by calling `sns.set()` : after doing this, even simple matplotlib plots will look much better. Let's look at a before and after:

```
1  x = np.linspace(0, 10, 1000)
2  plt.plot(x, np.sin(x), x, np.cos(x));
```



Run Again

```
1  import seaborn as sns
2  sns.set()
3  plt.plot(x, np.sin(x), x, np.cos(x));
```



Ah, much better!

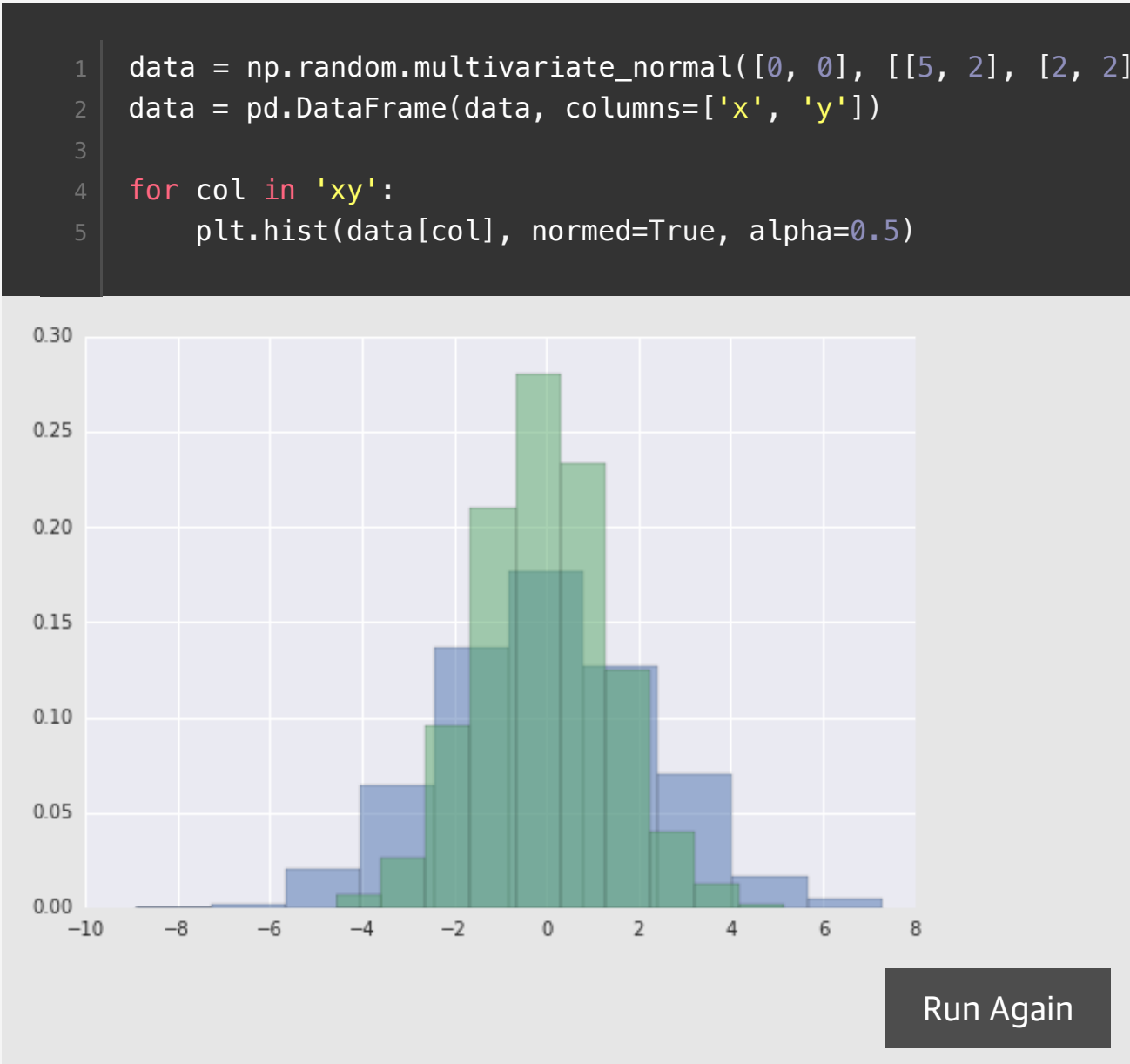
Exploring Seaborn Plots

The main idea of Seaborn is that it can create complicated plot types from Pandas data with relatively simple commands.

Let's take a look at a few of the datasets and plot types available in Seaborn. Note that all o the following *could* be done using raw matplotlib commands (this is, in fact, what Seaborn does under the hood) but the seaborn API is much more convenient.

