另一个优秀的资源是 Wes McKinney（Pandas 创建者）所著的《利用 Python 进行数据分析》。虽然这本书已经有些年头了，但仍然是学习 Pandas 的好资源，尤其是这本书重点介绍了时间序列工具在商业与金融业务中的应用，作者用大量笔墨介绍了工作日历、时区和相关主题的具体内容。

你当然可以用 IPython 的帮助功能来浏览和深入探索上面介绍过的函数与方法，我个人认为这是学习各种 Python 工具的最佳途径。

3.12.7 案例：美国西雅图自行车统计数据的可视化

下面来介绍一个比较复杂的时间序列数据，统计自 2012 年以来每天经过美国西雅图弗莱蒙特桥（http://www.openstreetmap.org/#map=17/47.64813/-122.34965）上的自行车的数量，数据由安装在桥东西两侧人行道的传感器采集。小时统计数据可以在 http://data.seattle.gov/ 下载，还有一个数据集的直接下载链接 https://data.seattle.gov/Transportation/Fremont-Bridge-Hourly-Bicycle-Counts-by-Month-Octo/65db-xm6k。

截至 2016 年夏，CSV 数据可以用以下命令下载：

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

下好数据之后，可以用 Pandas 读取 CSV 文件获取一个 `DataFrame`。我们将 `Date` 作为时间索引，并希望这些日期可以被自动解析：

```
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
```

