Now let's take a look at how it works with Seaborn. As we will see, Seaborn has many of its own high-level plotting routines, but it can also overwrite Matplotlib's default parameters and in turn get even simple Matplotlib scripts to produce vastly superior output. We can set the style by calling Seaborn's `set()` method. By convention, Seaborn is imported as `sns`:

```
In[4]: import seaborn as sns
       sns.set()
```

Now let's rerun the same two lines as before (Figure 4-112):

```
In[5]: # same plotting code as above!
       plt.plot(x, y)
       plt.legend('ABCDEF', ncol=2, loc='upper left');
```

Figure 4-112. Data in Seaborn's default style

Ah, much better!

Exploring Seaborn Plots

The main idea of Seaborn is that it provides high-level commands to create a variety of plot types useful for statistical data exploration, and even some statistical model fitting.

Let's take a look at a few of the datasets and plot types available in Seaborn. Note that all of 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. We have seen that this is relatively straightforward in Matplotlib (Figure 4-113):

```
In[6]: data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
       data = pd.DataFrame(data, columns=['x', 'y'])

       for col in 'xy':
           plt.hist(data[col], normed=True, alpha=0.5)
```

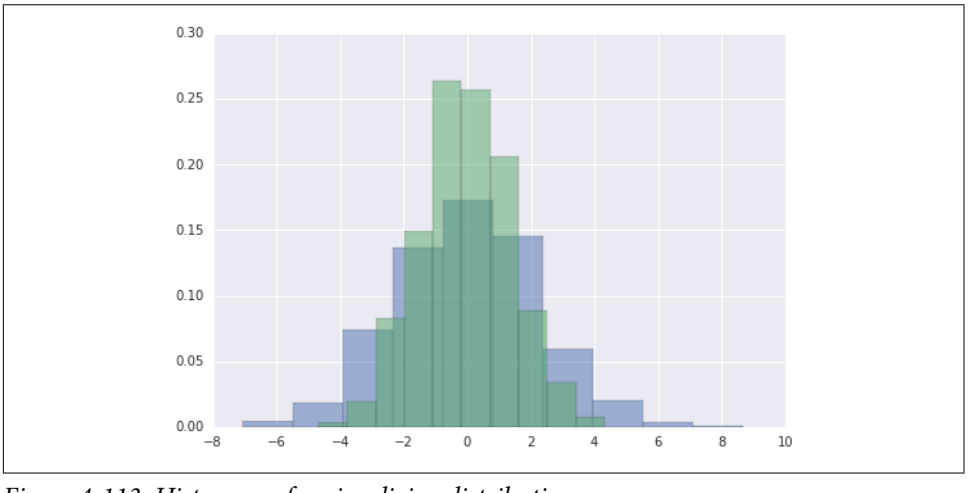

Figure 4-113. Histograms for visualizing distributions

Rather than a histogram, we can get a smooth estimate of the distribution using a kernel density estimation, which Seaborn does with `sns.kdeplot` (Figure 4-114):

```
In[7]: for col in 'xy':
           sns.kdeplot(data[col], shade=True)
```

