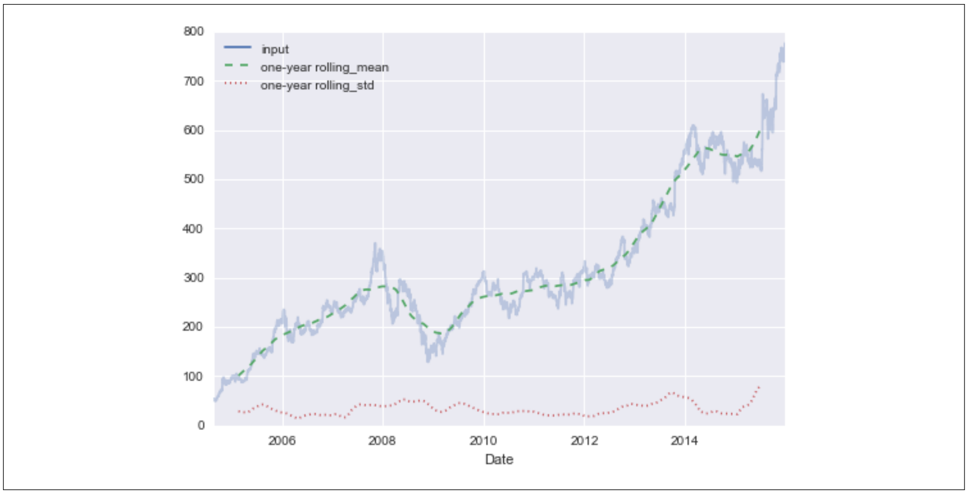

Figure 3-10. Rolling statistics on Google stock prices

As with `groupby` operations, the `aggregate()` and `apply()` methods can be used for custom rolling computations.

Where to Learn More

This section has provided only a brief summary of some of the most essential features of time series tools provided by Pandas; for a more complete discussion, you can refer to the "Time Series/Date" section of the Pandas online documentation.

Another excellent resource is the textbook *Python for Data Analysis* by Wes McKinney (O'Reilly, 2012). Although it is now a few years old, it is an invaluable resource on the use of Pandas. In particular, this book emphasizes time series tools in the context of business and finance, and focuses much more on particular details of business calendars, time zones, and related topics.

As always, you can also use the IPython help functionality to explore and try further options available to the functions and methods discussed here. I find this often is the best way to learn a new Python tool.

Example: Visualizing Seattle Bicycle Counts

As a more involved example of working with some time series data, let's take a look at bicycle counts on Seattle's Fremont Bridge. This data comes from an automated bicycle counter, installed in late 2012, which has inductive sensors on the east and west sidewalks of the bridge. The hourly bicycle counts can be downloaded from *http://data.seattle.gov/*; here is the direct link to the dataset.

As of summer 2016, the CSV can be downloaded as follows:

```
In[34]:
# !curl -o FremontBridge.csv
# https://data.seattle.gov/api/views/65db-xm6k/rows.csv?accessType=DOWNLOAD
```

Once this dataset is downloaded, we can use Pandas to read the CSV output into a `DataFrame`. We will specify that we want the `Date` as an index, and we want these dates to be automatically parsed:

```
In[35]:
data = pd.read_csv('FremontBridge.csv', index_col='Date', parse_dates=True)
data.head()
```

```
Out[35]:                             Fremont Bridge West Sidewalk  \\
         Date
         2012-10-03 00:00:00                                  4.0
         2012-10-03 01:00:00                                  4.0
         2012-10-03 02:00:00                                  1.0
         2012-10-03 03:00:00                                  2.0
         2012-10-03 04:00:00                                  6.0

                              Fremont Bridge East Sidewalk
         Date
         2012-10-03 00:00:00                                  9.0
         2012-10-03 01:00:00                                  6.0
         2012-10-03 02:00:00                                  1.0
         2012-10-03 03:00:00                                  3.0
         2012-10-03 04:00:00                                  1.0
```

For convenience, we'll further process this dataset by shortening the column names and adding a "Total" column:

```
In[36]: data.columns = ['West', 'East']
        data['Total'] = data.eval('West + East')
```