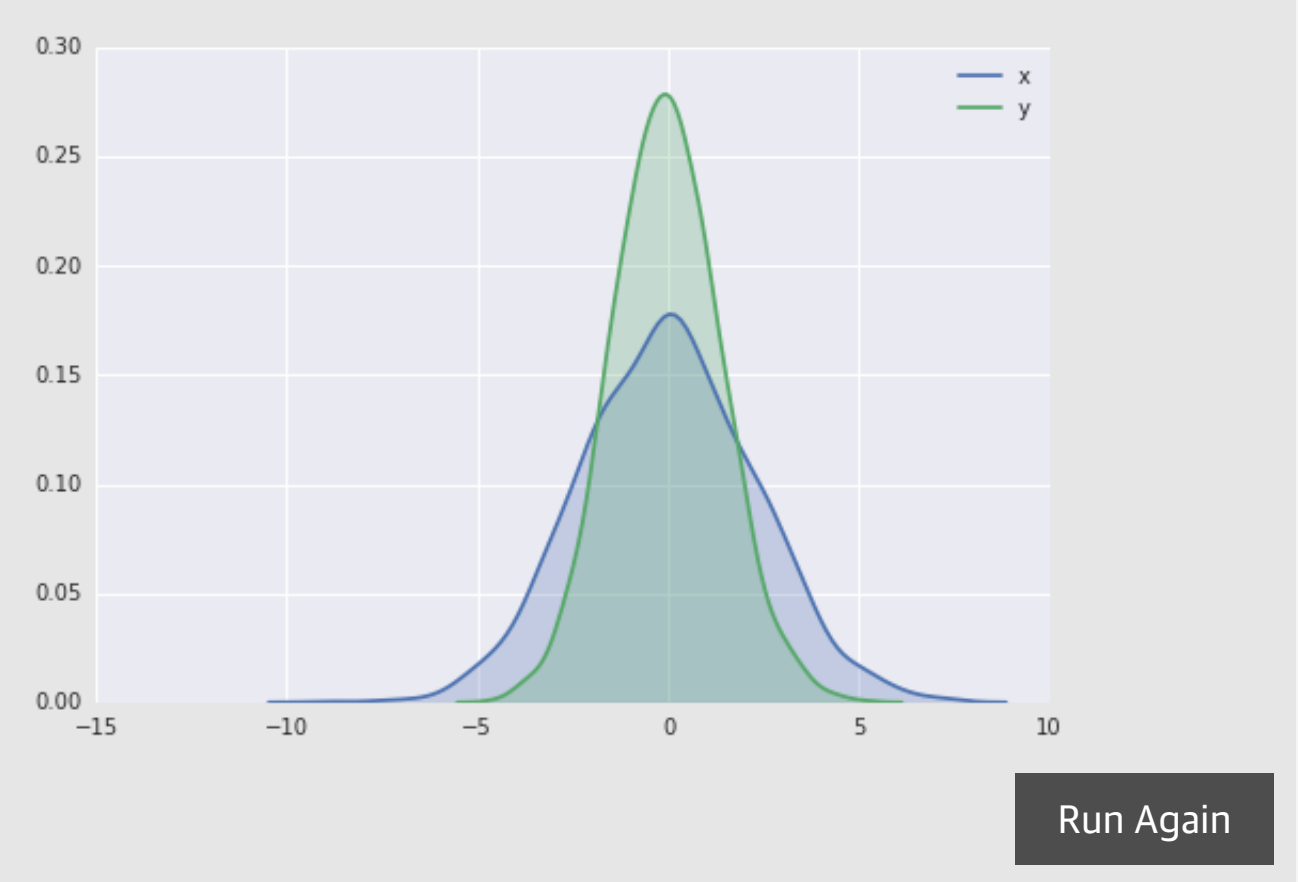
Histograms, KDE, and Densities

Often in statistical data visualization, all you want is to plot histograms and joint distributions of variables. Seaborn provides simple tools to make this happen:

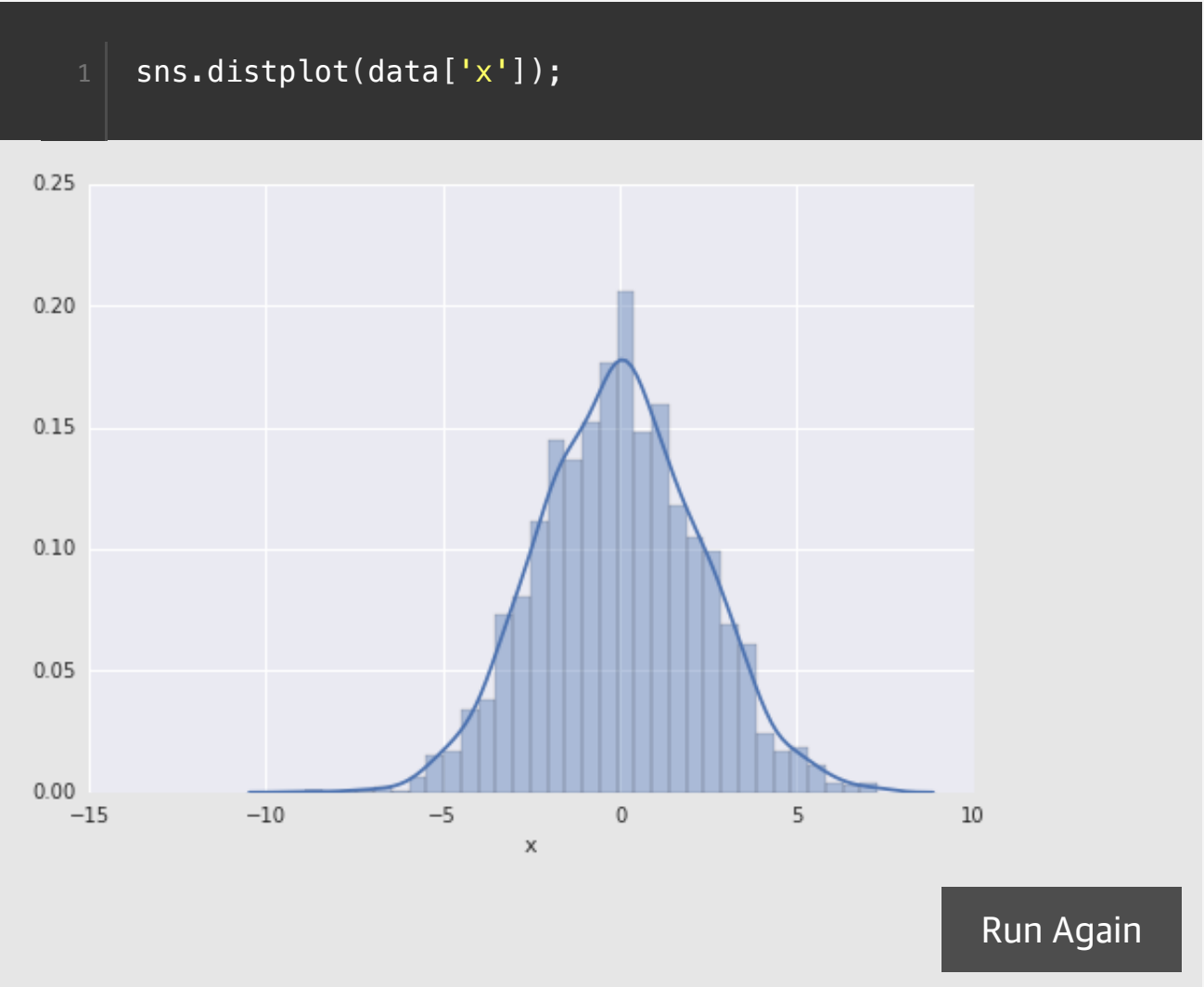


Rather than a histogram, we can get a smooth estimate of the distribution using a kernel density estimation:

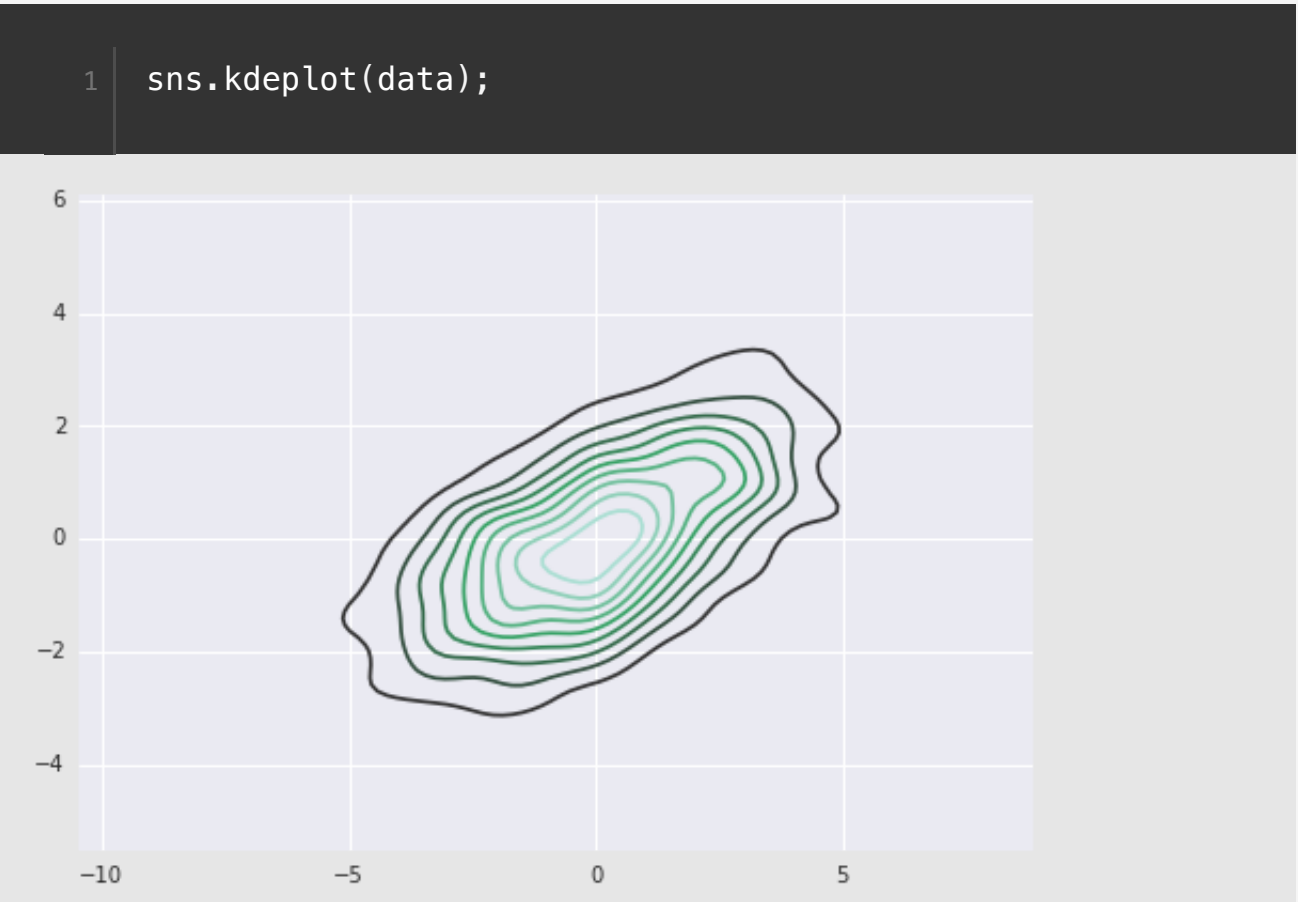
```
1 for col in 'xy':
2     sns.kdeplot(data[col], shade=True)
```



Histograms and KDE can be combined using `distplot` :