为了方便后面的计算，缩短数据集的列名，并新增一个 Total 列：

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

现在来看看这三列的统计值：

```
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
```

01. 数据可视化

通过可视化，我们可以对数据集有一些直观的认识。先为原始数据画图（如图 3-11 所示）：

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

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

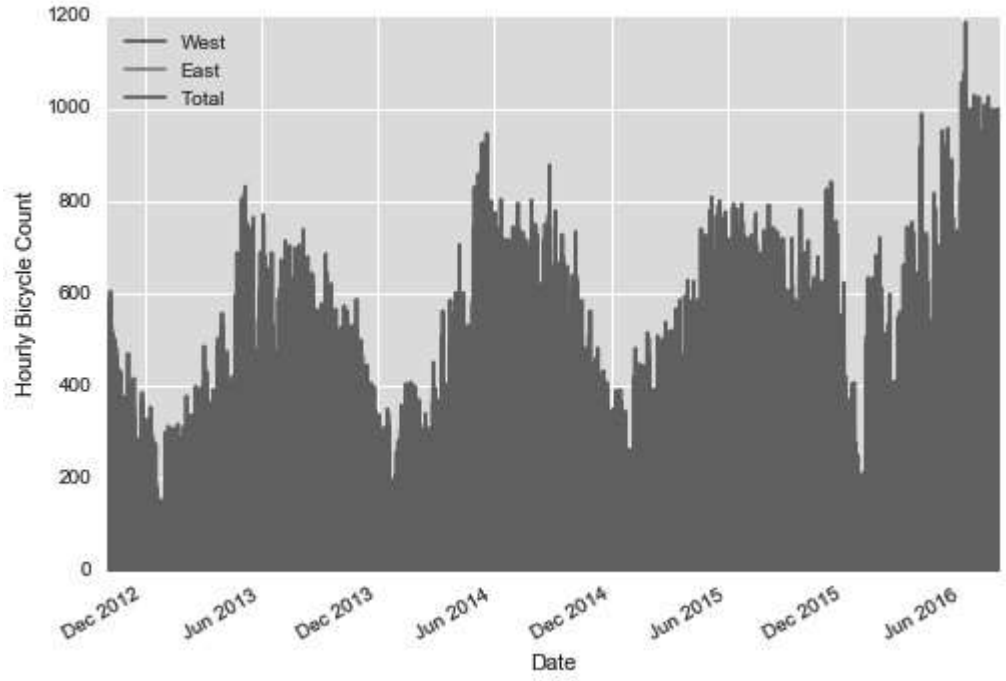

图 **3-11**：弗莱蒙特桥每小时通行的自行车数量

在图中显示大约 25 000 小时的样本数据对我们来说实在太多了，因此可以通过重新取样将数据转换成更大的颗粒度，比如按周累计（如图 3-12 所示）：

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

这就显示出一些季节性的特征了。正如你所想，夏天骑自行车的人比冬天多，而且某个季节中每一周的自行车数量也在变化（可能与天气有关，详情请参见 5.6 节）。

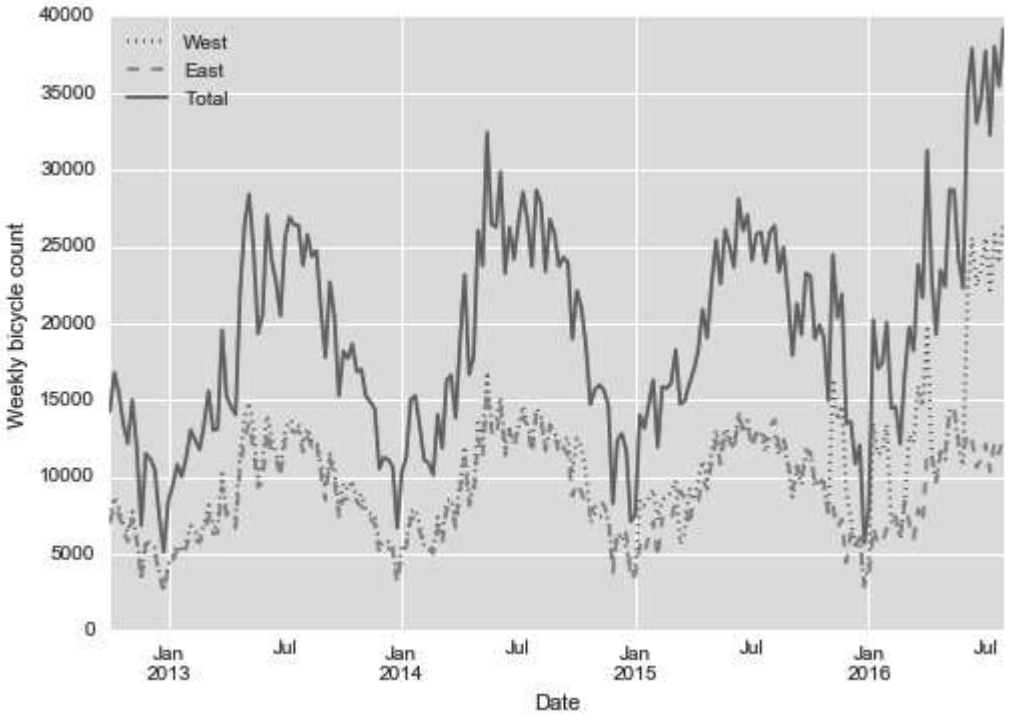

图 **3-12**：弗莱蒙特桥每周通行的自行车数量

另一种对数据进行累计的简便方法是用 `pd.rolling_mean()`[4] 函数求移动平均值。下面将计算数据的 30 日移动均值，并让图形在窗口居中显示（`center=True`）（如图 3-13 所示）：