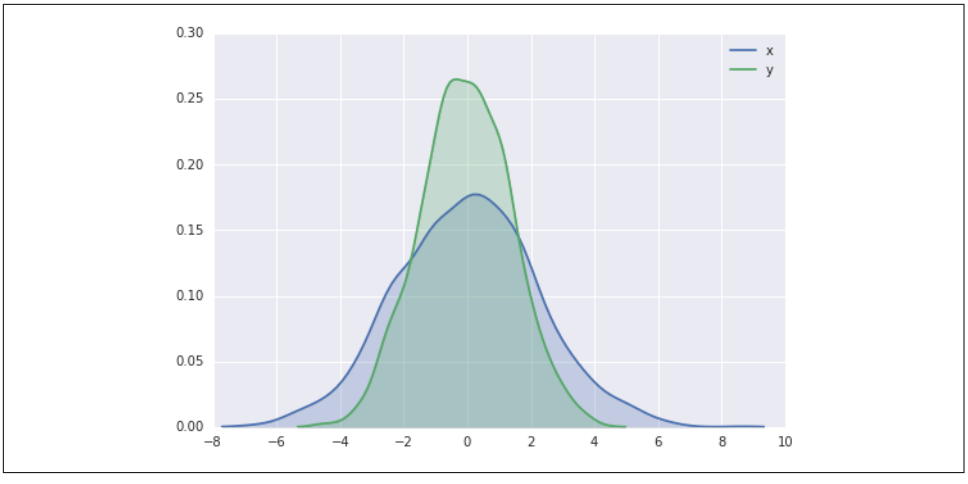

Figure 4-114. Kernel density estimates for visualizing distributions

Histograms and KDE can be combined using `distplot` (Figure 4-115):

```
In[8]: sns.distplot(data['x'])
       sns.distplot(data['y']);
```

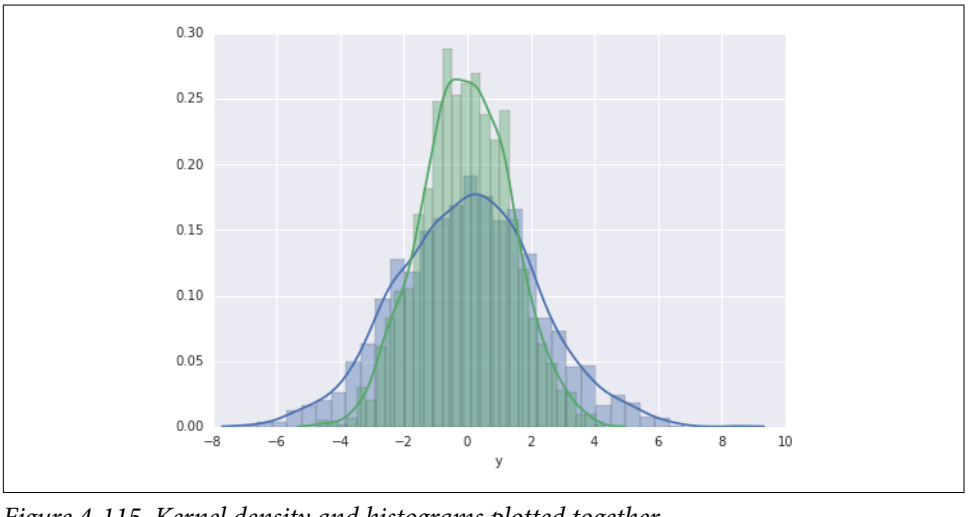

Figure 4-115. Kernel density and histograms plotted together

If we pass the full two-dimensional dataset to `kdeplot`, we will get a two-dimensional visualization of the data (Figure 4-116):

```
In[9]: sns.kdeplot(data);
```

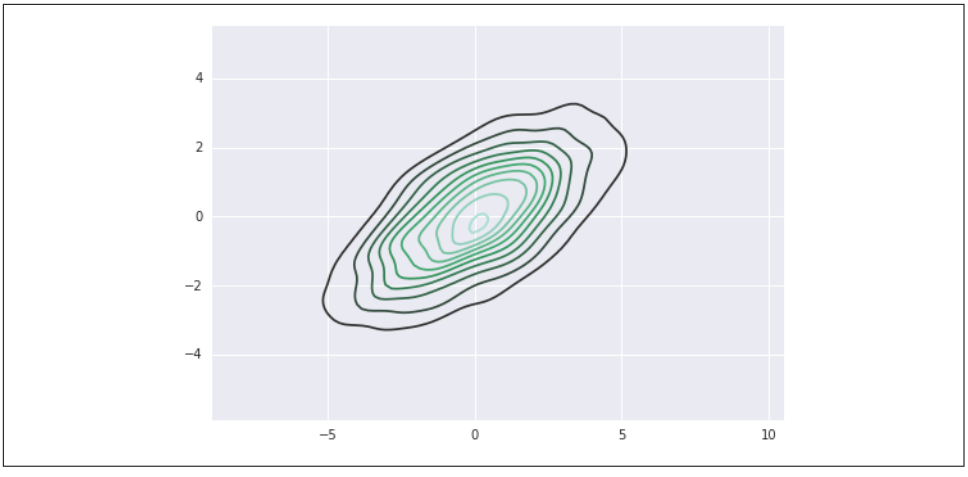

Figure 4-116. A two-dimensional kernel density plot

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 (Figure 4-117):

```
In[10]: with sns.axes_style('white'):
            sns.jointplot("x", "y", data, kind='kde');
```