Now let's take a look at the summary statistics for this data:

```
In[37]: data.dropna().describe()
```

```
Out[37]:                West          East         Total
         count  33544.000000  33544.000000  33544.000000
         mean      61.726568     53.541706    115.268275
         std       83.210813     76.380678    144.773983
         min        0.000000      0.000000      0.000000
         25%        8.000000      7.000000     16.000000
         50%       33.000000     28.000000     64.000000
         75%       80.000000     66.000000    151.000000
         max      825.000000    717.000000   1186.000000
```

Visualizing the data

We can gain some insight into the dataset by visualizing it. Let's start by plotting the raw data (Figure 3-11):

```
In[38]: %matplotlib inline
        import seaborn; seaborn.set()
```

```
In[39]: data.plot()
        plt.ylabel('Hourly Bicycle Count');
```

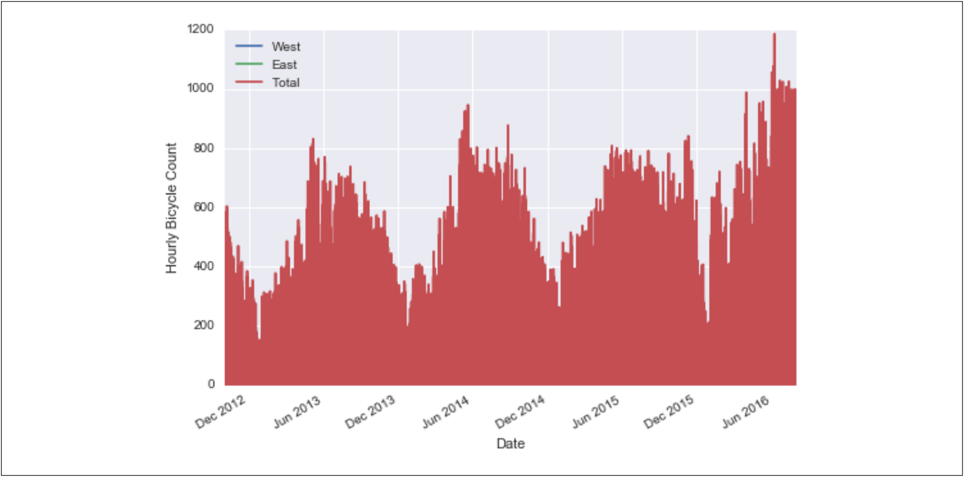

Figure 3-11. Hourly bicycle counts on Seattle's Fremont bridge

The ~25,000 hourly samples are far too dense for us to make much sense of. We can gain more insight by resampling the data to a coarser grid. Let's resample by week (Figure 3-12):

```
In[40]: weekly = data.resample('W').sum()
        weekly.plot(style=[':', '--', '-'])
        plt.ylabel('Weekly bicycle count');
```

This shows us some interesting seasonal trends: as you might expect, people bicycle more in the summer than in the winter, and even within a particular season the bicycle use varies from week to week (likely dependent on weather; see "In Depth: Linear Regression" on page 390 where we explore this further).