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

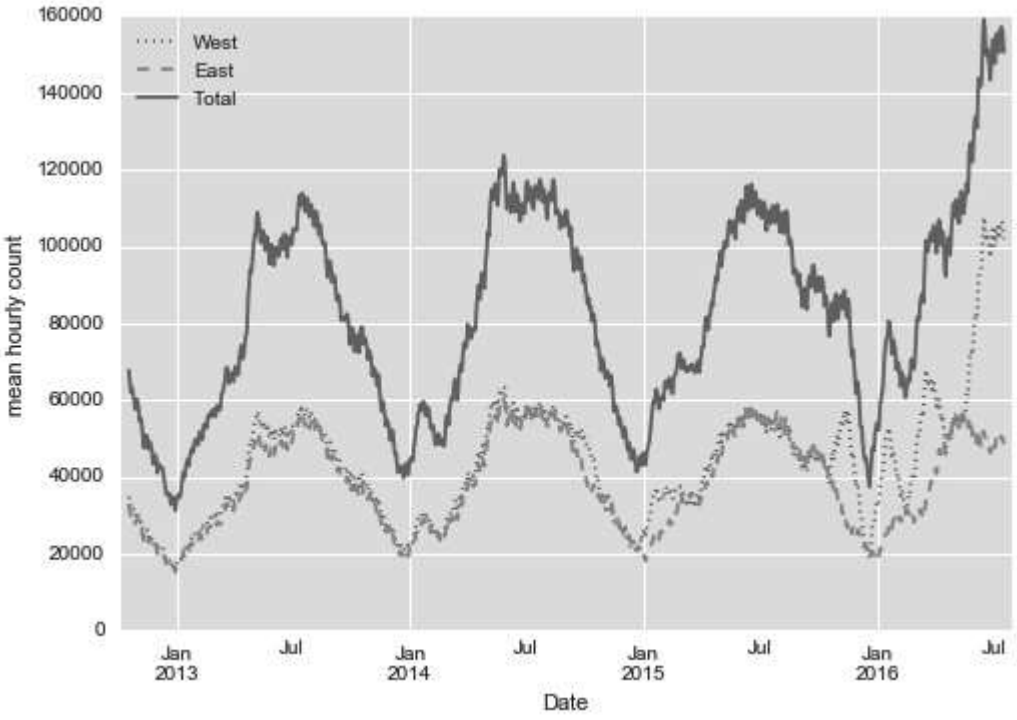

图 **3-13**：每 **30** 日自行车的移动日均值

由于窗口太小，现在的图形还不太平滑。我们可以用另一个移动均值的方法获得更平滑的图形，例如高斯分布时间窗口。下面的代码（可视化后如图 3-14 所示）将设置窗口的宽度（选择 50 天）和窗口内高斯平滑的宽度（选择 10 天）：

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

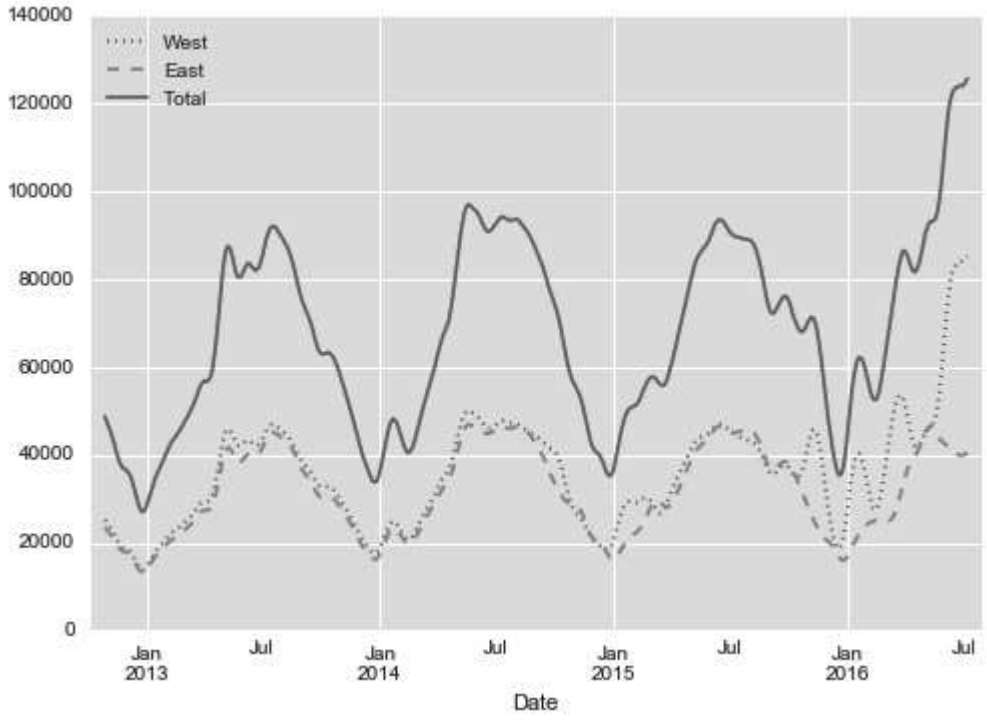

图 **3-14**：用高斯平滑方法处理每周自行车的移动均值

02. 深入挖掘数据

虽然我们已经从图 3-14 的平滑数据图观察到了数据的总体趋势，但是它们还隐藏了一些有趣的特征。例如，我们可能希望观察单日内的小时均值流量，这可以通过 GroupBy（详情请参见 3.9 节）操作来解决（如图 3-15 所示）：

```
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=[':', '--', '-']);
```

