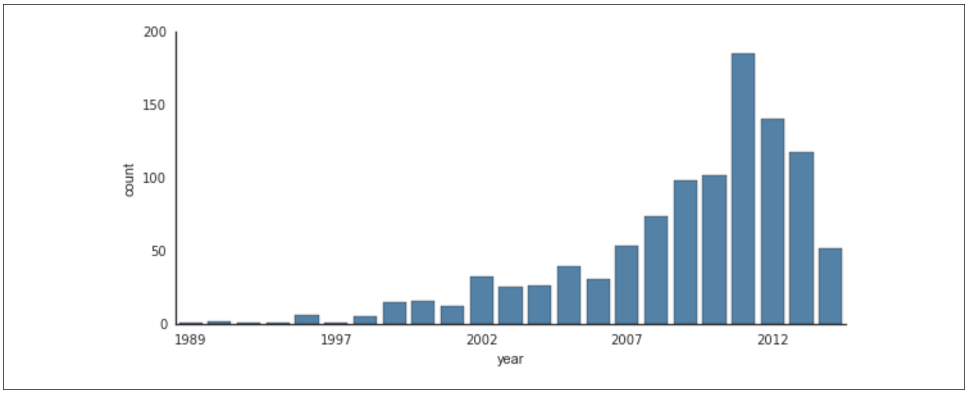

*Figure 4-124. A histogram as a special case of a factor plot*

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')
```

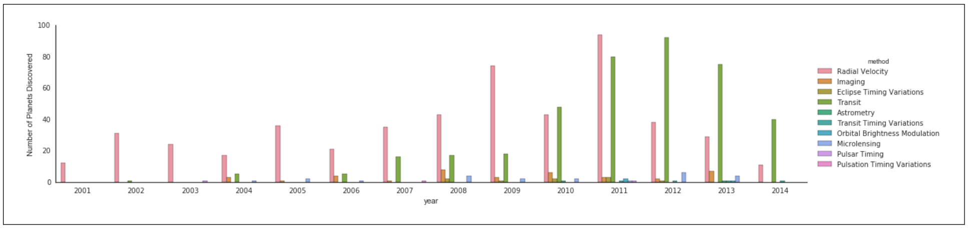

*Figure 4-125. Number of planets discovered by year and type (see the online appendix for a full-scale figure)*

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:

```
In[22]:
# !curl -O https://raw.githubusercontent.com/jakevdp/marathon-data/
# master/marathon-data.csv
```

```
In[23]: data = pd.read_csv('marathon-data.csv')
        data.head()
```

```
Out[23]:    age gender     split     final
         0   33      M  01:05:38  02:08:51
         1   32      M  01:06:26  02:09:28
         2   31      M  01:06:49  02:10:42
         3   38      M  01:06:16  02:13:45
         4   31      M  01:06:32  02:13:59
```

By default, Pandas loaded the time columns as Python strings (type `object`); we can see this by looking at the `dtypes` attribute of the `DataFrame`:

```
In[24]: data.dtypes
```

```
Out[24]: age        int64
         gender    object
         split     object
         final     object
         dtype: object
```

Let's fix this by providing a converter for the times:

```
In[25]: def convert_time(s):
            h, m, s = map(int, s.split(':'))
            return pd.datetools.timedelta(hours=h, minutes=m, seconds=s)

        data = pd.read_csv('marathon-data.csv',
                           converters={'split':convert_time, 'final':convert_time})
        data.head()
```

```
Out[25]:    age gender    split    final
         0   33      M 01:05:38 02:08:51
         1   32      M 01:06:26 02:09:28
         2   31      M 01:06:49 02:10:42
         3   38      M 01:06:16 02:13:45
         4   31      M 01:06:32 02:13:59
```

```
In[26]: data.dtypes
```

```
Out[26]: age                  int64
         gender              object
         split      timedelta64[ns]
         final      timedelta64[ns]
         dtype: object
```

That looks much better. For the purpose of our Seaborn plotting utilities, let's next add columns that give the times in seconds:

```
In[27]: data['split_sec'] = data['split'].astype(int) / 1E9
        data['final_sec'] = data['final'].astype(int) / 1E9
        data.head()
```

```
Out[27]:    age gender    split    final  split_sec  final_sec
         0   33      M 01:05:38 02:08:51     3938.0     7731.0
         1   32      M 01:06:26 02:09:28     3986.0     7768.0
         2   31      M 01:06:49 02:10:42     4009.0     7842.0
         3   38      M 01:06:16 02:13:45     3976.0     8025.0
         4   31      M 01:06:32 02:13:59     3992.0     8039.0
```

To get an idea of what the data looks like, we can plot a `jointplot` over the data (Figure 4-126):

```
In[28]: with sns.axes_style('white'):
            g = sns.jointplot("split_sec", "final_sec", data, kind='hex')
            g.ax_joint.plot(np.linspace(4000, 16000),
                            np.linspace(8000, 32000), ':k')
```

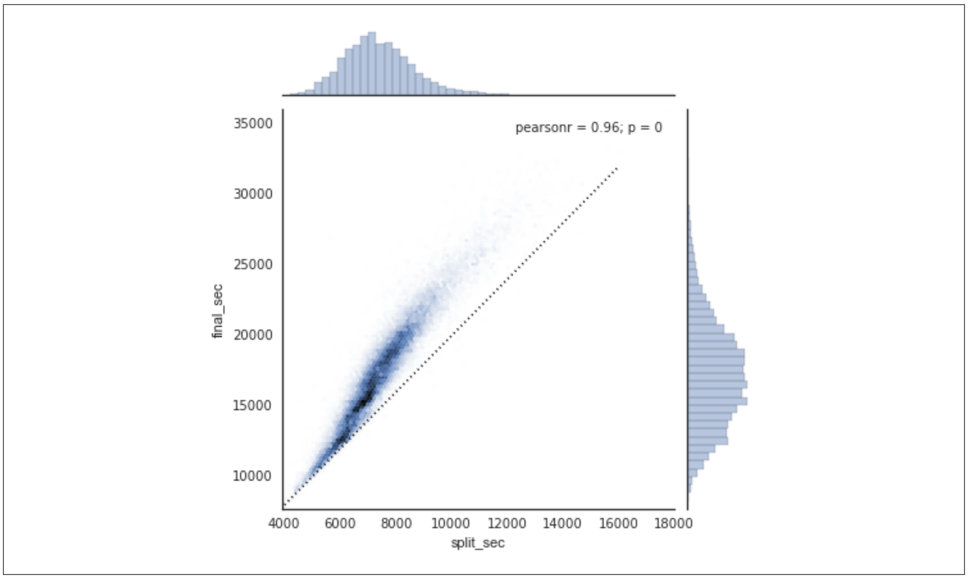

*Figure 4-126. The relationship between the split for the first half-marathon and the finishing time for the full marathon*

The dotted line shows where someone's time would lie if they ran the marathon at a perfectly steady pace. The fact that the distribution lies above this indicates (as you might expect) that most people slow down over the course of the marathon. If you have run competitively, you'll know that those who do the opposite—run faster during the second half of the race—are said to have "negative-split" the race.

Let's create another column in the data, the split fraction, which measures the degree to which each runner negative-splits or positive-splits the race:

```
In[29]: data['split_frac'] = 1 - 2 * data['split_sec'] / data['final_sec']
        data.head()
```

```
Out[29]:    age gender     split     final  split_sec  final_sec  split_frac
         0   33      M 01:05:38 02:08:51     3938.0     7731.0   -0.018756
         1   32      M 01:06:26 02:09:28     3986.0     7768.0   -0.026262
         2   31      M 01:06:49 02:10:42     4009.0     7842.0   -0.022443
         3   38      M 01:06:16 02:13:45     3976.0     8025.0    0.009097
         4   31      M 01:06:32 02:13:59     3992.0     8039.0    0.006842
```

Where this split difference is less than zero, the person negative-split the race by that fraction. Let's do a distribution plot of this split fraction (Figure 4-127):