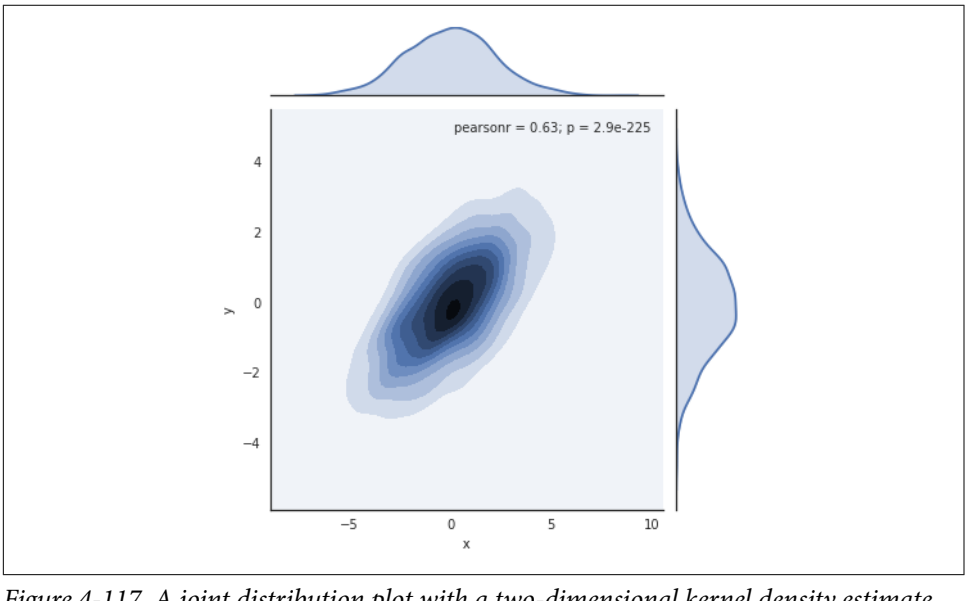

Figure 4-117. A joint distribution plot with a two-dimensional kernel density estimate

There are other parameters that can be passed to `jointplot`—for example, we can use a hexagonally based histogram instead (Figure 4-118):

```
In[11]: with sns.axes_style('white'):
            sns.jointplot("x", "y", data, kind='hex')
```