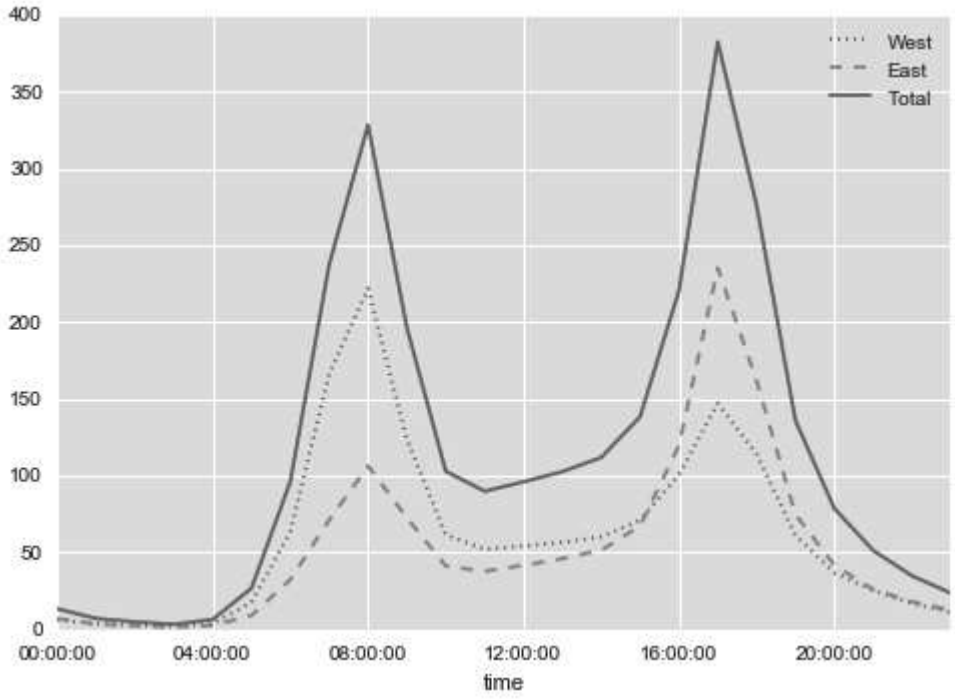

图 **3-15**：每小时的自行车流量

小时均值流量呈现出十分明显的双峰分布特征，早间峰值在上午 8 点，晚间峰值在下午 5 点。这充分反映了过桥上下班往返自行车流量的特征。进一步分析会发现，桥西的高峰在早上（因为人们每天会到西雅图的市中心上班），而桥东的高峰在下午（下班再从市中心离开）。