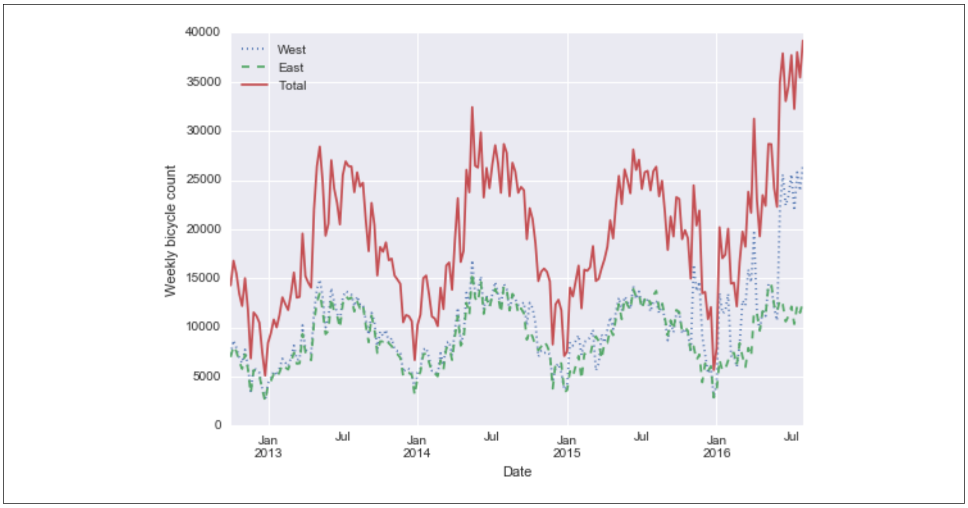

Figure 3-12. Weekly bicycle crossings of Seattle's Fremont bridge

Another way that comes in handy for aggregating the data is to use a rolling mean, utilizing the `pd.rolling_mean()` function. Here we'll do a 30-day rolling mean of our data, making sure to center the window (Figure 3-13):

```
In[41]: daily = data.resample('D').sum()
        daily.rolling(30, center=True).sum().plot(style=[':', '--', '-'])
        plt.ylabel('mean hourly count');
```

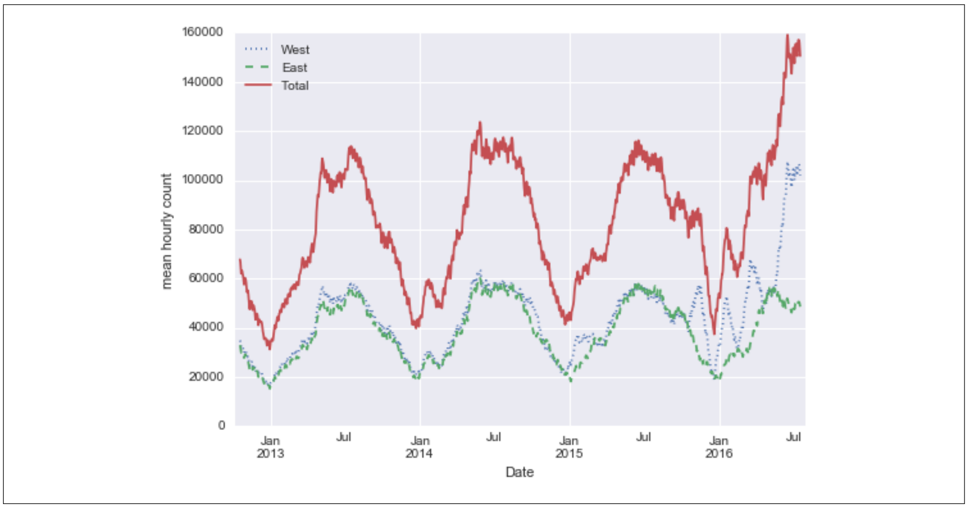

Figure 3-13. Rolling mean of weekly bicycle counts

The jaggedness of the result is due to the hard cutoff of the window. We can get a smoother version of a rolling mean using a window function—for example, a Gaussian window. The following code (visualized in Figure 3-14) specifies both the width of the window (we chose 50 days) and the width of the Gaussian within the window (we chose 10 days):

```
In[42]:
daily.rolling(50, center=True,
              win_type='gaussian').sum(std=10).plot(style=[':', '--', '-']);
```