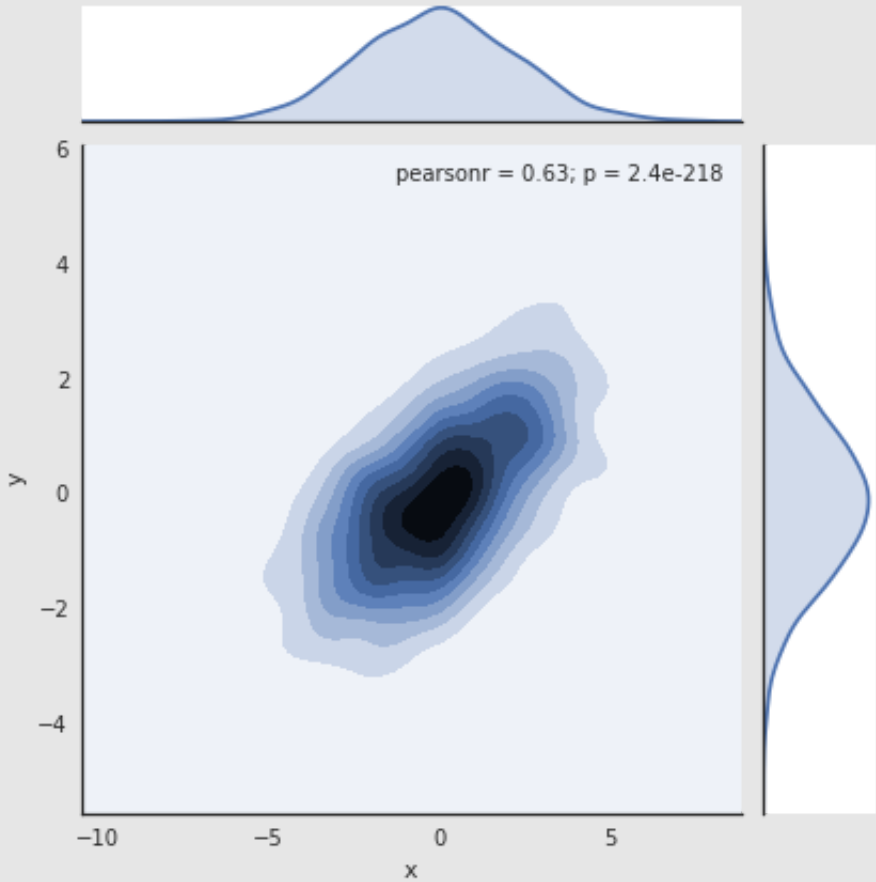
If we pass the full two-dimensional dataset to `kdeplot` , we will get a two-dimensional visualization of the data:



Run Again

We can see the joint distribution and the marginal distributions together using `sns.jointplot` . For this plot, we'll set the style to a white background:

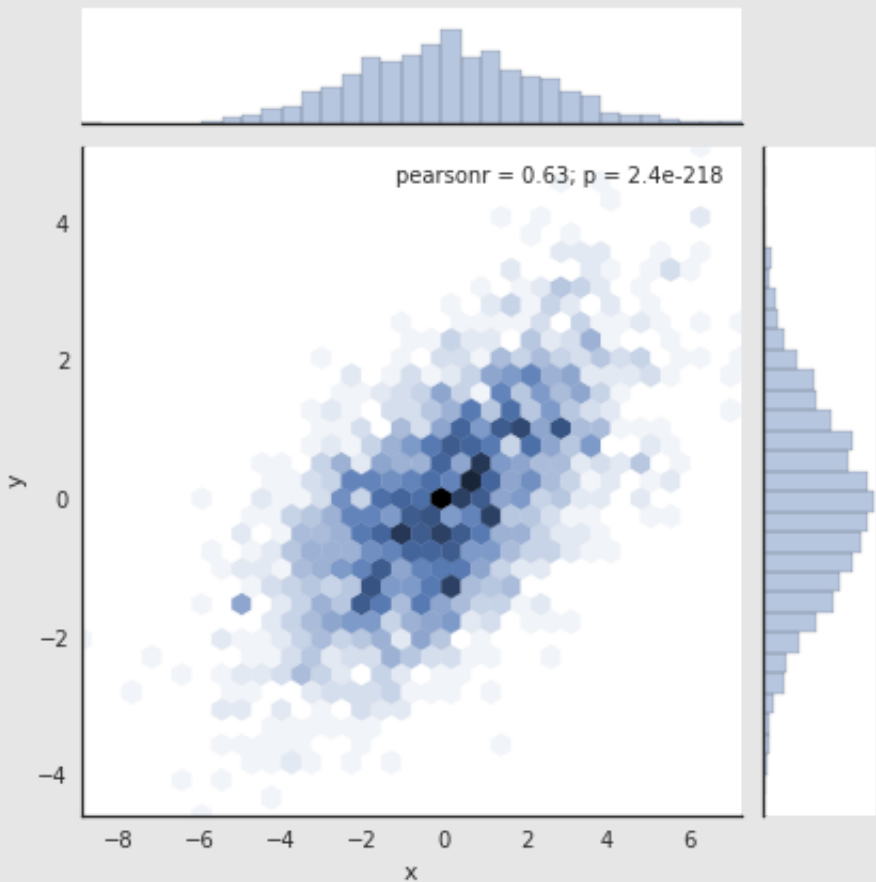
```
1 with sns.axes_style('white'):  
2     sns.jointplot("x", "y", data, kind='kde');
```



Run Again

There are other parameters which can be passed to `jointplot` : for example, we can use a hexagonally-based histogram instead:

```
1 with sns.axes_style('white'):  
2     sns.jointplot("x", "y", data, kind='hex');
```



Run Again

Pairplots

When you generalize joint plots to data sets of larger dimensions, you end up with *pair plots*. This is very useful for exploring correlations between multi-dimensional data, when you'd like to plot all pairs of values against each other.

We'll demo this with the well-known *iris* dataset, which lists measurements of petals and sepals of three iris species:

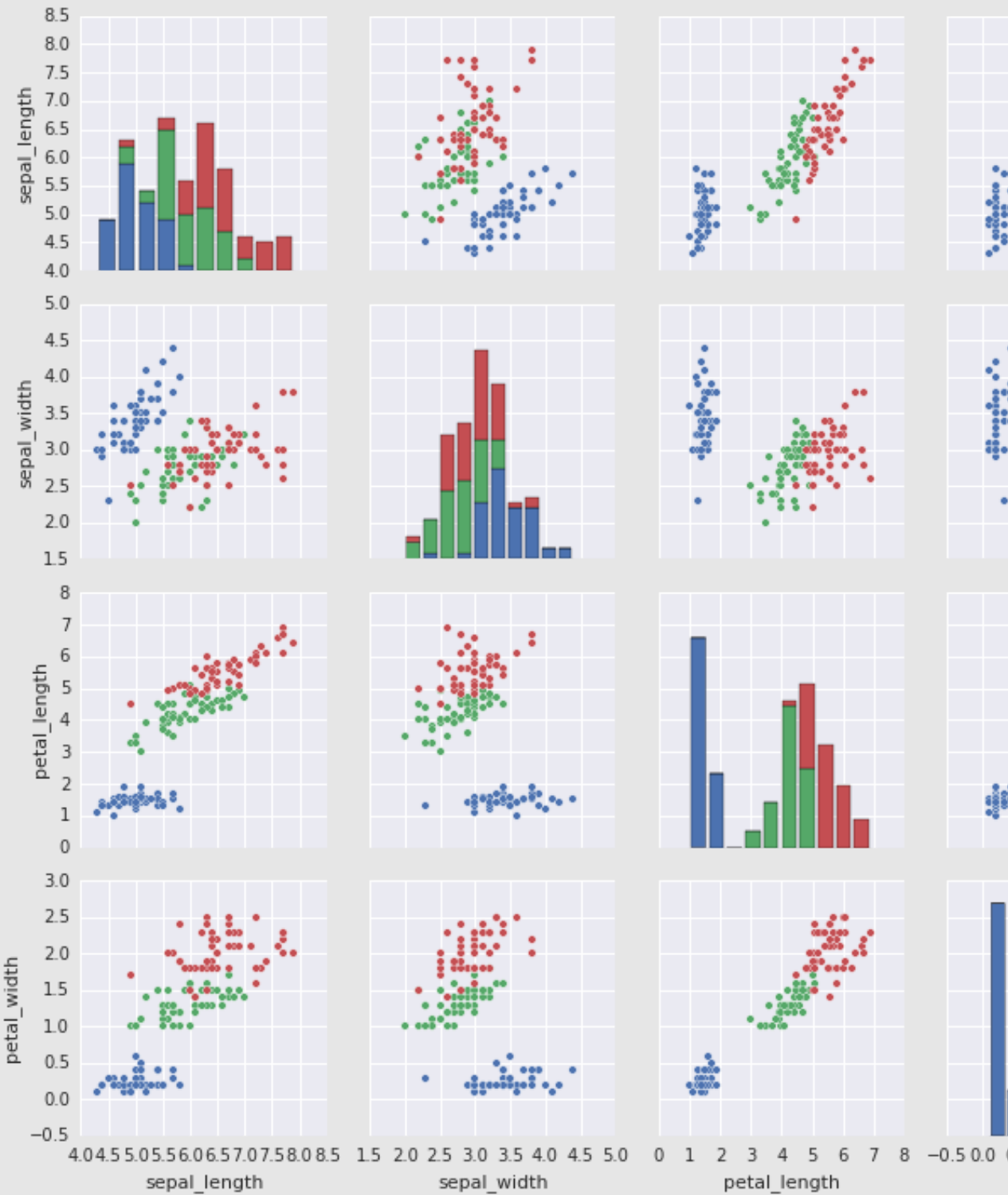
```
1 iris = sns.load_dataset("iris")
2 iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	sp
0	5.1	3.5	1.4	0.2	se
1	4.9	3.0	1.4	0.2	se
2	4.7	3.2	1.3	0.2	se
3	4.6	3.1	1.5	0.2	se
4	5.0	3.6	1.4	0.2	se

Run Again

Visualizing the multi-dimensional relationships among the samples is as easy as calling `sns.pairplot` :

```
1 sns.pairplot(iris, hue='species', size=2.5);
```



Run Again

Faceted Histograms

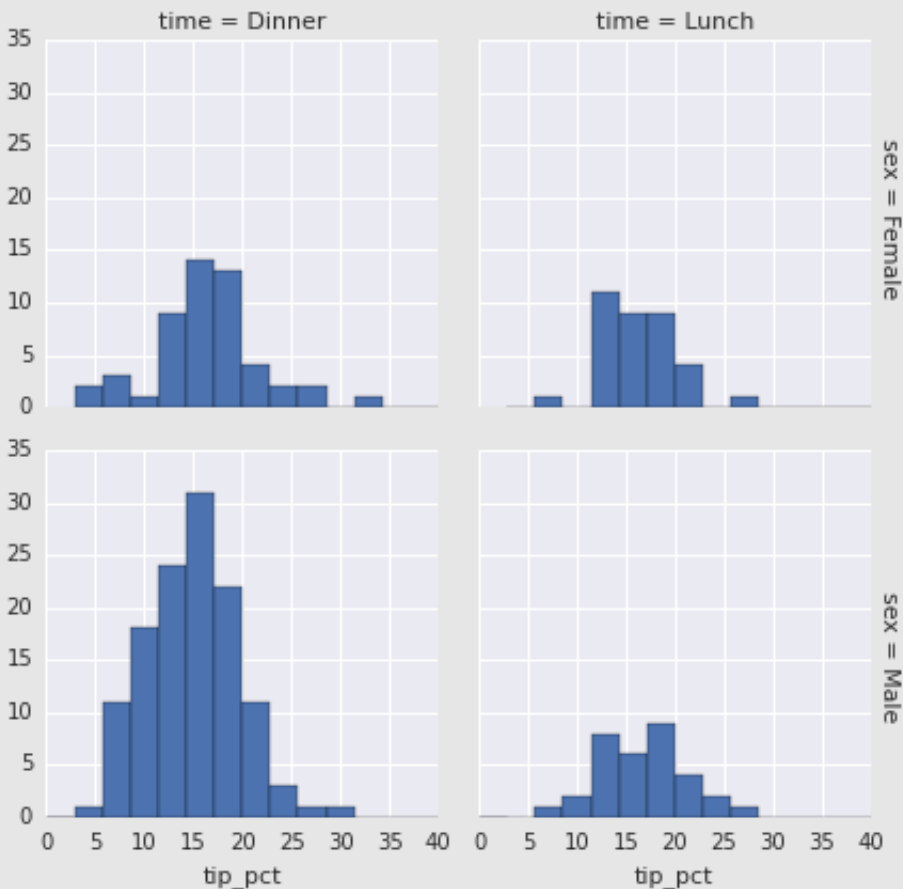
Sometimes the best way to view data is via histograms of subsets. Seaborn's `FacetGrid` makes this extremely simple. We'll take a look at some data which shows the amount that restaurant staff receive in tips based on various indicator data:

```
1 tips = sns.load_dataset('tips')
2 tips.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Run Again

```
1 tips['tip_pct'] = 100 * tips['tip'] / tips['total_bill']
2
3 grid = sns.FacetGrid(tips, row="sex", col="time", margin_titles=True)
4 grid.map(plt.hist, "tip_pct", bins=np.linspace(0, 40, 15));
```

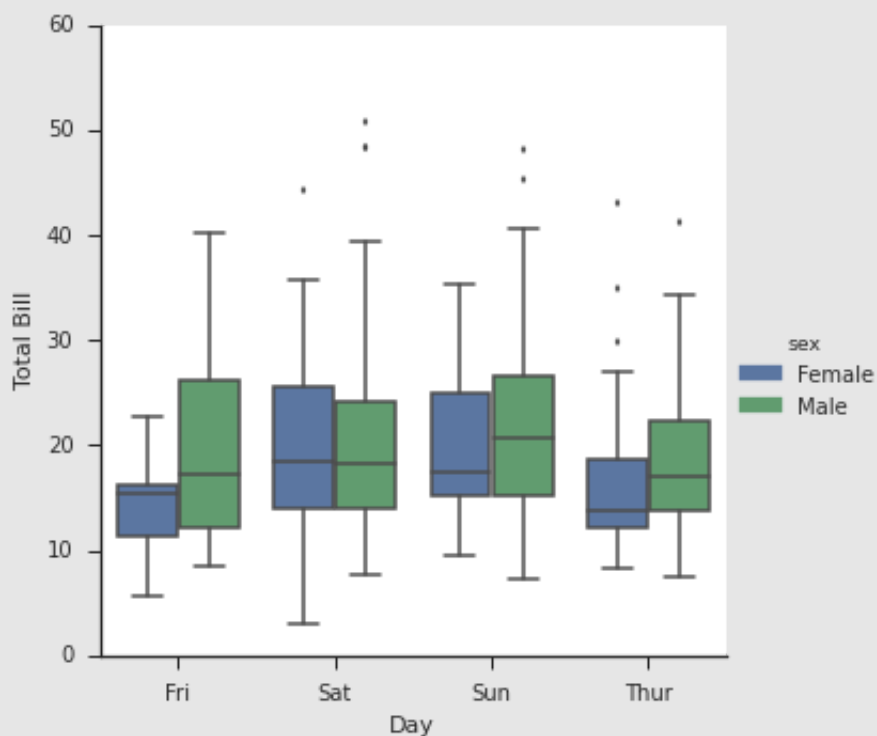


Run Again

Factor Plots

Factor plots can be used to visualize this data as well. This allows you to view the distribution of a parameter within bins defined by any other parameter:

```
1 with sns.axes_style(style='ticks'):
2     g = sns.factorplot("day", "total_bill", "sex", data=tips)
3     g.set_axis_labels("Day", "Total Bill");
```

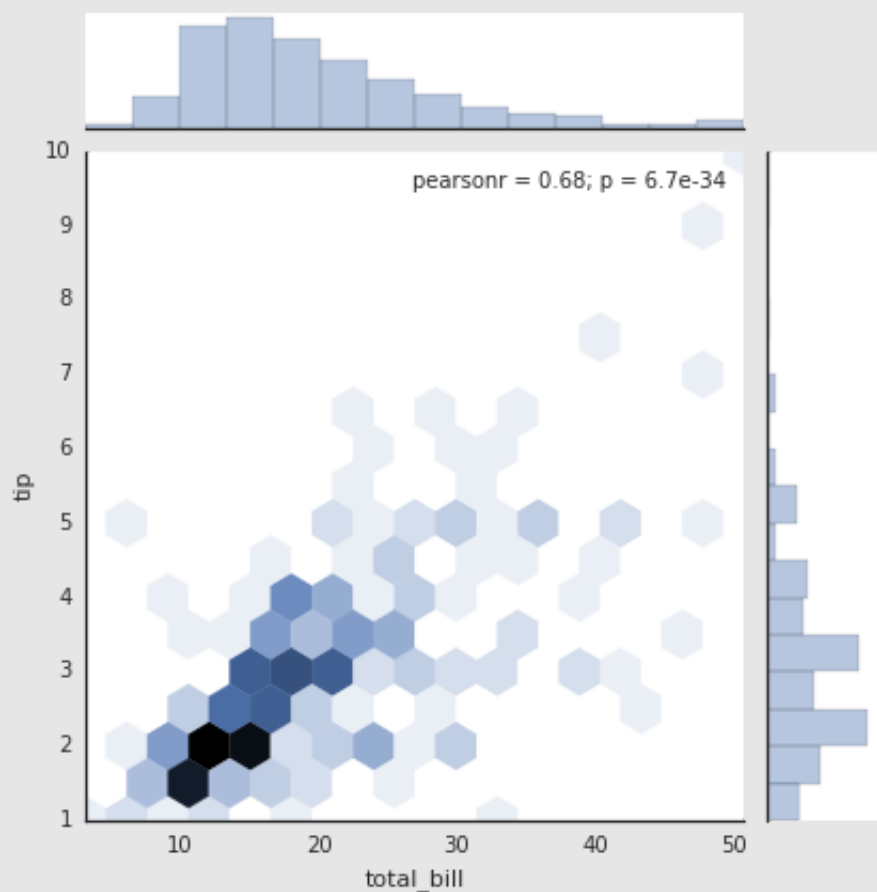


Run Again

Joint Distributions

Similar to the pairplot we saw above, we can use `sns.jointplot` to show the joint distribution between different datasets, along with the associated marginal distributions:

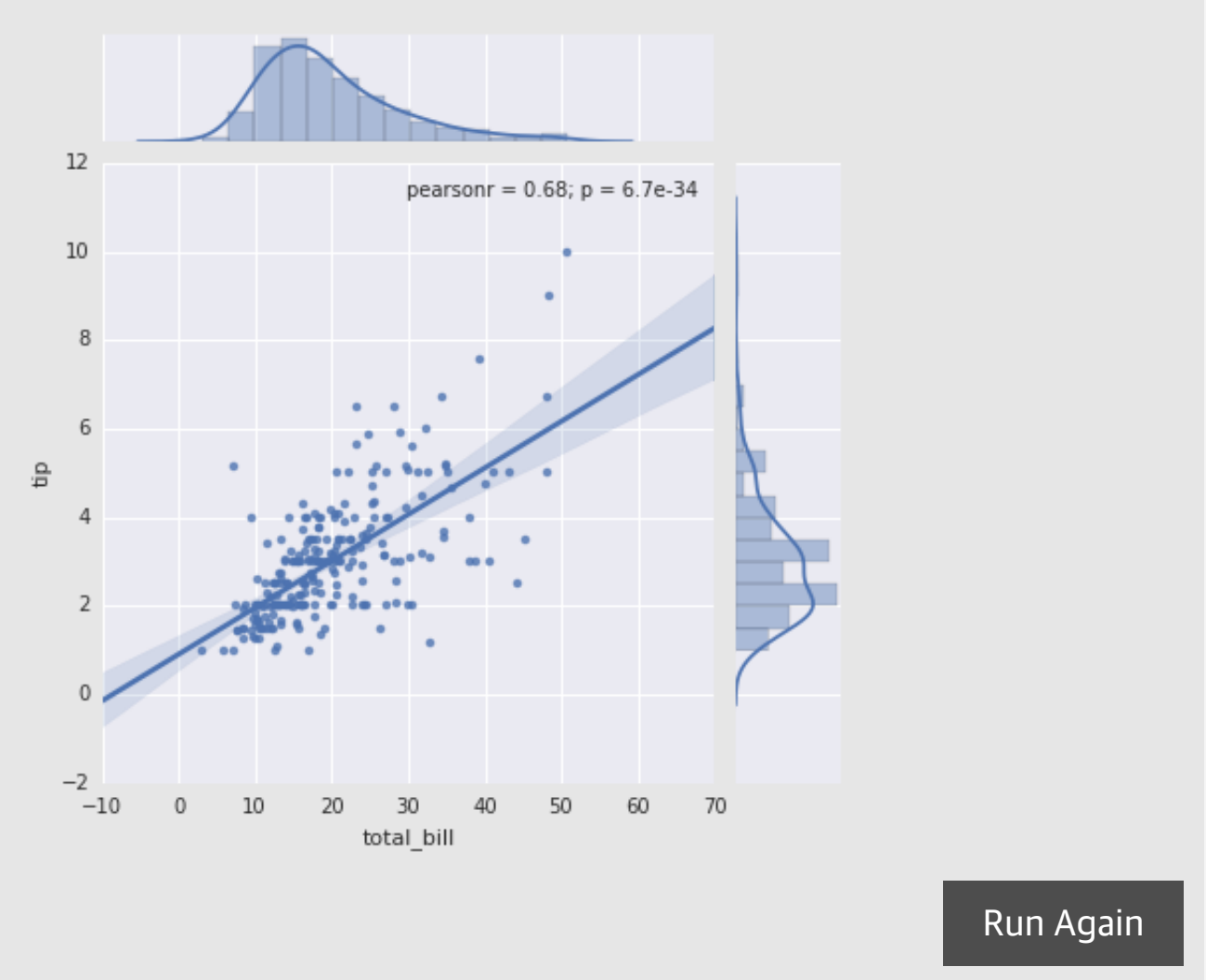
```
1 with sns.axes_style('white'):
2     sns.jointplot("total_bill", "tip", data=tips, kind='hex');
```



Run Again

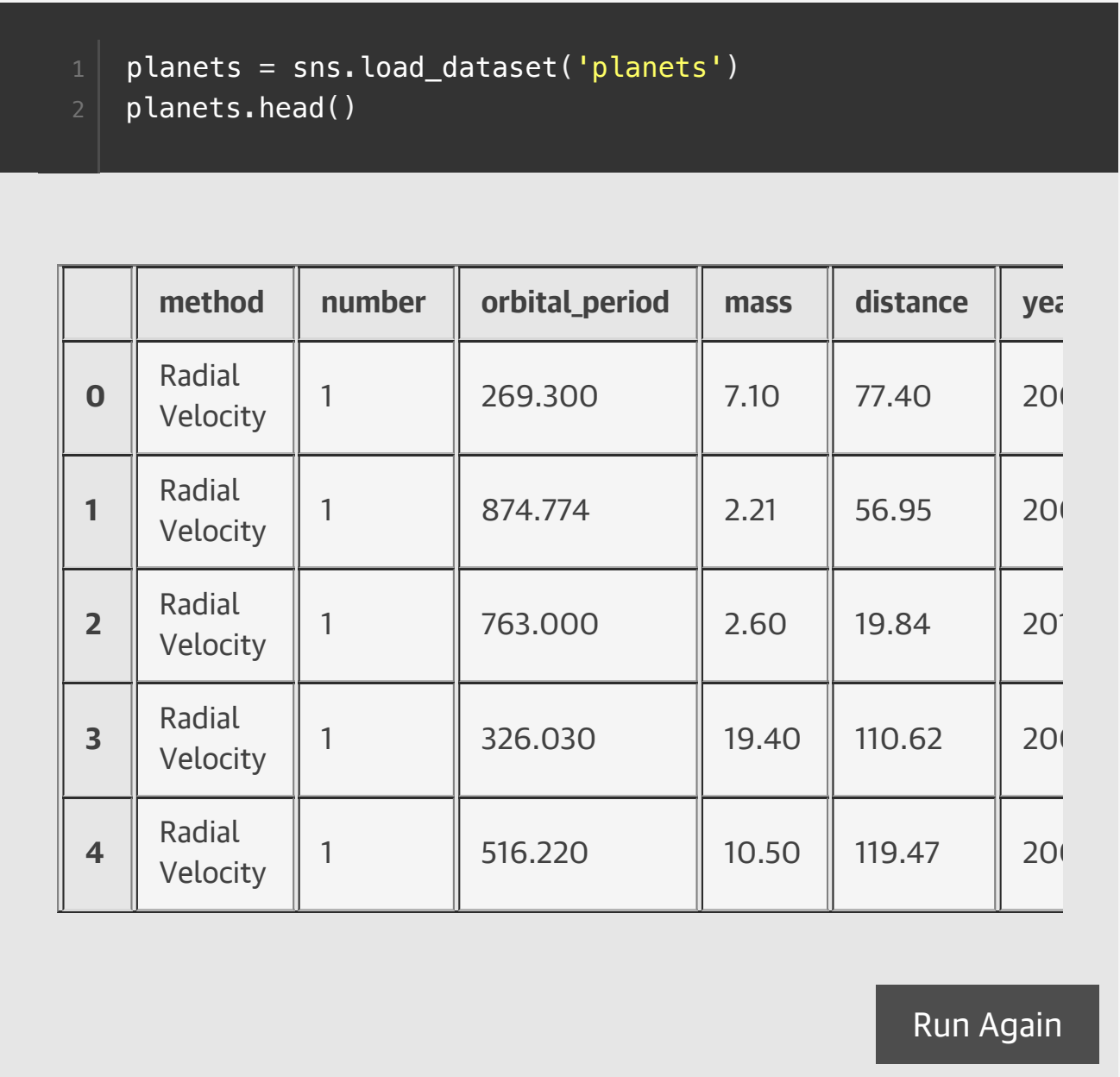
The joint plot can even do some automatic kernel density estimation and regression:

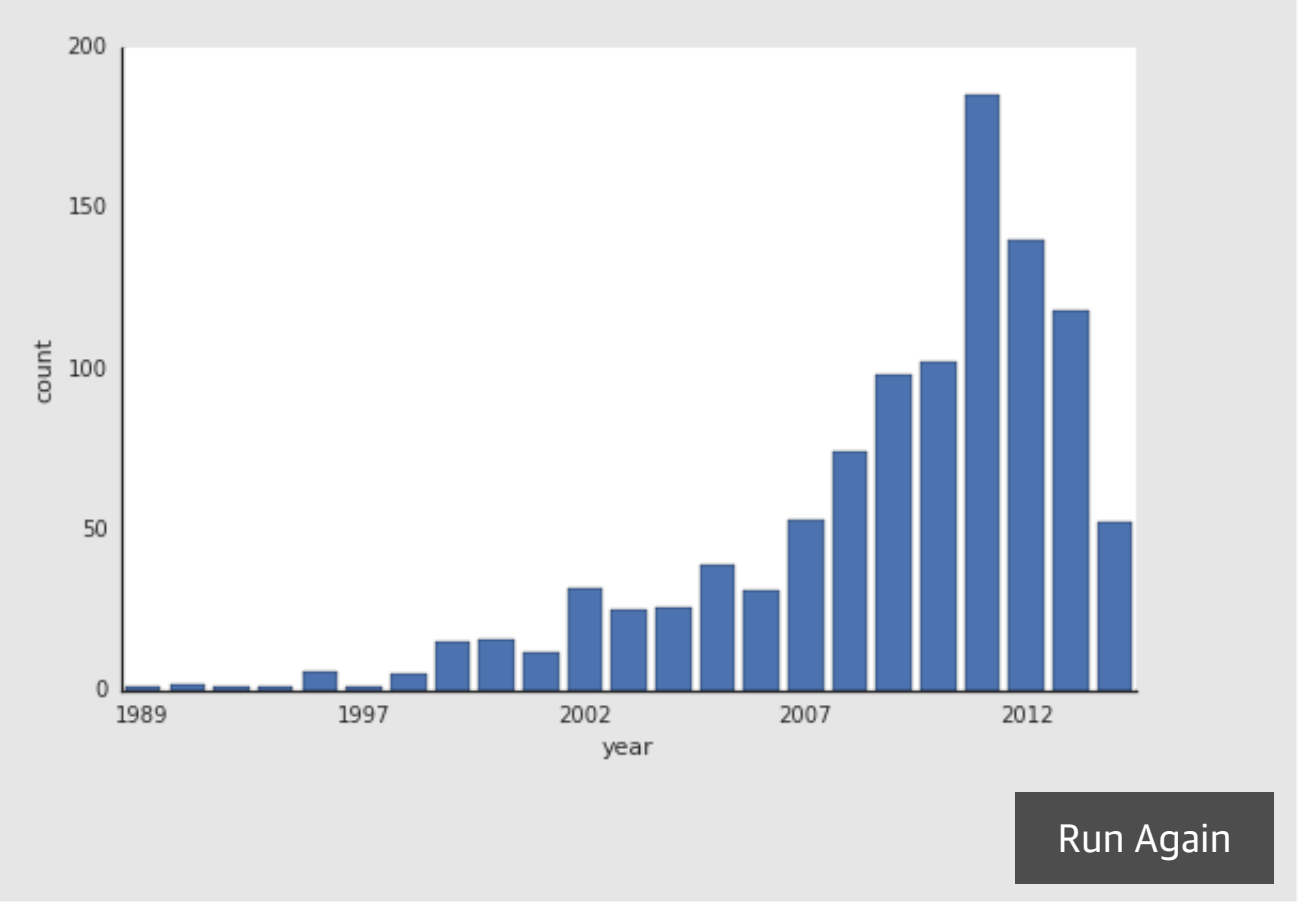
```
1 sns.jointplot("total_bill", "tip", data=tips, kind='reg');
```



Bar Plots

Time series can be plotted using `sns.factorplot` :





We can learn more by looking at the **method** of discovery of each of these planets:



For more information on plotting with Seaborn, see the [seaborn documentation](#), the [seaborn gallery](#), and the official [seaborn tutorial](#).