我们可能还会对周内每天的变化产生兴趣，这时依然可以通过一个简单的 groupby 来实现（如图 3-16 所示）：

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

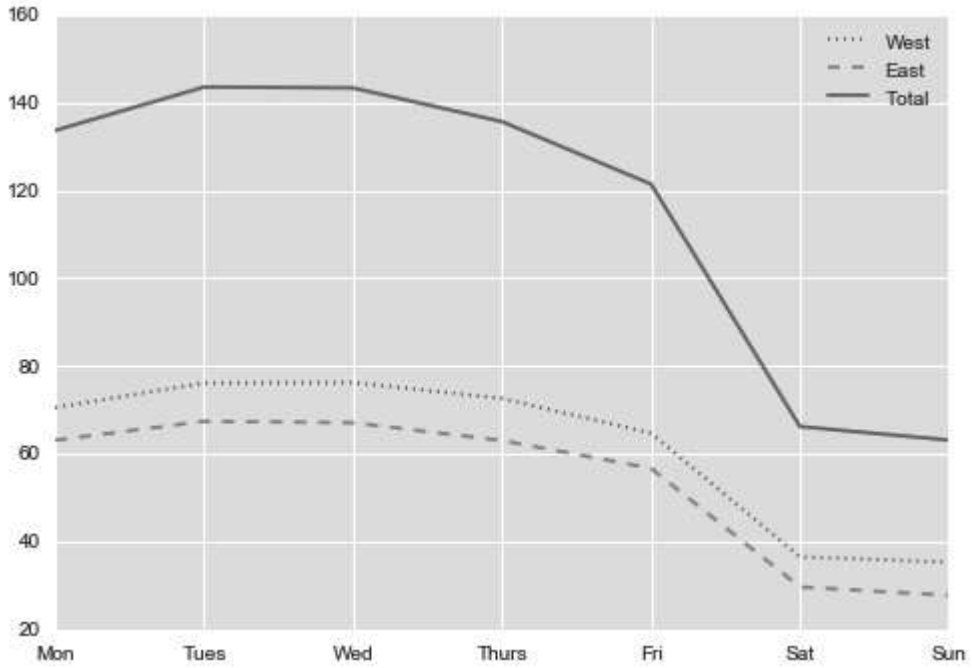

图 **3-16**：每周每天的自行车流量

工作日与周末的自行车流量差十分显著，周一到周五通过的自行车差不多是周六、周日的两倍。

看到这个特征之后，让我们用一个复合 groupby 来观察一周内工作日与双休日每小时的数据。用一个标签表示双休日和工作日的不同小时：

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

现在用一些 Matplotlib 工具（详情请参见 4.10 节）画出两张图（如图 3-17 所示）：

```
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=[':', '-
```

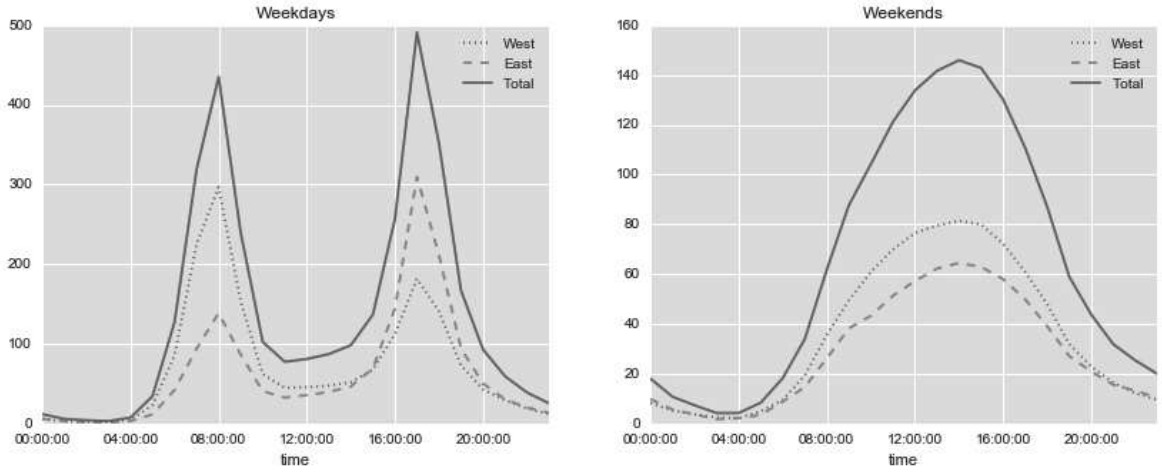

图 **3-17**：工作日与双休日每小时的自行车流量

结果很有意思，我们会发现工作日的自行车流量呈双峰通勤模式（bimodal commute pattern），而到了周末就变成了单峰娱乐模式（unimodal recreational pattern）。假如继续挖掘数据应该还会发现更多有趣的信息，比如研究天气、温度、一年中的不同时间以及其他因素对人们通勤模式的影响。关于更深入的分析内容，请参考我的博文“Is Seattle Really Seeing an Uptick In Cycling?”（https://jakevdp.github.io/blog/2014/06/10/is-seattle-really-seeing-an-uptick-in-cycling/），里面用数据的子集作了一些分析。我们将在 5.6 节继续使用这个数据集。

[4]原书代码与正文不符。作者在正文中说“用 `pd.rolling_meaning()` 函数”，但作者代码中 `daily.rolling(30,center=True).sum()` 等价于 `pd.rolling_sum()`。另外，Pandas 文档提到，`pd.rolling_mean` 方法即将被废弃，用 `DataFrame.rolling(center=False,window=D).mean()` 的形式代替 `pd.rolling_mean()`。考虑到原文图题是“30 天自行车数量”，因此按照 30 天的日均值作相应的修改。——译者注