```
In[30]: sns.distplot(data['split_frac'], kde=False);
        plt.axvline(0, color="k", linestyle="--");
```

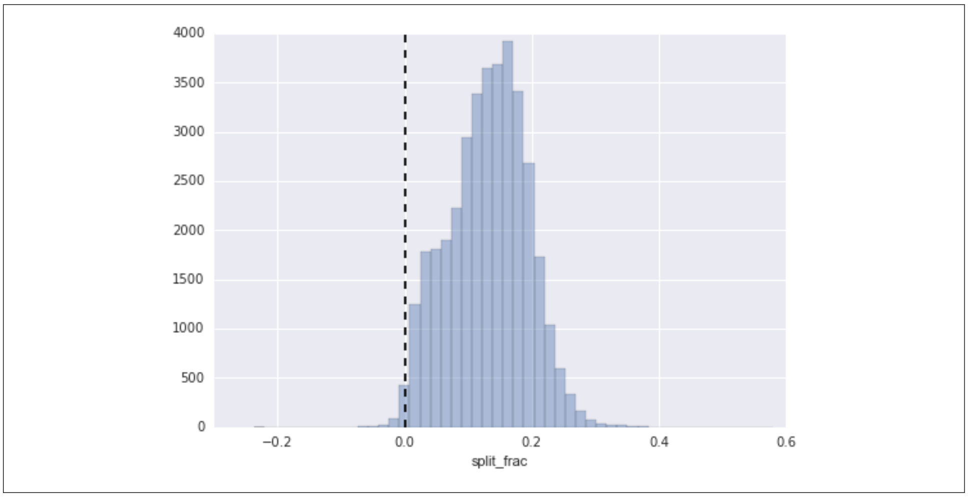

*Figure 4-127. The distribution of split fractions; 0.0 indicates a runner who completed the first and second halves in identical times*

```
In[31]: sum(data.split_frac < 0)
```

```
Out[31]: 251
```

Out of nearly 40,000 participants, there were only 250 people who negative-split their marathon.

Let's see whether there is any correlation between this split fraction and other variables. We'll do this using a `pairgrid`, which draws plots of all these correlations (Figure 4-128):