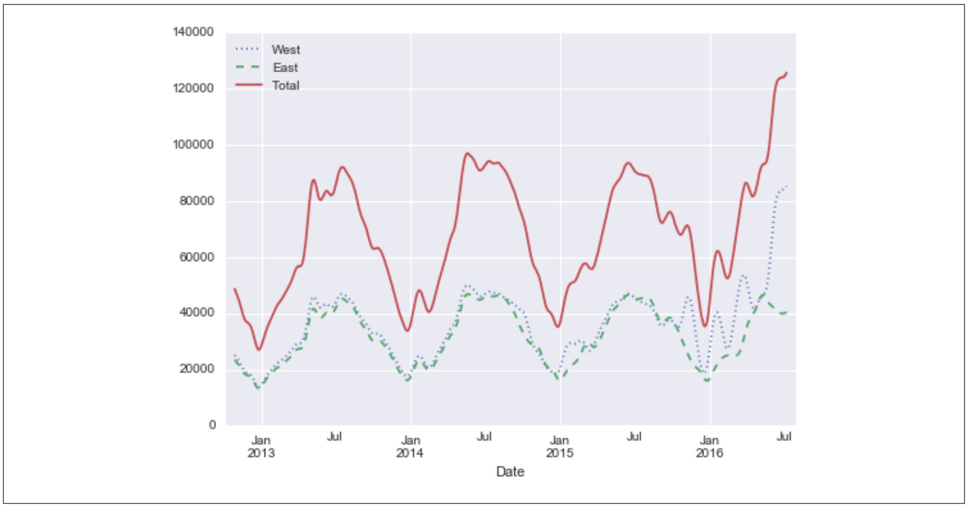

Figure 3-14. Gaussian smoothed weekly bicycle counts

Digging into the data

While the smoothed data views in Figure 3-14 are useful to get an idea of the general trend in the data, they hide much of the interesting structure. For example, we might want to look at the average traffic as a function of the time of day. We can do this using the GroupBy functionality discussed in "Aggregation and Grouping" on page 158 (Figure 3-15):

```
In[43]: by_time = data.groupby(data.index.time).mean()
        hourly_ticks = 4 * 60 * 60 * np.arange(6)
        by_time.plot(xticks=hourly_ticks, style=[':', '--', '-']);
```

The hourly traffic is a strongly bimodal distribution, with peaks around 8:00 in the morning and 5:00 in the evening. This is likely evidence of a strong component of commuter traffic crossing the bridge. This is further evidenced by the differences between the western sidewalk (generally used going toward downtown Seattle), which peaks more strongly in the morning, and the eastern sidewalk (generally used going away from downtown Seattle), which peaks more strongly in the evening.

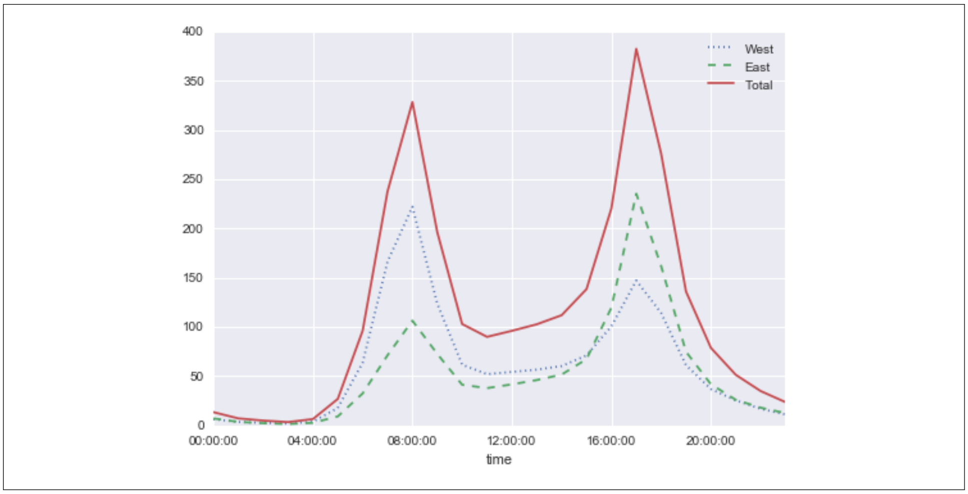

Figure 3-15. Average hourly bicycle counts

We also might be curious about how things change based on the day of the week. Again, we can do this with a simple `groupby` (Figure 3-16):

```
In[44]: by_weekday = data.groupby(data.index.dayofweek).mean()
        by_weekday.index = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun']
        by_weekday.plot(style=[':', '--', '-']);
```