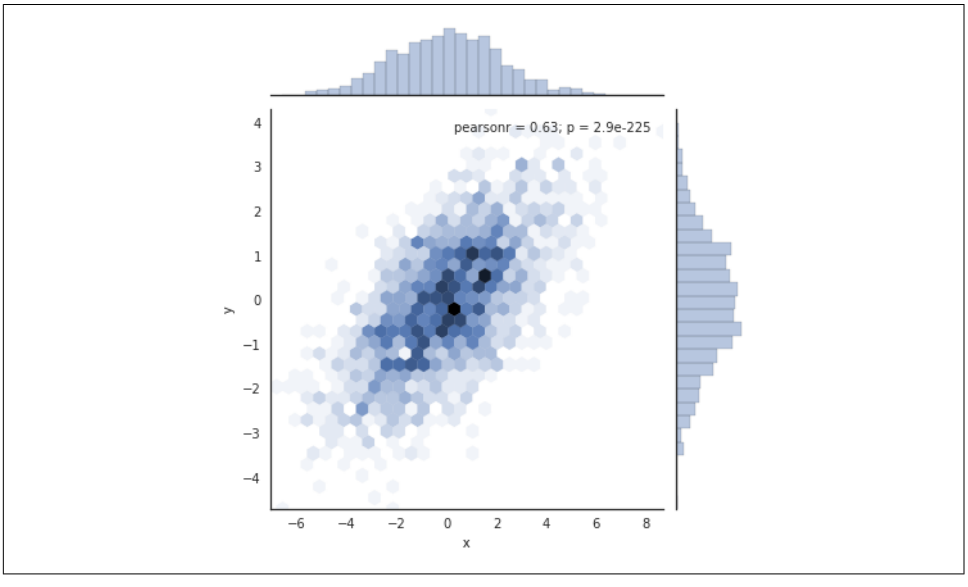

Figure 4-118. A joint distribution plot with a hexagonal bin representation

Pair plots

When you generalize joint plots to datasets of larger dimensions, you end up with *pair plots*. This is very useful for exploring correlations between multidimensional 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:

```
In[12]: iris = sns.load_dataset("iris")
        iris.head()

Out[12]:    sepal_length  sepal_width  petal_length  petal_width species
         0           5.1          3.5           1.4          0.2  setosa
         1           4.9          3.0           1.4          0.2  setosa
         2           4.7          3.2           1.3          0.2  setosa
         3           4.6          3.1           1.5          0.2  setosa
         4           5.0          3.6           1.4          0.2  setosa
```

Visualizing the multidimensional relationships among the samples is as easy as calling `sns.pairplot` (Figure 4-119):

```
In[13]: sns.pairplot(iris, hue='species', size=2.5);
```

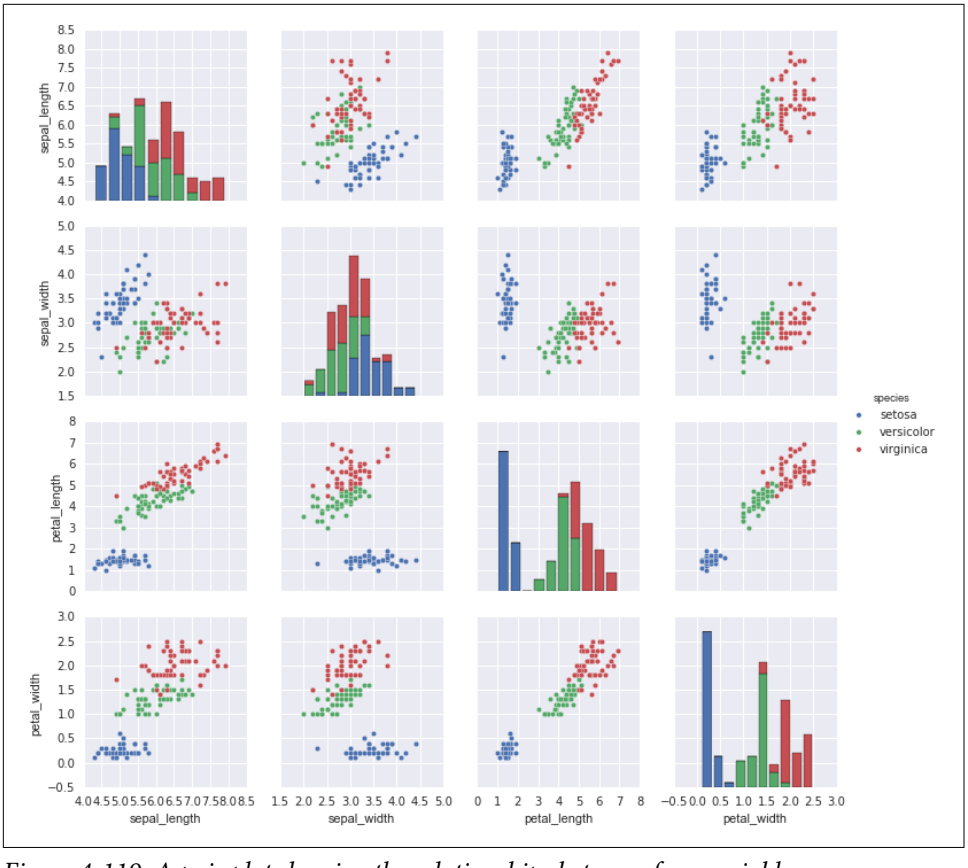

Figure 4-119. A pair plot showing the relationships between four variables

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 that shows the amount that restaurant staff receive in tips based on various indicator data (Figure 4-120):

```
In[14]: tips = sns.load_dataset('tips')
        tips.head()

Out[14]:    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
```

```
In[15]: tips['tip_pct'] = 100 * tips['tip'] / tips['total_bill']

        grid = sns.FacetGrid(tips, row="sex", col="time", margin_titles=True)
        grid.map(plt.hist, "tip_pct", bins=np.linspace(0, 40, 15));
```