```
In[32]:
g = sns.PairGrid(data, vars=['age', 'split_sec', 'final_sec', 'split_frac'],
                 hue='gender', palette='RdBu_r')
g.map(plt.scatter, alpha=0.8)
g.add_legend();
```

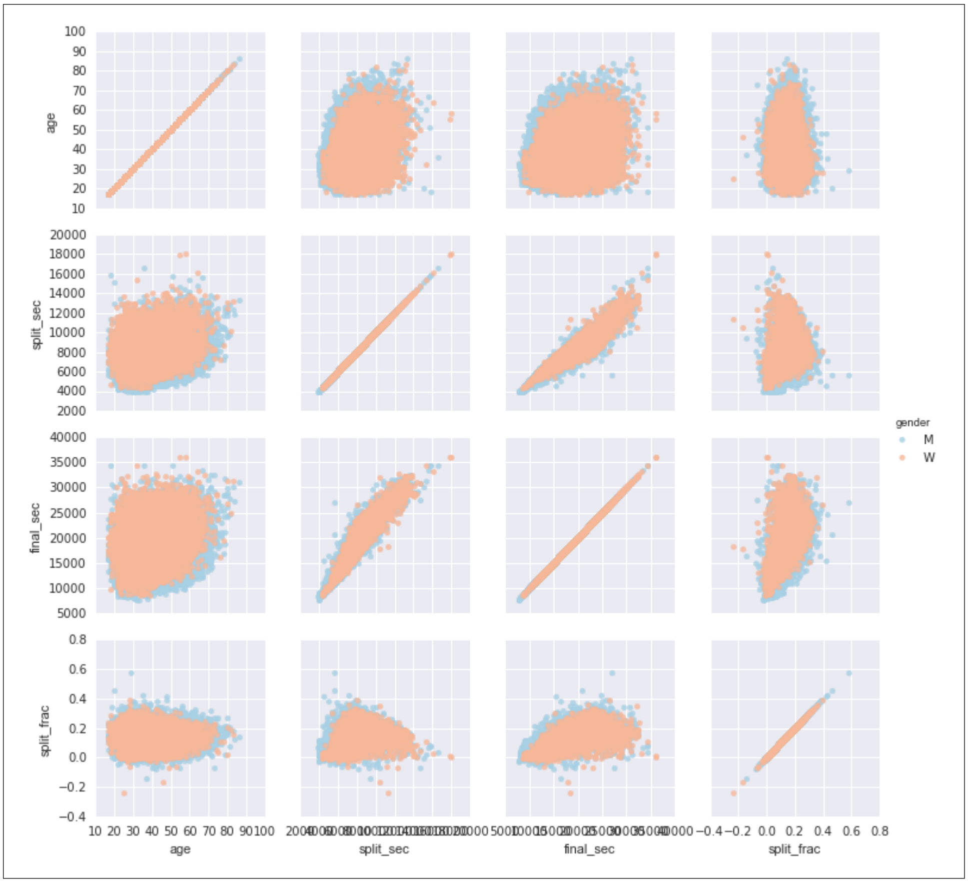

*Figure 4-128. The relationship between quantities within the marathon dataset*

It looks like the split fraction does not correlate particularly with age, but does correlate with the final time: faster runners tend to have closer to even splits on their marathon time. (We see here that Seaborn is no panacea for Matplotlib's ills when it comes to plot styles: in particular, the *x*-axis labels overlap. Because the output is a simple Matplotlib plot, however, the methods in "Customizing Ticks" on page 275 can be used to adjust such things if desired.)

The difference between men and women here is interesting. Let's look at the histogram of split fractions for these two groups (Figure 4-129):

```
In[33]: sns.kdeplot(data.split_frac[data.gender=='M'], label='men', shade=True)
        sns.kdeplot(data.split_frac[data.gender=='W'], label='women', shade=True)
        plt.xlabel('split_frac');
```

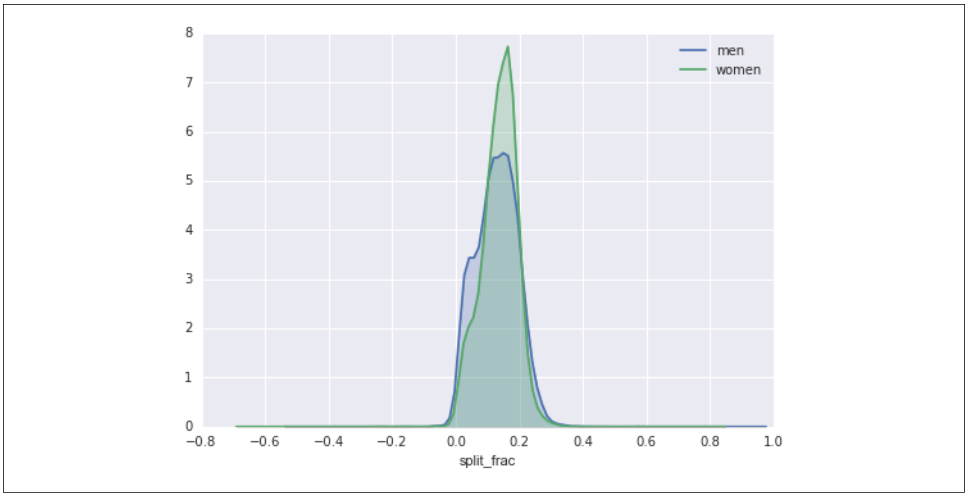

*Figure 4-129. The distribution of split fractions by gender*

The interesting thing here is that there are many more men than women who are running close to an even split! This almost looks like some kind of bimodal distribution among the men and women. Let's see if we can suss out what's going on by looking at the distributions as a function of age.

A nice way to compare distributions is to use a *violin plot* (Figure 4-130):