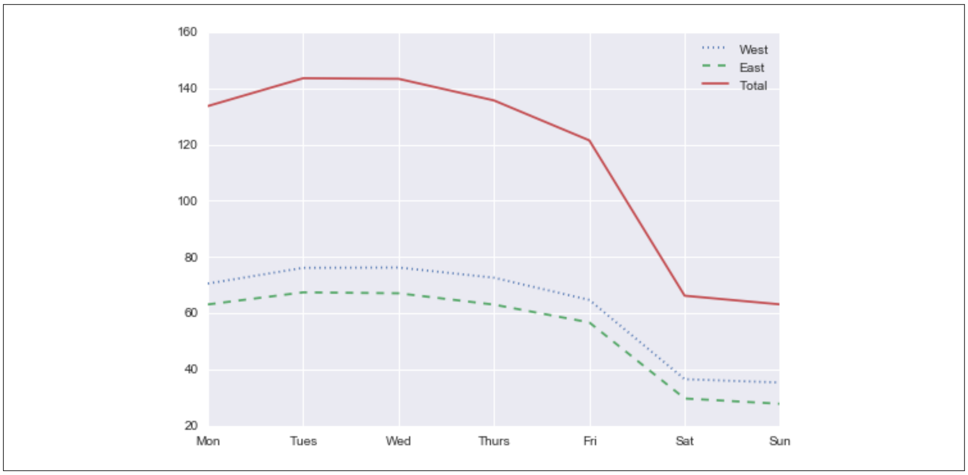

Figure 3-16. Average daily bicycle counts

This shows a strong distinction between weekday and weekend totals, with around twice as many average riders crossing the bridge on Monday through Friday than on Saturday and Sunday.

With this in mind, let's do a compound `groupby` and look at the hourly trend on weekdays versus weekends. We'll start by grouping by both a flag marking the weekend, and the time of day:

```
In[45]: weekend = np.where(data.index.weekday < 5, 'Weekday', 'Weekend')
        by_time = data.groupby([weekend, data.index.time]).mean()
```

Now we'll use some of the Matplotlib tools described in "Multiple Subplots" on page 262 to plot two panels side by side (Figure 3-17):

```
In[46]: import matplotlib.pyplot as plt
        fig, ax = plt.subplots(1, 2, figsize=(14, 5))
        by_time.ix['Weekday'].plot(ax=ax[0], title='Weekdays',
                                   xticks=hourly_ticks, style=[':', '--', '-'])
        by_time.ix['Weekend'].plot(ax=ax[1], title='Weekends',
                                   xticks=hourly_ticks, style=[':', '--', '-']);
```

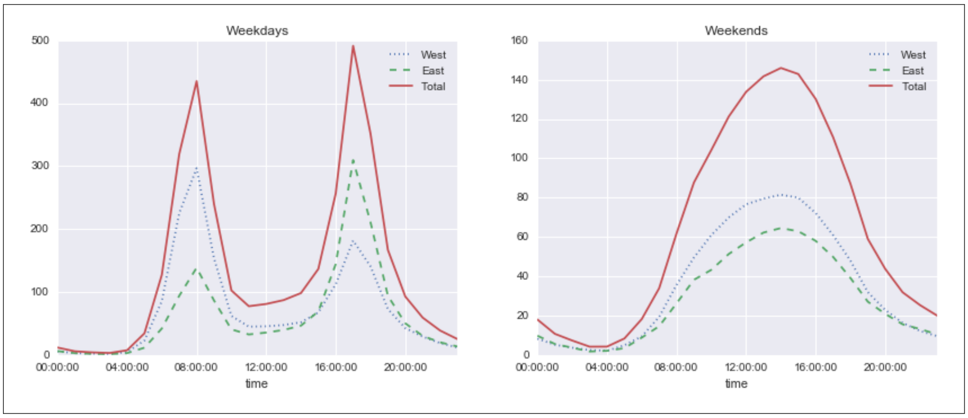

Figure 3-17. Average hourly bicycle counts by weekday and weekend

The result is very interesting: we see a bimodal commute pattern during the work week, and a unimodal recreational pattern during the weekends. It would be interesting to dig through this data in more detail, and examine the effect of weather, temperature, time of year, and other factors on people's commuting patterns; for further discussion, see my blog post "Is Seattle Really Seeing an Uptick In Cycling?", which uses a subset of this data. We will also revisit this dataset in the context of modeling in "In Depth: Linear Regression" on page 390.

High-Performance Pandas: eval() and query()

As we've already seen in previous chapters, the power of the PyData stack is built upon the ability of NumPy and Pandas to push basic operations into C via an intuitive syntax: examples are vectorized/broadcasted operations in NumPy, and grouping-type operations in Pandas. While these abstractions are efficient and effec-