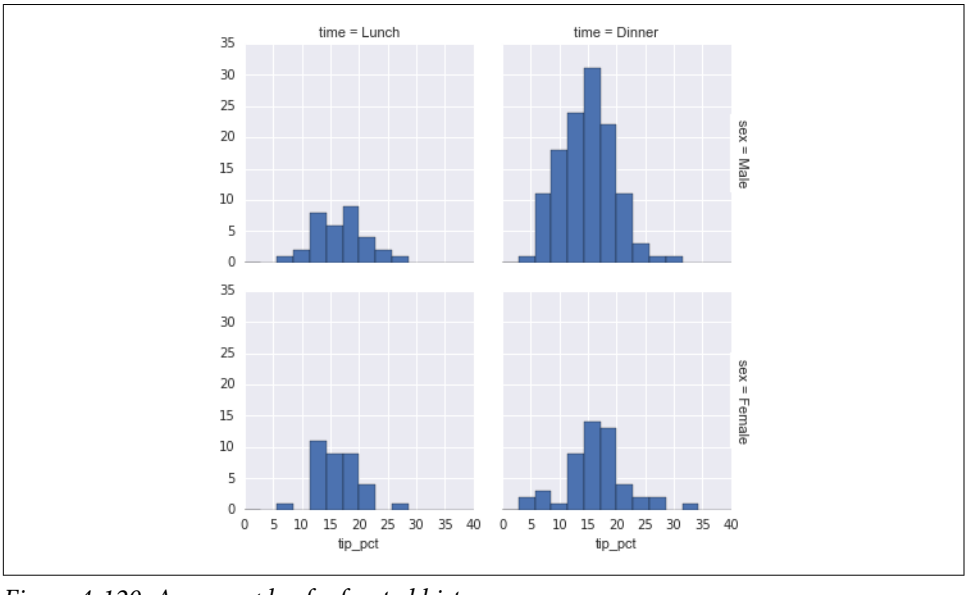

Figure 4-120. An example of a faceted histogram

Factor plots

Factor plots can be useful for this kind of visualization as well. This allows you to view the distribution of a parameter within bins defined by any other parameter (Figure 4-121):

```
In[16]: with sns.axes_style(style='ticks'):
            g = sns.factorplot("day", "total_bill", "sex", data=tips, kind="box")
            g.set_axis_labels("Day", "Total Bill");
```

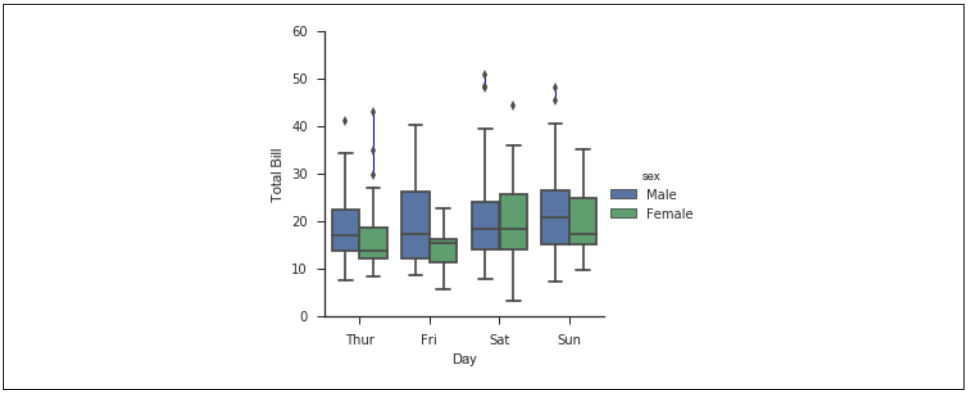

Figure 4-121. An example of a factor plot, comparing distributions given various discrete factors

Joint distributions

Similar to the pair plot we saw earlier, we can use `sns.jointplot` to show the joint distribution between different datasets, along with the associated marginal distributions (Figure 4-122):

```
In[17]: with sns.axes_style('white'):
            sns.jointplot("total_bill", "tip", data=tips, kind='hex')
```