```
In[34]:
sns.violinplot("gender", "split_frac", data=data,
               palette=["lightblue", "lightpink"]);
```

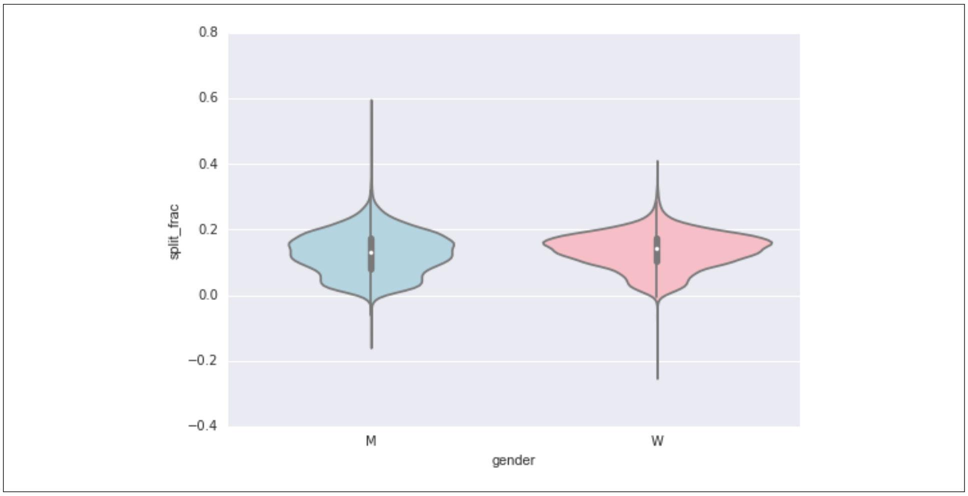

*Figure 4-130. A violin plot showing the split fraction by gender*

This is yet another way to compare the distributions between men and women.

Let's look a little deeper, and compare these violin plots as a function of age. We'll start by creating a new column in the array that specifies the decade of age that each person is in (Figure 4-131):

```
In[35]: data['age_dec'] = data.age.map(lambda age: 10 * (age // 10))
data.head()

Out[35]:
   age gender    split    final  split_sec  final_sec  split_frac  age_dec
0   33      M 01:05:38 02:08:51     3938.0     7731.0   -0.018756       30
1   32      M 01:06:26 02:09:28     3986.0     7768.0   -0.026262       30
2   31      M 01:06:49 02:10:42     4009.0     7842.0   -0.022443       30
3   38      M 01:06:16 02:13:45     3976.0     8025.0    0.009097       30
4   31      M 01:06:32 02:13:59     3992.0     8039.0    0.006842       30

In[36]:
men = (data.gender == 'M')
women = (data.gender == 'W')

with sns.axes_style(style=None):
    sns.violinplot("age_dec", "split_frac", hue="gender", data=data,
                   split=True, inner="quartile",
                   palette=["lightblue", "lightpink"]);
```

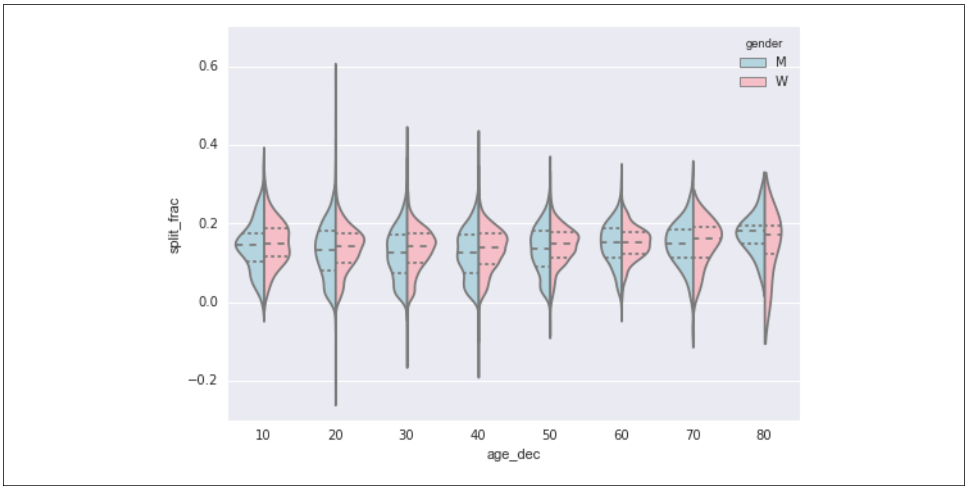

*Figure 4-131. A violin plot showing the split fraction by gender and age*

Looking at this, we can see where the distributions of men and women differ: the split distributions of men in their 20s to 50s show a pronounced over-density toward lower splits when compared to women of the same age (or of any age, for that matter).

Also surprisingly, the 80-year-old women seem to outperform *everyone* in terms of their split time. This is probably due to the fact that we're estimating the distribution from small numbers, as there are only a handful of runners in that range:

```
In[38]: (data.age > 80).sum()
```

```
Out[38]: 7
```

Back to the men with negative splits: who are these runners? Does this split fraction correlate with finishing quickly? We can plot this very easily. We'll use `regplot`, which will automatically fit a linear regression to the data (Figure 4-132):

```
In[37]: g = sns.lmplot('final_sec', 'split_frac', col='gender', data=data,
                       markers=".", scatter_kws=dict(color='c'))
        g.map(plt.axhline, y=0.1, color="k", ls=":");
```

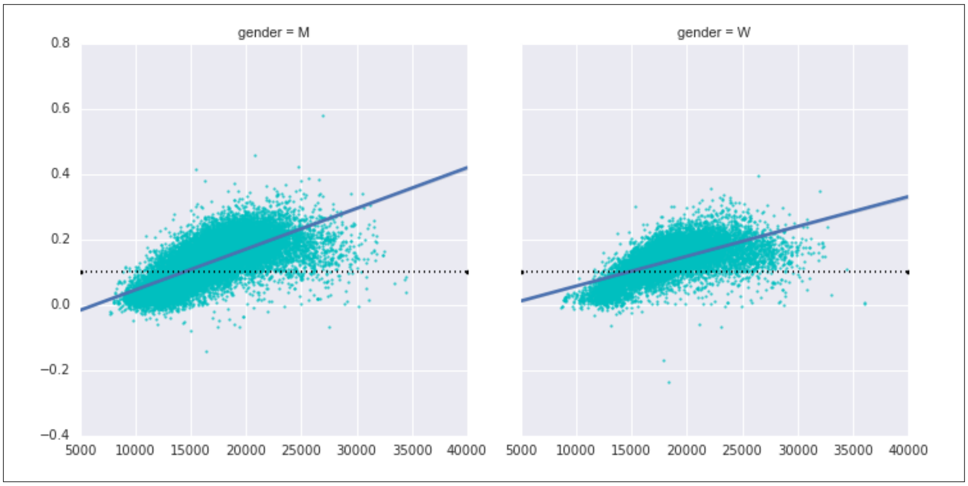

*Figure 4-132. Split fraction versus finishing time by gender*

Apparently the people with fast splits are the elite runners who are finishing within ~15,000 seconds, or about 4 hours. People slower than that are much less likely to have a fast second split.

## Further Resources

### Matplotlib Resources

A single chapter in a book can never hope to cover all the available features and plot types available in Matplotlib. As with other packages we've seen, liberal use of IPython's tab-completion and help functions (see "Help and Documentation in IPython" on page 3) can be very helpful when you're exploring Matplotlib's API. In addition, Matplotlib's online documentation can be a helpful reference. See in particular the