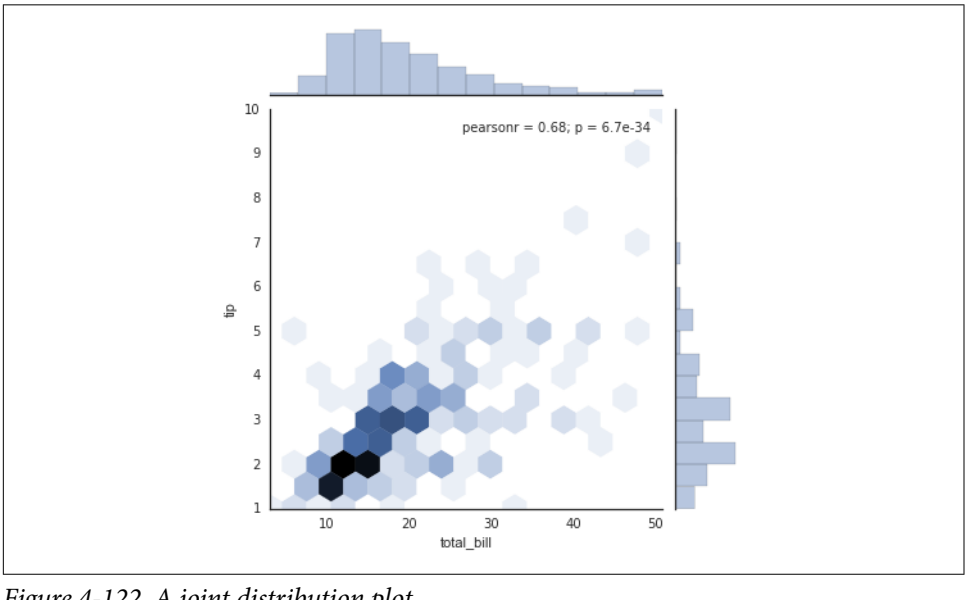

Figure 4-122. A joint distribution plot

The joint plot can even do some automatic kernel density estimation and regression (Figure 4-123):

```
In[18]: sns.jointplot("total_bill", "tip", data=tips, kind='reg');
```

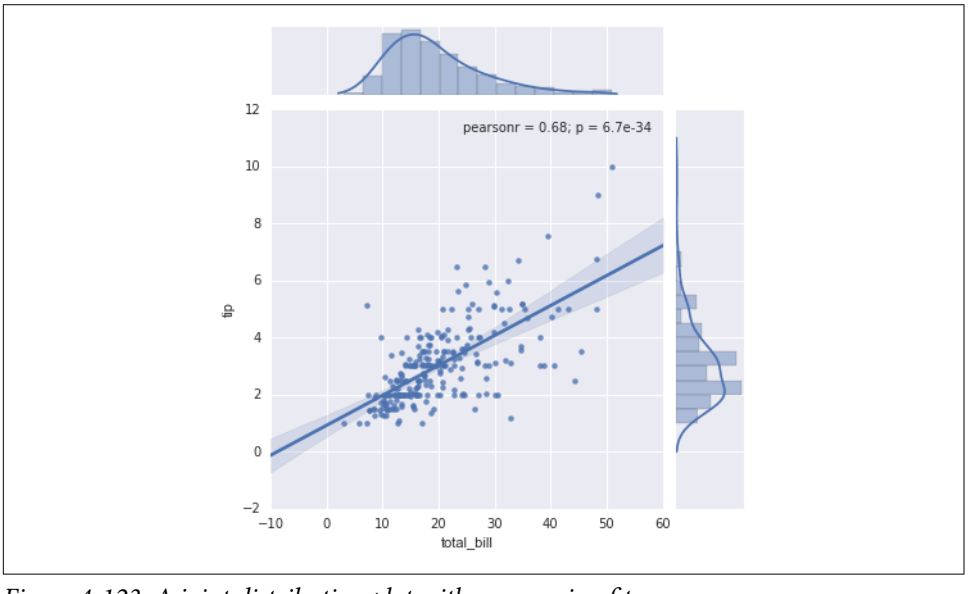

Figure 4-123. A joint distribution plot with a regression fit

Bar plots

Time series can be plotted with `sns.factorplot`. In the following example (visualized in Figure 4-124), we'll use the Planets data that we first saw in "Aggregation and Grouping" on page 158:

```
In[19]: planets = sns.load_dataset('planets')
        planets.head()
```

```
Out[19]:            method  number  orbital_period   mass  distance  year
         0  Radial Velocity       1         269.300   7.10     77.40  2006
         1  Radial Velocity       1         874.774   2.21     56.95  2008
         2  Radial Velocity       1         763.000   2.60     19.84  2011
         3  Radial Velocity       1         326.030  19.40    110.62  2007
         4  Radial Velocity       1         516.220  10.50    119.47  2009
```

```
In[20]: with sns.axes_style('white'):
            g = sns.factorplot("year", data=planets, aspect=2,
                               kind="count", color='steelblue')
            g.set_xticklabels(step=5)
```

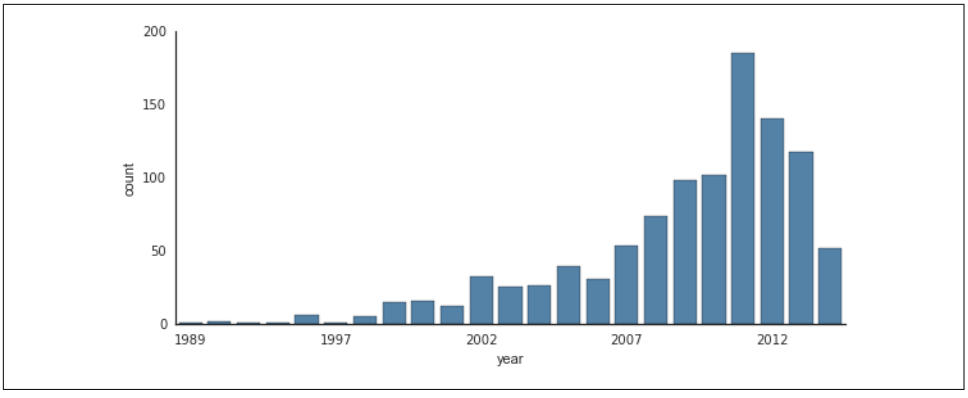

Figure 4-124. A histogram as a special case of a factor plot

We can learn more by looking at the *method* of discovery of each of these planets, as illustrated in Figure 4-125:

```
In[21]: with sns.axes_style('white'):
            g = sns.factorplot("year", data=planets, aspect=4.0, kind='count',
                               hue='method', order=range(2001, 2015))
            g.set_ylabels('Number of Planets Discovered')
```

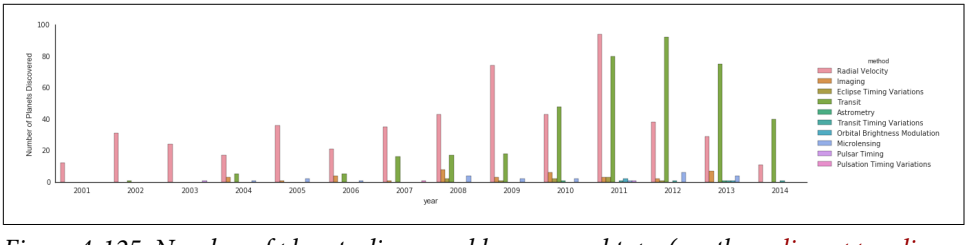

Figure 4-125. Number of planets discovered by year and type (see the online appendix for a full-scale figure)

For more information on plotting with Seaborn, see the Seaborn documentation, a tutorial, and the Seaborn gallery.

Example: Exploring Marathon Finishing Times

Here we'll look at using Seaborn to help visualize and understand finishing results from a marathon. I've scraped the data from sources on the Web, aggregated it and removed any identifying information, and put it on GitHub where it can be downloaded (if you are interested in using Python for web scraping, I would recommend *Web Scraping with Python* by Ryan Mitchell). We will start by downloading the data from the Web, and loading it into Pandas: