

# 绘图和可视化

绘图是数据分析工作中最重要的任务之一，是探索过程的一部分，例如，帮助我们找出异常值、必要的数据转换、得出有关模型的idea等。此外，还可以利用诸如d3.js (<http://d3js.org/>) 之类的工具为Web应用构建交互式图像。Python有许多可视化工具（参见本章末尾），但是我主要讲解matplotlib (<http://matplotlib.sourceforge.net>)。

matplotlib是一个用于创建出版质量图表的桌面绘图包（主要是2D方面）。该项目是由John Hunter于2002年启动的，其目的是为Python构建一个MATLAB式的绘图接口。从那时起，John Hunter、Fernando Pérez（IPython的创始人）等许多人就一起合作，共同致力于将IPython和matplotlib结合起来以提供一种功能丰富且高效的科学计算环境。如果结合使用一种GUI工具包（如IPython），matplotlib还具有诸如缩放和平移等交互功能。它不仅支持各种操作系统上许多不同的GUI后端，而且还能将图片导出为各种常见的矢量（vector）和光栅（raster）图：PDF、SVG、JPG、PNG、BMP、GIF等。本书中的大部分图形都是用它生成的。

matplotlib还有许多插件工具集，如用于3D图形的`matplotlib3d`以及用于地图和投影的`basemap`。我将在本章末尾介绍一个利用`basemap`在地图上绘制数据和读取shapefiles的例子。

要使用本章中的代码示例，请确保你的IPython是以Pylab模式启动的（`ipython --pylab`），或通过%gui魔术命令打开了GUI事件循环集成。

## matplotlib API入门

使用matplotlib的办法有很多种，最常用的方式是Pylab模式的IPython（`ipython`



--pylab)。这样会将IPython配置为使用你所指定的matplotlib GUI后端（Tk、wxPython、PyQt、Mac OS X native、GTK）。对大部分用户而言，默认的后端就已经够用了。Pylab模式还会向IPython引入一大堆模块和函数以提供一种更接近于MATLAB的界面（见图8-1）。绘制一张简单的图表即可测试是否一切准备就绪：

```
plot(np.arange(10))
```

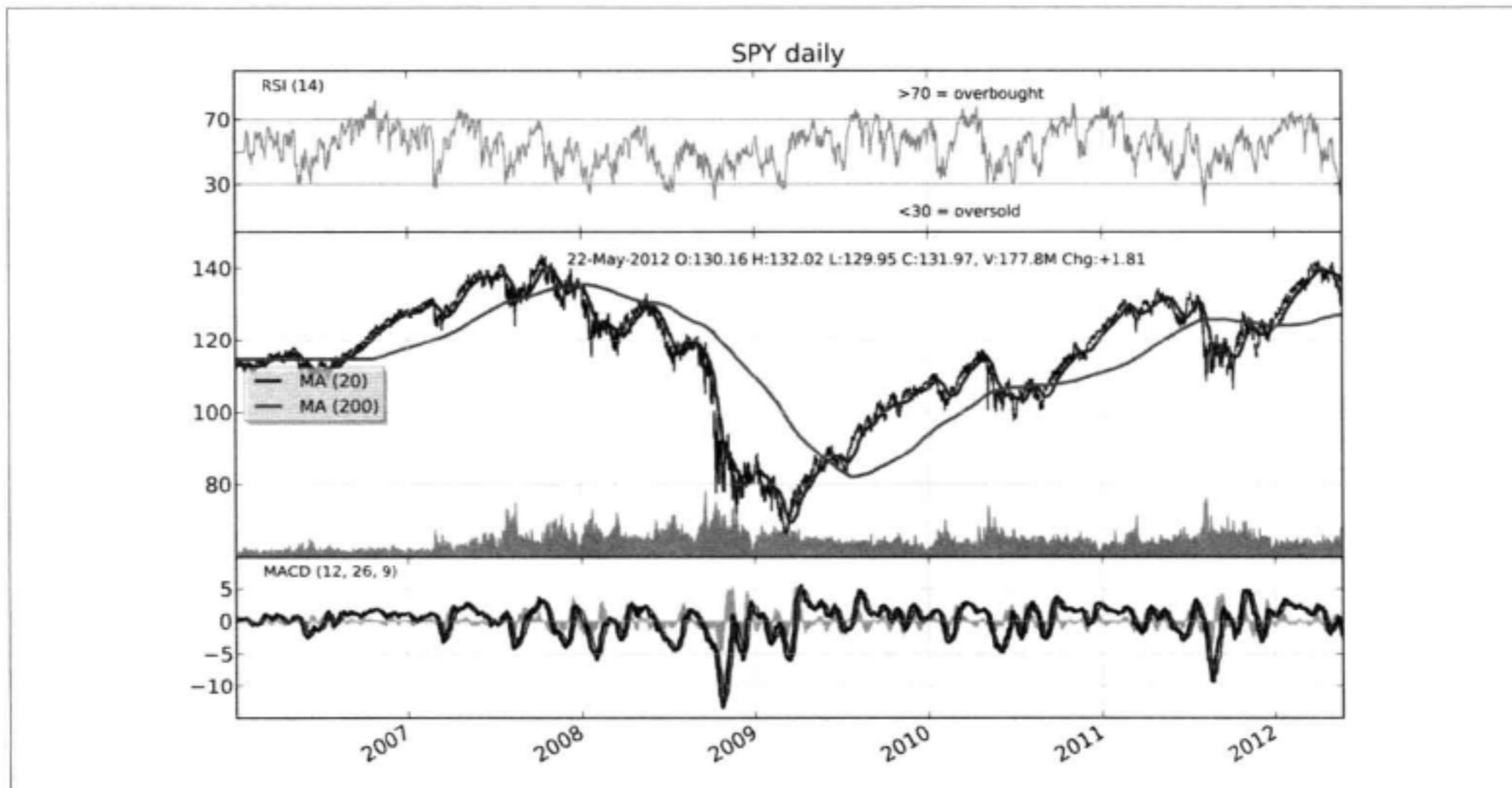


图8-1：一张比较复杂的matplotlib金融曲线图

如果一切都是正确的，就会弹出一个新窗口，其中绘制的是一条直线。你可以用鼠标或输入close()来关闭它。matplotlib API函数（如plot和close）都位于matplotlib.pyplot模块中，其通常的引入约定是：

```
import matplotlib.pyplot as plt
```

虽然pandas的绘图函数（稍后介绍）能够处理许多普通的绘图任务，但如果需要自定义一些高级功能的话就必须学习matplotlib API。

---

**注意：** 虽然本书没有详细地讨论matplotlib的各种功能，但足以将你引入门。matplotlib的示例库和文档是成为绘图高手的最佳学习资源。

## Figure和Subplot

matplotlib的图像都位于Figure对象中。你可以用plt.figure创建一个新的Figure：

```
In [13]: fig = plt.figure()
```



这时会弹出一个空窗口。plt.figure有一些选项，特别是figsize，它用于确保当图片保存到磁盘时具有一定的大小和纵横比。matplotlib中的Figure还支持一种MATLAB式的编号架构（例如plt.figure(2)）。通过plt.gcf()即可得到当前Figure的引用。

不能通过空Figure绘图。必须用add\_subplot创建一个或多个subplot才行：

```
In [14]: ax1 = fig.add_subplot(2, 2, 1)
```

这条代码的意思是：图像应该是 $2 \times 2$ 的，且当前选中的是4个subplot中的第一个（编号从1开始）。如果再把后面两个subplot也创建出来，最终得到的图像如图8-2所示。

```
In [15]: ax2 = fig.add_subplot(2, 2, 2)
```

```
In [16]: ax3 = fig.add_subplot(2, 2, 3)
```

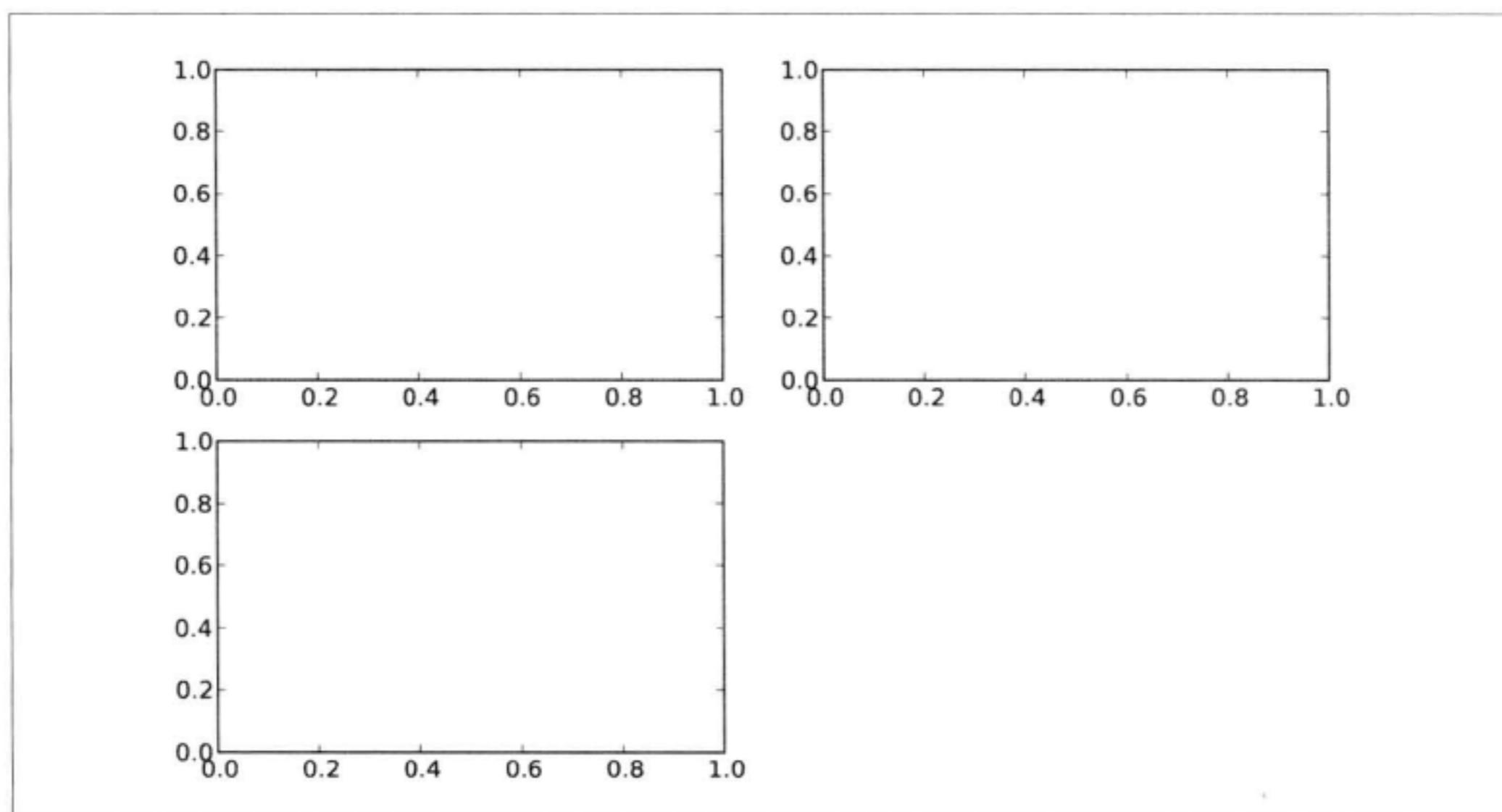


图8-2：带有三个subplot的Figure

如果这时发出一条绘图命令（如plt.plot([1.5, 3.5, -2, 1.6]))，matplotlib就会在最后一个用过的subplot（如果没有则创建一个）上进行绘制。因此，如果我们执行下列命令，你就会得到如图8-3所示的结果：

```
In [17]: from numpy.random import randn
```

```
In [18]: plt.plot(randn(50).cumsum(), 'k--')
```



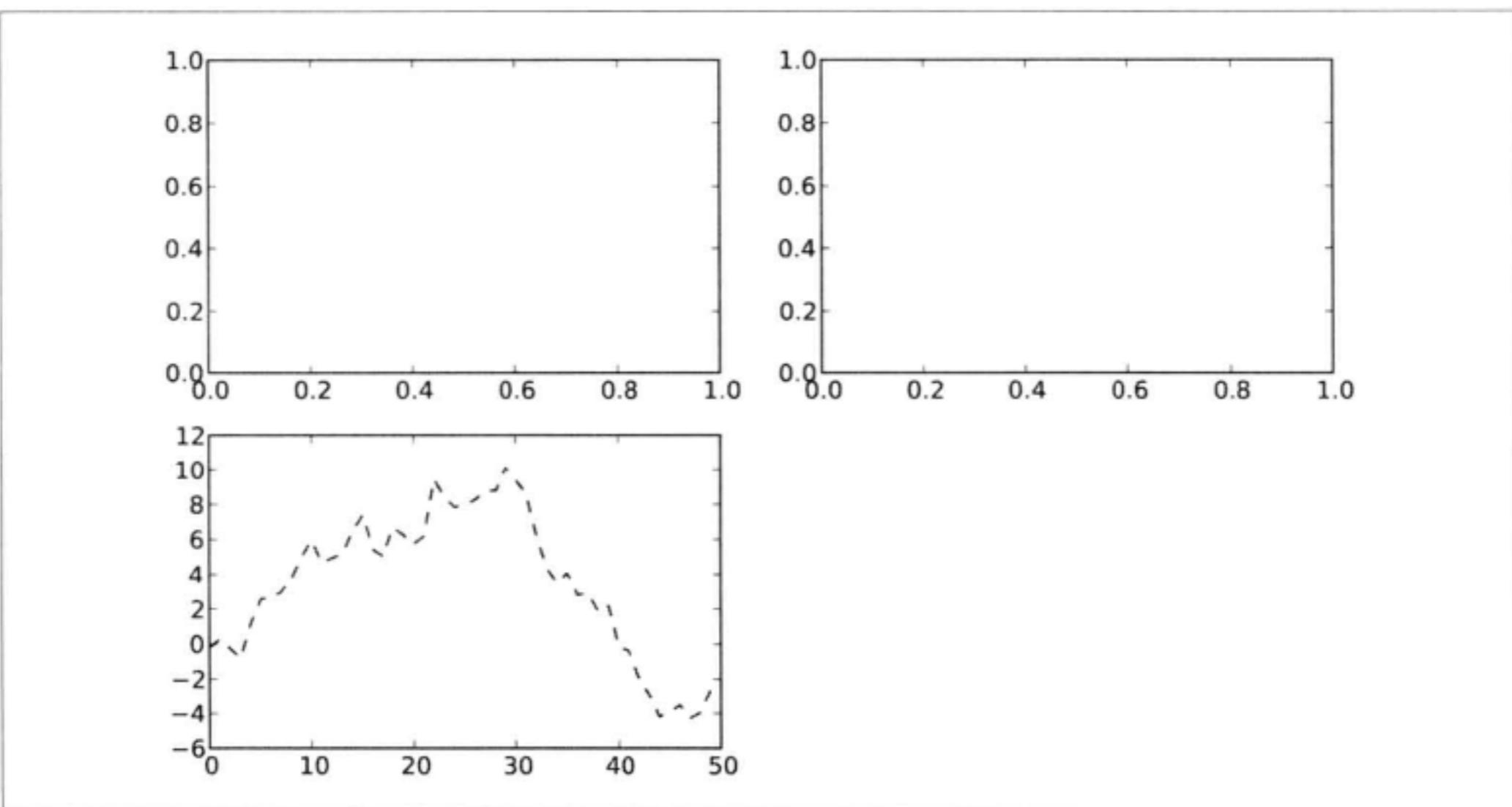


图8-3：绘制一次之后的图像

“`k--`”是一个线型选项，用于告诉matplotlib绘制黑色虚线图。上面那些由`fig.add_subplot`所返回的对象是`AxesSubplot`对象，直接调用它们的实例方法就可以在其他空着的格子里面画图了，如图8-4所示：

```
In [19]: _ = ax1.hist(randn(100), bins=20, color='k', alpha=0.3)
```

```
In [20]: ax2.scatter(np.arange(30), np.arange(30) + 3 * randn(30))
```

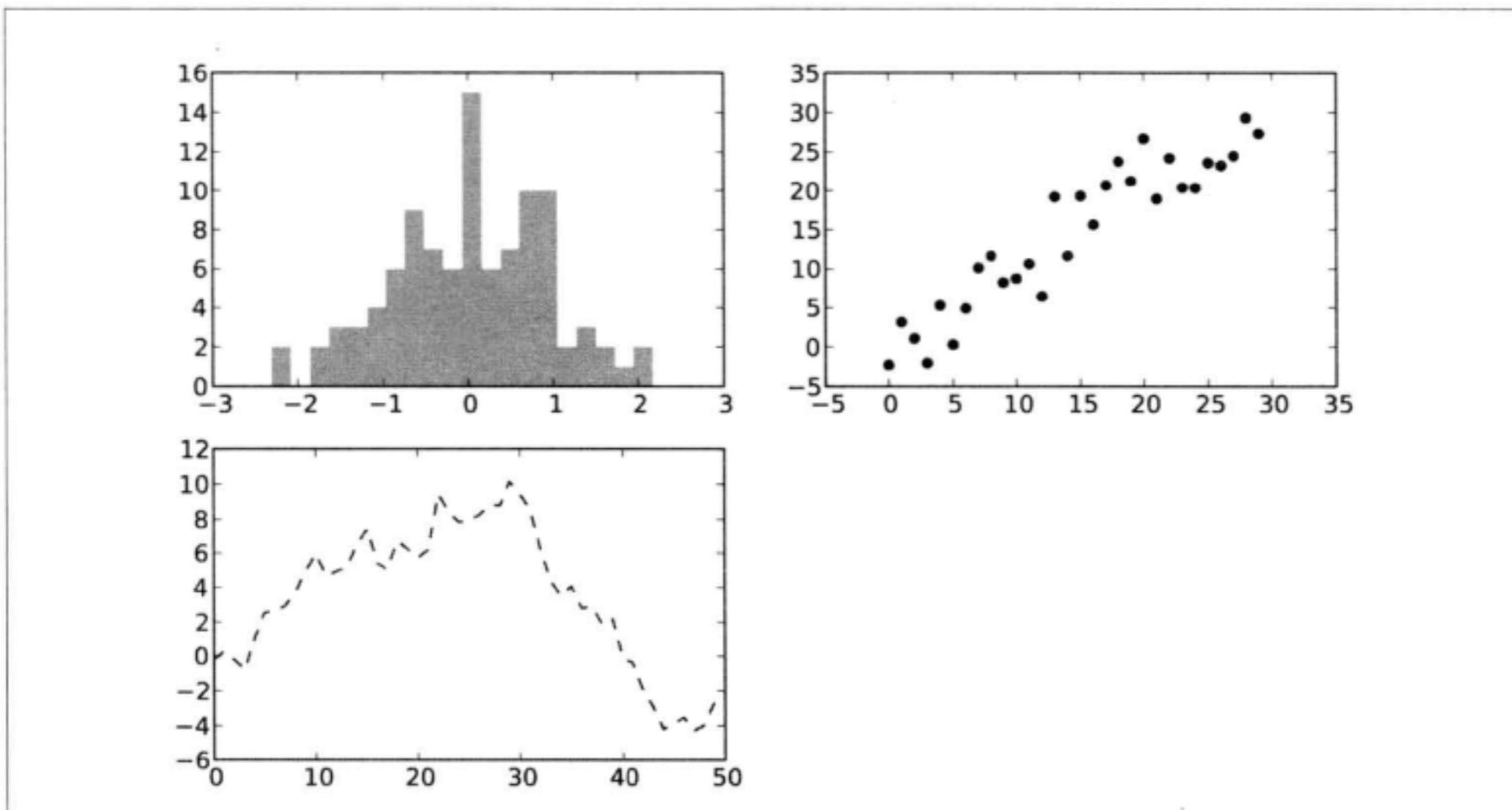


图8-4：继续绘制两次之后的图像



你可以在matplotlib的文档中找到各种图表类型。由于根据特定布局创建Figure和subplot是一件非常常见的任务，于是便出现了一个更为方便的方法（`plt.subplots`），它可以创建一个新的Figure，并返回一个含有已创建的subplot对象的NumPy数组：

```
In [22]: fig, axes = plt.subplots(2, 3)

In [23]: axes
Out[23]:
array([[Axes(0.125,0.536364;0.227941x0.363636),
       Axes (0.398529,0.536364;0.227941x0.363636),
       Axes (0.672059,0.536364;0.227941x0.363636)],
       [Axes (0.125,0.1;0.227941x0.363636),
       Axes (0.398529,0.1;0.227941x0.363636),
       Axes (0.672059,0.1;0.227941x0.363636)]], dtype=object)
```

这是非常实用的，因为可以轻松地对axes数组进行索引，就好像是一个二维数组一样，例如，`axes[0, 1]`。你还可以通过sharex和sharey指定subplot应该具有相同的X轴或Y轴。在比较相同范围的数据时，这也是非常实用的，否则，matplotlib会自动缩放各图表的界限。有关该方法的更多信息，请参见表8-1。

表8-1：pyplot.subplots的选项

参数	说明
nrows	subplot的行数
ncols	subplot的列数
sharex	所有subplot应该使用相同的X轴刻度（调节xlim将会影响所有subplot）
sharey	所有subplot应该使用相同的Y轴刻度（调节ylim将会影响所有subplot）
subplot_kw	用于创建各subplot的关键字字典
**fig_kw	创建figure时的其他关键字，如 <code>plt.subplots(2,2,figsize=(8,6))</code>

## 调整subplot周围的间距

默认情况下，matplotlib会在subplot外围留下一定的边距，并在subplot之间留下一定的间距。间距跟图像的高度和宽度有关，因此，如果你调整了图像大小（不管是编程还是手工），间距也会自动调整。利用Figure的`subplots_adjust`方法可以轻而易举地修改间距，此外，它也是个顶级函数：

```
subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None,
                 hspace=None)
```

wspace和hspace用于控制宽度和高度的百分比，可以用作subplot之间的间距。下面是一个简单的例子，其中我将间距收缩到了0（如图8-5所示）：



```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(randn(500), bins=50, color='k', alpha=0.5)
plt.subplots_adjust(wspace=0, hspace=0)
```

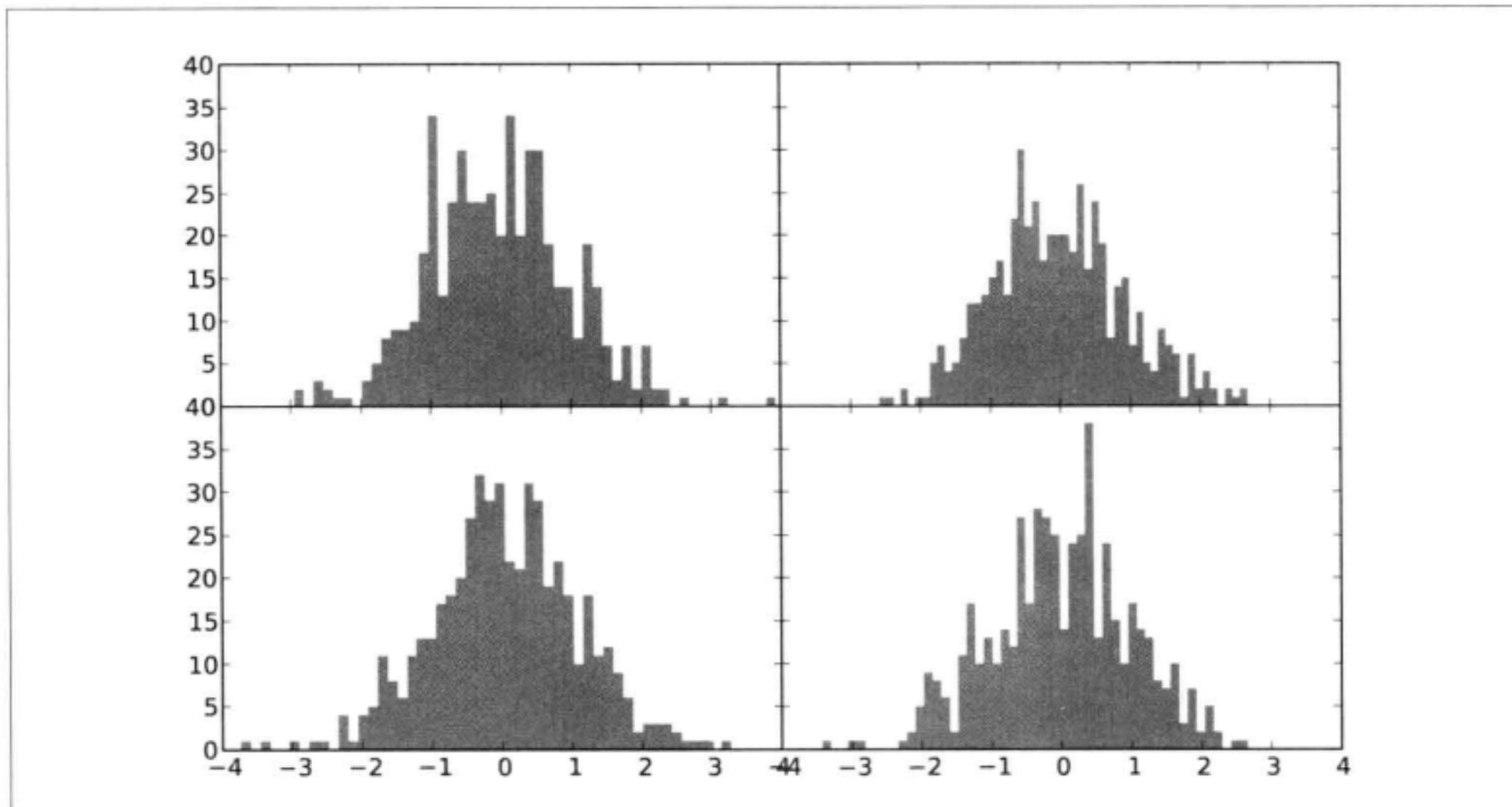


图8-5：各subplot之间没有间距

不难看出，其中的轴标签重叠了。matplotlib不会检查标签是否重叠，所以对于这种情况，你只能自己设定刻度位置和刻度标签。后面几节将会详细介绍该内容。

## 颜色、标记和线型

matplotlib的plot函数接受一组X和Y坐标，还可以接受一个表示颜色和线型的字符串缩写。例如，要根据x和y绘制绿色虚线，你可以执行如下代码：

```
ax.plot(x, y, 'g--')
```

这种在一个字符串中指定颜色和线型的方式非常方便。通过下面这种更为明确的方式也能得到同样的效果：

```
ax.plot(x, y, linestyle='--', color='g')
```

常用的颜色都有一个缩写词，要使用其他任意颜色则可以通过指定其RGB值的形式使用（例如，'#CECECE'）。完整的linestyle列表请参见plot的文档。

线型图还可以加上一些标记（marker），以强调实际的数据点。由于matplotlib创建的是



连续的线型图（点与点之间插值），因此有时可能不太容易看出真实数据点的位置。标记也可以放到格式字符串中，但标记类型和线型必须放在颜色后面（如图8-6所示）：

```
In [28]: plt.plot(randn(30).cumsum(), 'ko--')
```

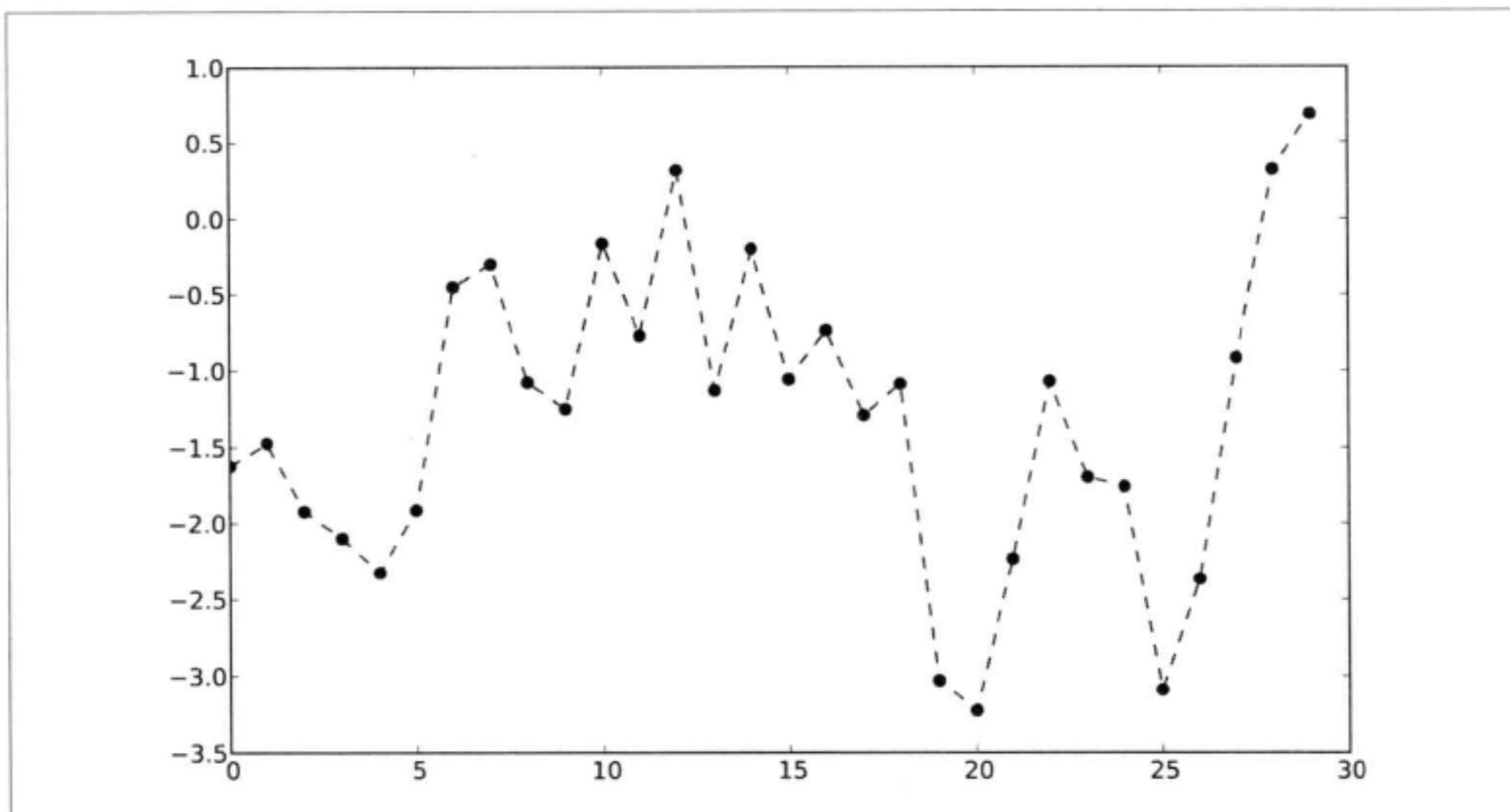


图8-6：带有标记的线型图示例

还可以将其写成更为明确的形式：

```
plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
```

在线型图中，非实际数据点默认是按线性方式插值的。可以通过drawstyle选项修改：

```
In [30]: data = randn(30).cumsum()
```

```
In [31]: plt.plot(data, 'k--', label='Default')
Out[31]: []
```

```
In [32]: plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
Out[32]: []
```

```
In [33]: plt.legend(loc='best')
```

## 刻度、标签和图例

对于大多数的图表装饰项，其主要实现方式有二：使用过程型的pyplot接口（MATLAB用户非常熟悉）以及更为面向对象的原生matplotlib API。



`pyplot`接口的设计目的就是交互式使用，含有诸如`xlim`、`xticks`和`xticklabels`之类的方法。它们分别控制图表的范围、刻度位置、刻度标签等。其使用方式有以下两种：

- 调用时不带参数，则返回当前的参数值<sup>译注1</sup>。例如，`plt.xlim()`返回当前的X轴绘图范围。
- 调用时带参数，则设置参数值。因此，`plt.xlim([0, 10])`会将X轴的范围设置为0到10。

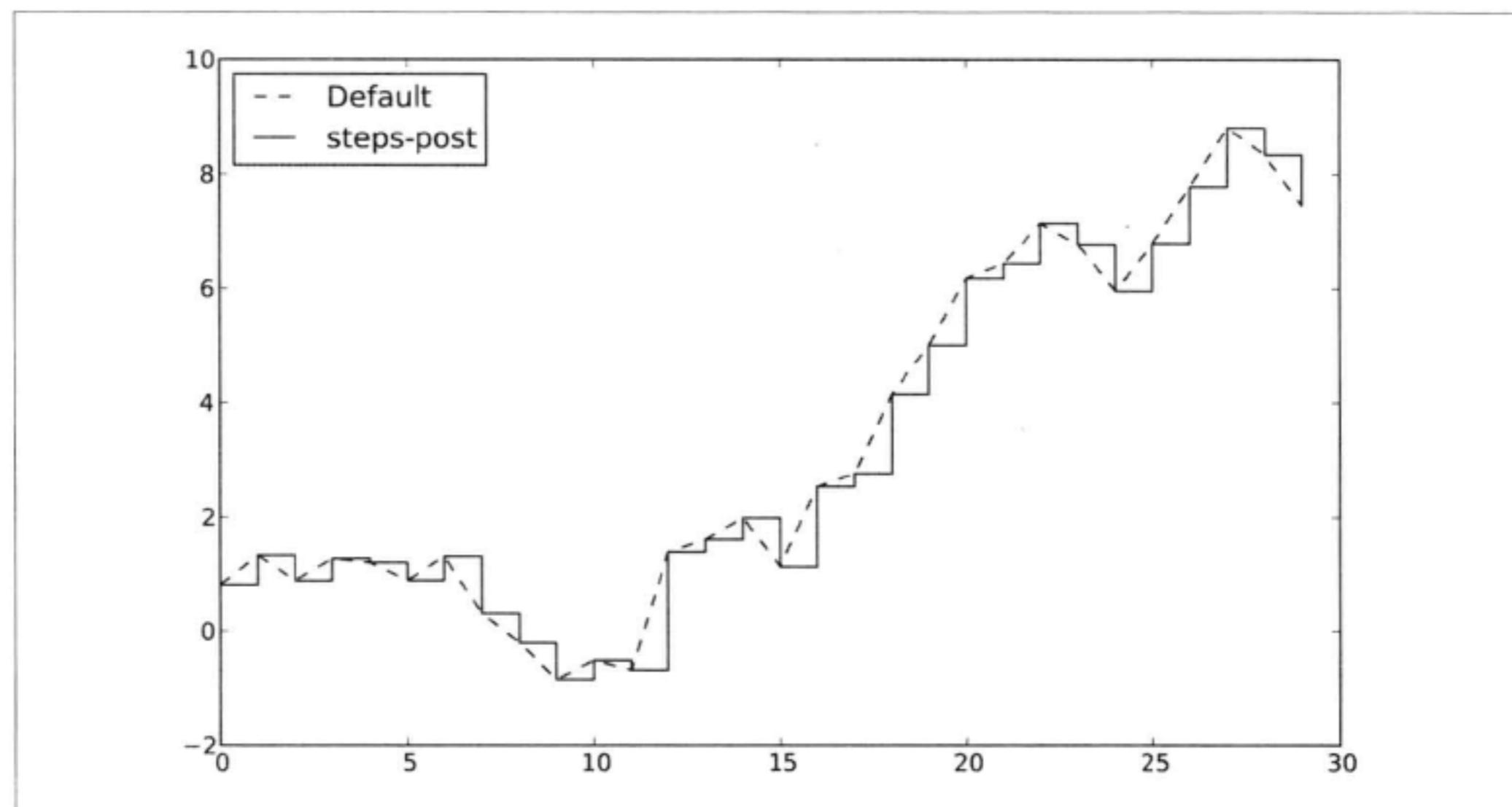


图8-7：不同drawstyle选项的线型图

所有这些方法都是对当前或最近创建的AxesSubplot起作用的。它们各自对应subplot对象上的两个方法，以`xlim`为例，就是`ax.get_xlim`和`ax.set_xlim`。我更喜欢使用subplot的实例方法（因为我喜欢明确的事情，而且在处理多个subplot时这样也更清楚一些）。当然你完全可以选择自己觉得方便的那个。

## 设置标题、轴标签、刻度以及刻度标签

为了说明轴的自定义，我将创建一个简单的图像并绘制一段随机漫步（如图8-8所示）：

```
In [34]: fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
```

```
In [35]: ax.plot(randn(1000).cumsum())
```

译注1：前面的参数是argument，后面的参数是parameter。我觉得后面那个parameter不太合适，但又实在想不出更好的表达方式。各位读者可以把后面那个parameter理解为“当前配置值”。下面那条也是如此。



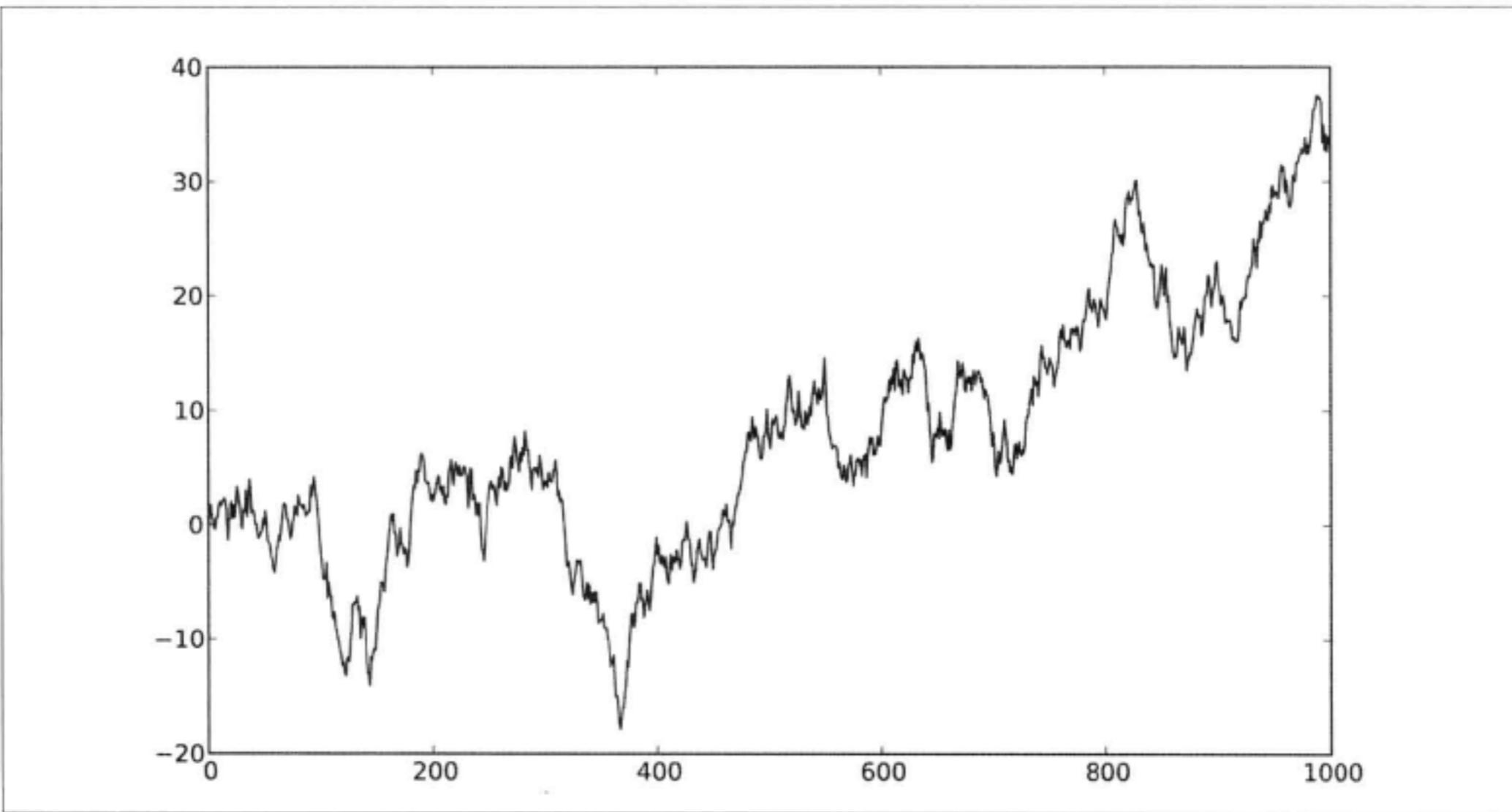


图8-8：用于演示xticks的简单线型图

要修改X轴的刻度，最简单的办法是使用`set_xticks`和`set_xticklabels`。前者告诉matplotlib要将刻度放在数据范围中的哪些位置，默认情况下，这些位置也就是刻度标签。但我们可以`set_xticklabels`将任何其他的值用作标签：

```
In [36]: ticks = ax.set_xticks([0, 250, 500, 750, 1000])
```

```
In [37]: labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],
...:                                     rotation=30, fontsize='small')
```

最后，再用`set_xlabel`为X轴设置一个名称，并用`set_title`设置一个标题：

```
In [38]: ax.set_title('My first matplotlib plot')
Out[38]: <matplotlib.text.Text at 0x7f9190912850>
```

```
In [39]: ax.set_xlabel('Stages')
```

最终结果如图8-9所示。Y轴的修改方式与此类似，只需将上述代码中的`x`替换为`y`即可。

## 添加图例

图例（legend）是另一种用于标识图表元素的重要工具。添加图例的方式有二。最简单的是在添加`subplot`的时候传入`label`参数：

```
In [40]: fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
```

```
In [41]: ax.plot(randn(1000).cumsum(), 'k', label='one')
Out[41]: [<matplotlib.lines.Line2D at 0x4720a90>]
```



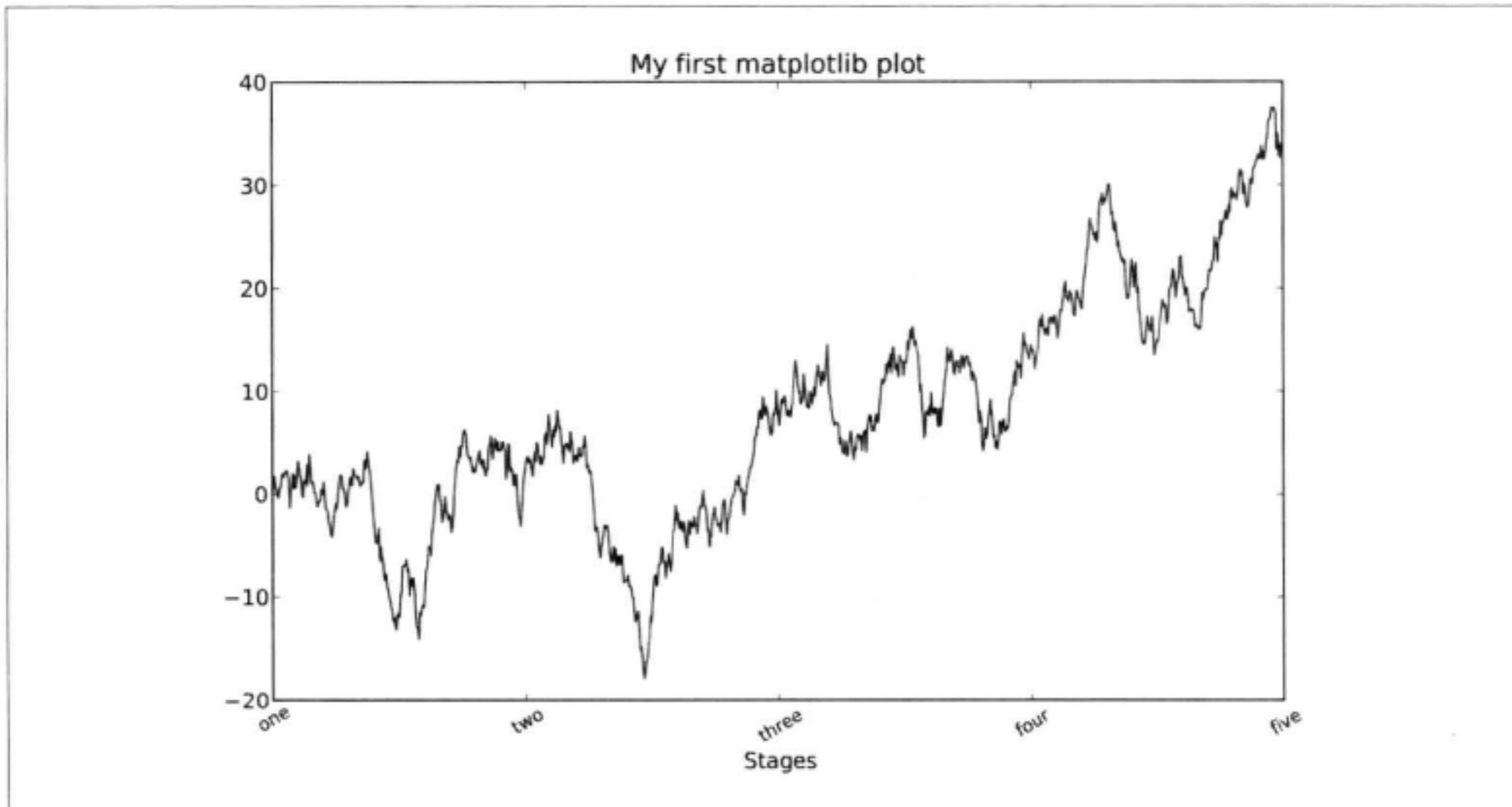


图8-9：用于演示xticks的简单线型图

```
In [42]: ax.plot(randn(1000).cumsum(), 'k--', label='two')
Out[42]: [
```

```
In [43]: ax.plot(randn(1000).cumsum(), 'k.', label='three')
Out[43]: [
```

在此之后，你可以调用`ax.legend()`或`plt.legend()`来自动创建图例：

```
In [44]: ax.legend(loc='best')
```

如图8-10所示。`loc`告诉`matplotlib`要将图例放在哪。如果你不是吹毛求疵的话，“`best`”是不错的选择，因为它会选择最不碍事的位置。要从图例中去除一个或多个元素，不传入`label`或传入`label='_nolegend_'`即可。

## 注解以及在Subplot上绘图

除标准的图表对象之外，你可能还希望绘制一些自定义的注解（比如文本、箭头或其他图形等）。

注解可以通过`text`、`arrow`和`annotate`等函数进行添加。`text`可以将文本绘制在图表的指定坐标( $x$ ,  $y$ )，还可以加上一些自定义格式：

```
ax.text(x, y, 'Hello world!',
        family='monospace', fontsize=10)
```



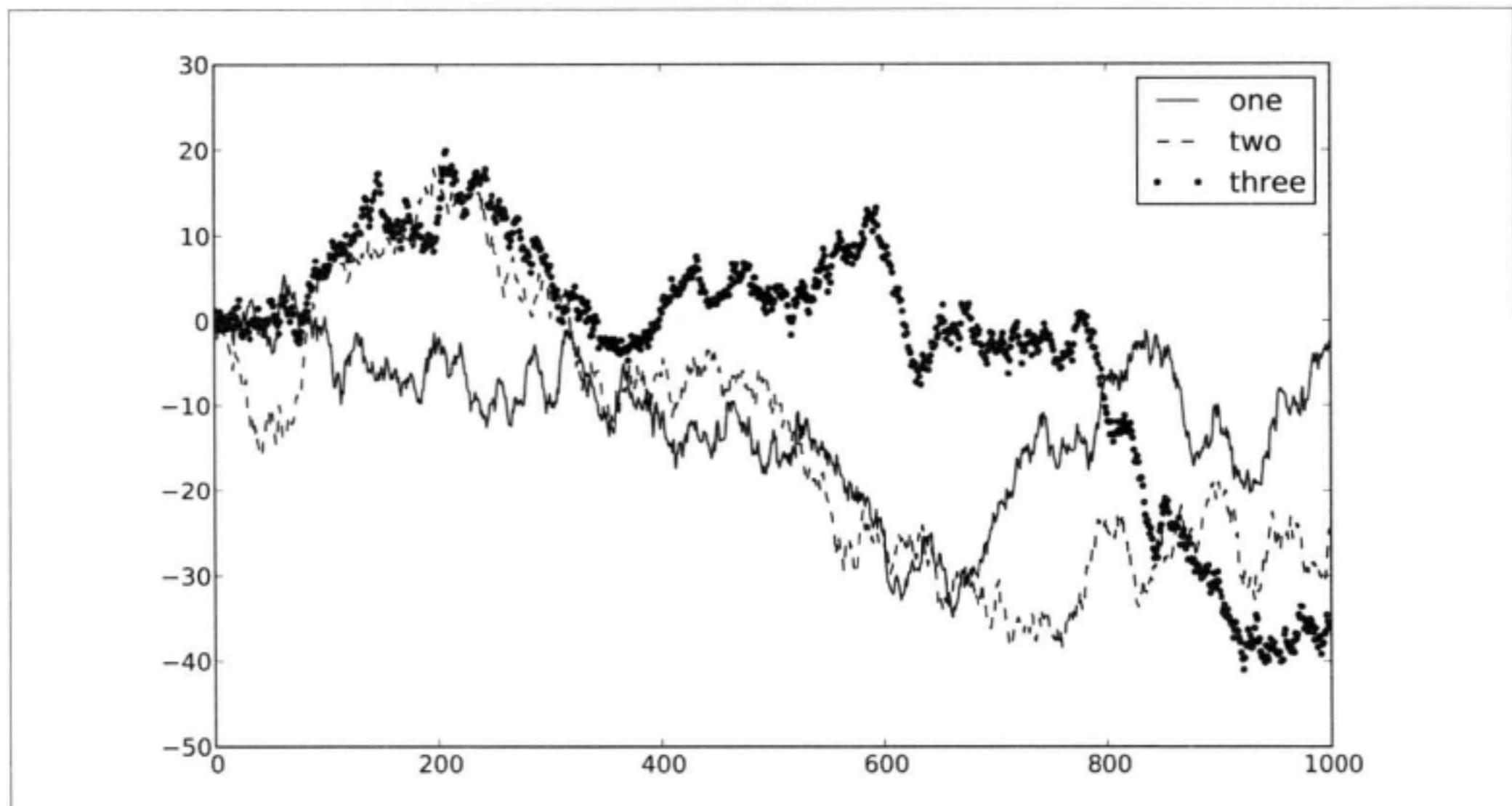


图8-10：带有三条线以及图例的简单线型图

注解中可以既含有文本也含有箭头。例如，我们根据2007年以来的标准普尔500指数收盘价格（来自Yahoo! Finance）绘制一张曲线图，并标出2008年到2009年金融危机期间的一些重要日期。结果如图8-11所示：

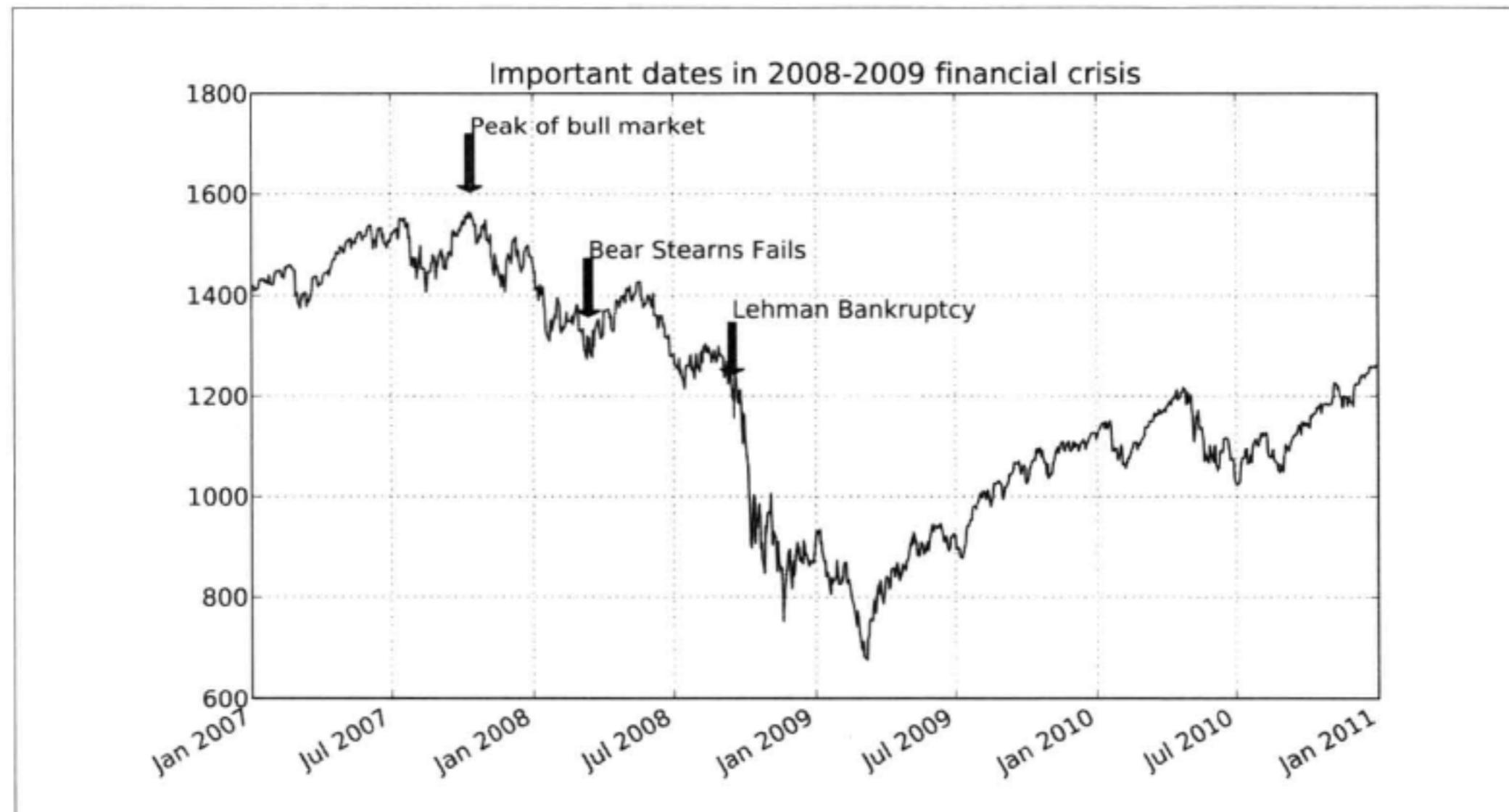


图8-11：2008—2009年金融危机期间的重要日期

```
from datetime import datetime
```



```

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

data = pd.read_csv('ch08/spx.csv', index_col=0, parse_dates=True)
spx = data['SPX']

spx.plot(ax=ax, style='k-')

crisis_data = [
    (datetime(2007, 10, 11), 'Peak of bull market'),
    (datetime(2008, 3, 12), 'Bear Stearns Fails'),
    (datetime(2008, 9, 15), 'Lehman Bankruptcy')
]

for date, label in crisis_data:
    ax.annotate(label, xy=(date, spx.asof(date) + 50),
                xytext=(date, spx.asof(date) + 200),
                arrowprops=dict(facecolor='black'),
                horizontalalignment='left', verticalalignment='top')

# 放大到2007-2010
ax.set_xlim(['1/1/2007', '1/1/2011'])
ax.set_ylim([600, 1800])

ax.set_title('Important dates in 2008-2009 financial crisis')

```

更多有关注解的示例，请访问matplotlib的在线示例库。

图形的绘制要麻烦一些。matplotlib有一些表示常见图形的对象。这些对象被称为块(patch)。其中有些可以在matplotlib.pyplot中找到（如Rectangle和Circle），但完整集合位于matplotlib.patches。

要在图表中添加一个图形，你需要创建一个块对象shp，然后通过ax.add\_patch(shp)将其添加到subplot中（如图8-12所示）：

```

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

rect = plt.Rectangle((0.2, 0.75), 0.4, 0.15, color='k', alpha=0.3)
circ = plt.Circle((0.7, 0.2), 0.15, color='b', alpha=0.3)
pgon = plt.Polygon([[0.15, 0.15], [0.35, 0.4], [0.2, 0.6]], color='g', alpha=0.5)

ax.add_patch(rect)
ax.add_patch(circ)
ax.add_patch(pgon)

```

如果查看许多常见图表对象的具体实现代码，你就会发现它们其实就是由块组装而成的。



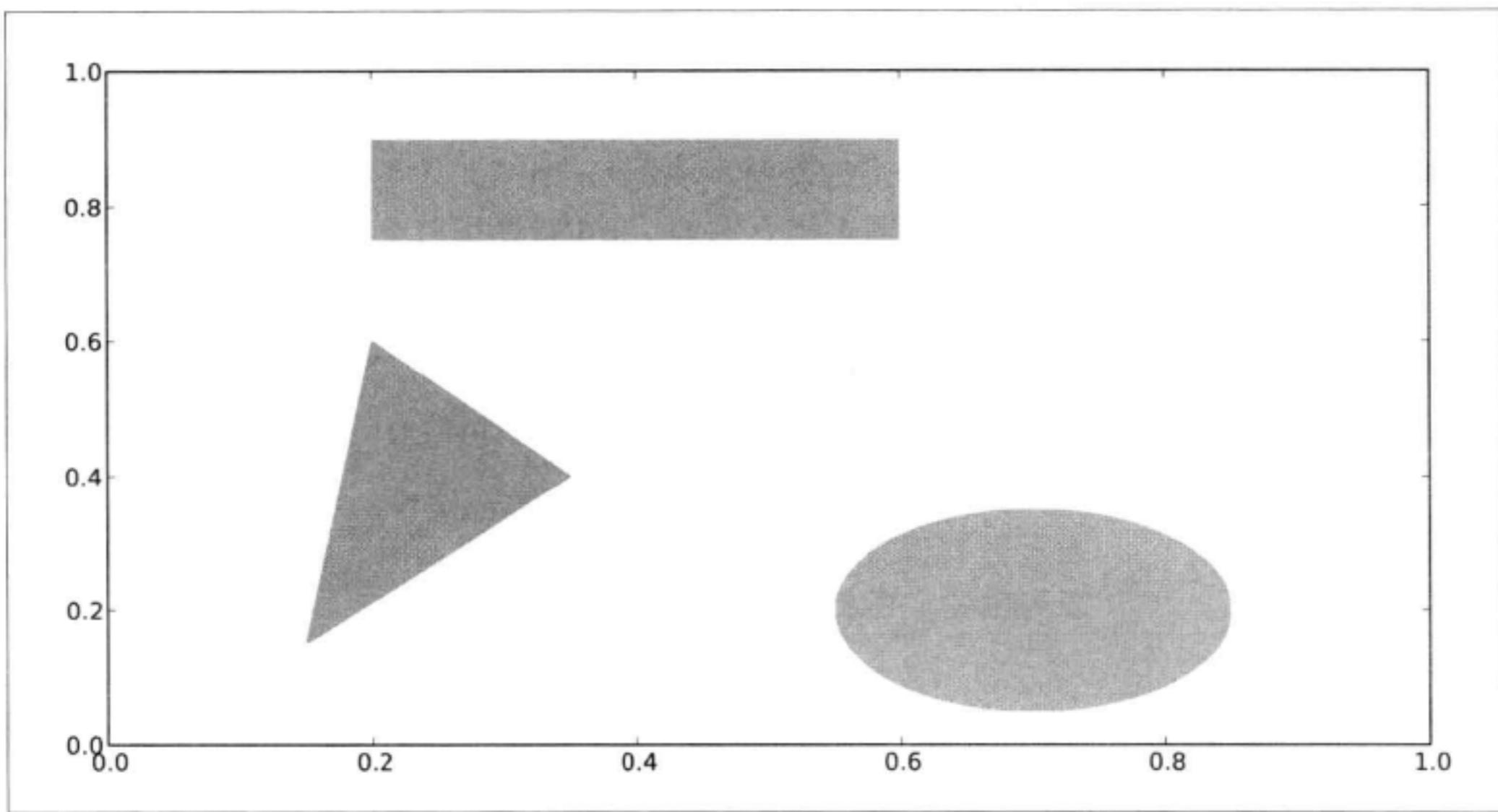


图8-12：由三个块图形组成的图

## 将图表保存到文件

利用`plt.savefig`可以将当前图表保存到文件。该方法相当于Figure对象的实例方法`savefig`。例如，要将图表保存为SVG文件，你只需输入：

```
plt.savefig('figpath.svg')
```

文件类型是通过文件扩展名推断出来的。因此，如果你使用的是`.pdf`，就会得到一个PDF文件。我在发布图片时最常用到两个重要的选项是`dpi`（控制“每英寸点数”分辨率）和`bbox_inches`（可以剪除当前图表周围的空白部分）。要得到一张带有最小白边且分辨率为400DPI的PNG图片，你可以：

```
plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
```

`savefig`并非一定要写入磁盘，也可以写入任何文件型的对象，比如`StringIO`：

```
from io import StringIO  
buffer = StringIO()  
plt.savefig(buffer)  
plot_data = buffer.getvalue()
```

这对在Web上提供动态生成的图片是很实用的。

`Figure.savefig`方法的参数及说明如表8-2所示。



表8-2: Figure.savefig的选项

参数	说明
fname	含有文件路径的字符串或Python的文件型对象。图像格式由文件扩展名推断得出，例如，.pdf推断出PDF，.png推断出PNG
dpi	图像分辨率（每英寸点数），默认为100
facecolor、edgecolor	图像的背景色，默认为“w”（白色）
format	显式设置文件格式（“png”、“pdf”、“svg”、“ps”、“eps”……）
bbox_inches	图表需要保存的部分。如果设置为“tight”，则将尝试剪除图表周围的空白部分

## matplotlib配置

matplotlib自带一些配色方案，以及为生成出版质量的图片而设定的默认配置信息。幸运的是，几乎所有默认行为都能通过一组全局参数进行自定义，它们可以管理图像大小、subplot边距、配色方案、字体大小、网格类型等。操作matplotlib配置系统的方式主要有两种。第一种是Python编程方式，即利用rc方法。比如说，要将全局的图像默认大小设置为 $10 \times 10$ ，你可以执行：

```
plt.rc('figure', figsize=(10, 10))
```

rc的第一个参数是希望自定义的对象，如'figure'、'axes'、'xtick'、'ytick'、'grid'、'legend'等。其后可以跟上一系列的关键字参数。最简单的办法是将这些选项写成一个字典：

```
font_options = {'family': 'monospace',
                'weight': 'bold',
                'size': 'small'}
plt.rc('font', **font_options)
```

要了解全部的自定义选项，请查阅matplotlib的配置文件matplotlibrc（位于matplotlib/mpl-data目录中）。如果对该文件进行了自定义，并将其放在你自己的.matplotlibrc目录<sup>译注2</sup>中，则每次使用matplotlib时就会加载该文件。

## pandas中的绘图函数

不难看出，matplotlib实际上是一种比较低级的工具。要组装一张图表，你得用它的各种基础组件才行：数据展示（即图表类型：线型图、柱状图、盒形图、散布图、等值线图等）、图例、标题、刻度标签以及其他注解型信息。这是因为要根据数据制作一张完整

译注2：正确的目录名是.matplotlib。



图表通常都需要用到多个对象。在pandas中，我们有行标签、列标签以及分组信息（可能有）。这也就是说，要制作一张完整的图表，原本需要一大堆的matplotlib代码，现在只需一两条简洁的语句就可以了。pandas有许多能够利用DataFrame对象数据组织特点来创建标准图表的高级绘图方法（这些函数的数量还在不断增加）。

---

**警告：**到目前为止，pandas团队已经在绘图功能上下了很大工夫。一个参加了“2012 Google Summer of Code计划”的学生正在夜以继日地添加新功能，并使该接口具有更好的一致性和可用性。因此，本书中的这部分代码可能很快就要过时了。如果那样的话，pandas在线文档将会是最好的学习资源。

---

## 线型图

Series和DataFrame都有一个用于生成各类图表的plot方法。默认情况下，它们所生成的是线型图（如图8-13所示）：

```
In [55]: s = Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))  
In [56]: s.plot()
```

该Series对象的索引会被传给matplotlib，并用以绘制X轴。可以通过use\_index=False禁用该功能。X轴的刻度和界限可以通过xticks和xlim选项进行调节，Y轴就用yticks和ylim。plot参数的完整列表请参见表8-3。我只会讲解其中几个，剩下的就留给读者自己去研究了。

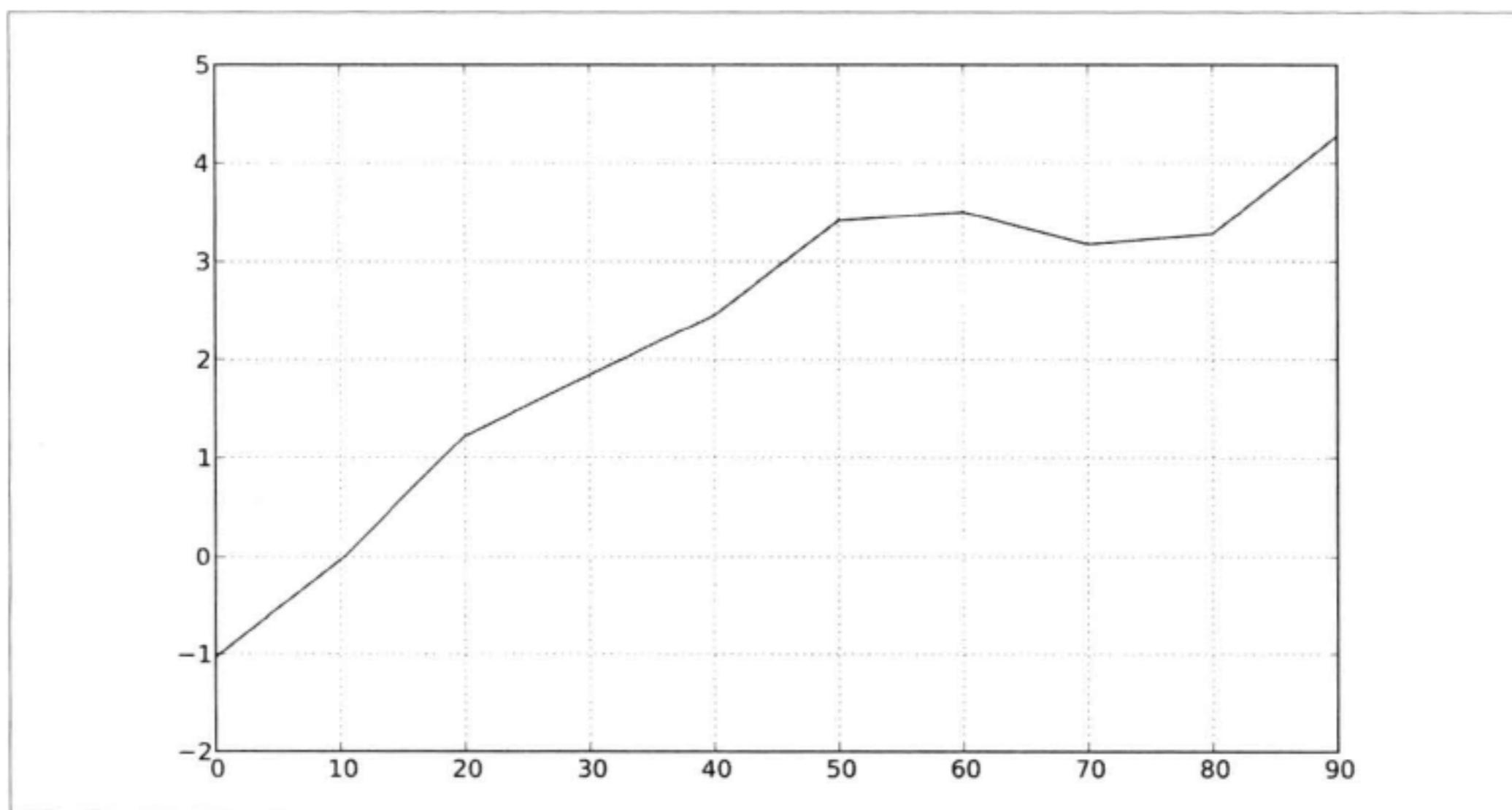


图8-13：简单的Series图表示例



pandas的大部分绘图方法都有一个可选的`ax`参数，它可以是一个matplotlib的subplot对象。这使你能够在网格布局中更为灵活地处理subplot的位置。

DataFrame的`plot`方法会在一个subplot中为各列绘制一条线，并自动创建图例（如图8-14所示）：

```
In [57]: df = DataFrame(np.random.randn(10, 4).cumsum(0),
...:                      columns=['A', 'B', 'C', 'D'],
...:                      index=np.arange(0, 100, 10))
In [58]: df.plot()
```

注意：`plot`的其他关键字参数会被传给相应的matplotlib绘图函数，所以要更深入地自定义图表，就必须学习更多有关matplotlib API的知识。

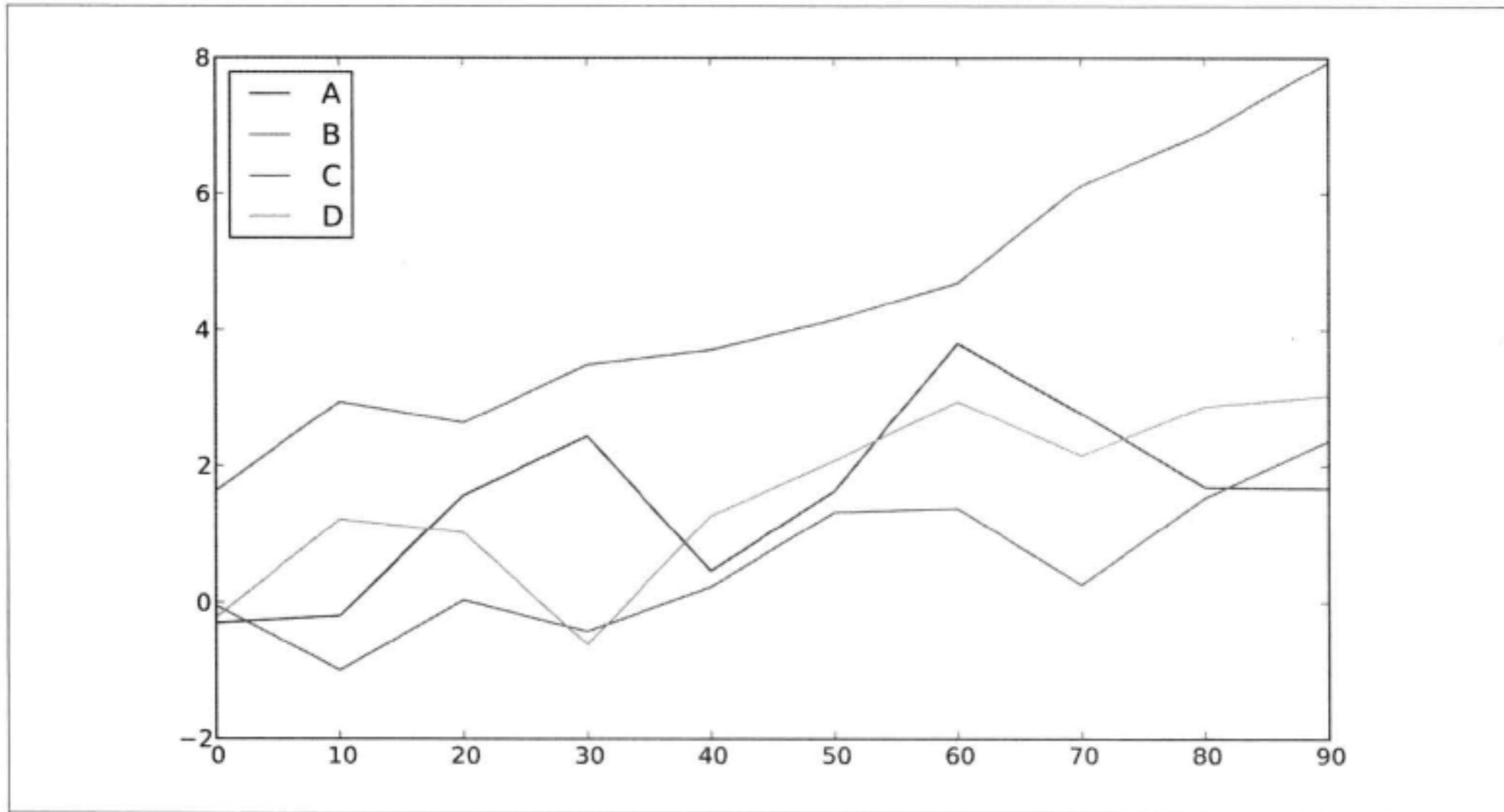


图8-14：简单的DataFrame图表示例

表8-3：Series.plot方法的参数

参数	说明
<code>label</code>	用于图例的标签
<code>ax</code>	要在其上进行绘制的matplotlib subplot对象。如果没有设置，则使用当前matplotlib subplot
<code>style</code>	将要传给matplotlib的风格字符串（如 <code>'ko--'</code> ）
<code>alpha</code>	图表的填充不透明度（0到1之间）



表8-3: Series.plot方法的参数 (续)

参数	说明
kind	可以是'line'、'bar'、'barh'、'kde'
logy	在Y轴上使用对数标尺
use_index	将对象的索引用作刻度标签
rot	旋转刻度标签 (0到360)
xticks	用作X轴刻度的值
yticks	用作Y轴刻度的值
xlim	X轴的界限 (例如[0, 10])
ylim	Y轴的界限
grid	显示轴网格线 (默认打开)

DataFrame还有一些用于对列进行灵活处理的选项，例如，是要将所有列都绘制到一个 subplot中还是创建各自的subplot。详细信息请参见表8-4。

表8-4: 专用于DataFrame的plot的参数

参数	说明
subplots	将各个DataFrame列绘制到单独的subplot中
sharex	如果subplots=True，则共用同一个X轴，包括刻度和界限
sharey	如果subplots=True，则共用同一个Y轴
figsize	表示图像大小的元组
title	表示图像标题的字符串
legend	添加一个subplot图例 (默认为True)
sort_columns	以字母表顺序绘制各列，默认使用当前列顺序

注意：有关时间序列的绘制技术，请参见第10章。

## 柱状图

在生成线型图的代码中加上kind='bar' (垂直柱状图) 或kind='barh' (水平柱状图) 即可生成柱状图。这时，Series和DataFrame的索引将会被用作X (bar) 或Y (barh) 刻度 (如图8-15所示)：

```
In [59]: fig, axes = plt.subplots(2, 1)
```

```
In [60]: data = Series(np.random.rand(16), index=list('abcdefghijklmnp'))
```



```
In [61]: data.plot(kind='bar', ax=axes[0], color='k', alpha=0.7)
```

```
Out[61]: <matplotlib.axes.AxesSubplot at 0x4ee7750>
```

```
In [62]: data.plot(kind='barh', ax=axes[1], color='k', alpha=0.7)
```

注意：更多有关plt.subplots函数以及matplotlib轴和图像的信息，请参见本章后续的内容。

对于DataFrame，柱状图会将每一行的值分为一组，如图8-16所示：

```
In [63]: df = DataFrame(np.random.rand(6, 4),  
...:                     index=['one', 'two', 'three', 'four', 'five', 'six'],  
...:                     columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
```

```
In [64]: df
```

```
Out[64]:
```

Genus	A	B	C	D
one	0.301686	0.156333	0.371943	0.270731
two	0.750589	0.525587	0.689429	0.358974
three	0.381504	0.667707	0.473772	0.632528
four	0.942408	0.180186	0.708284	0.641783
five	0.840278	0.909589	0.010041	0.653207
six	0.062854	0.589813	0.811318	0.060217

```
In [65]: df.plot(kind='bar')
```

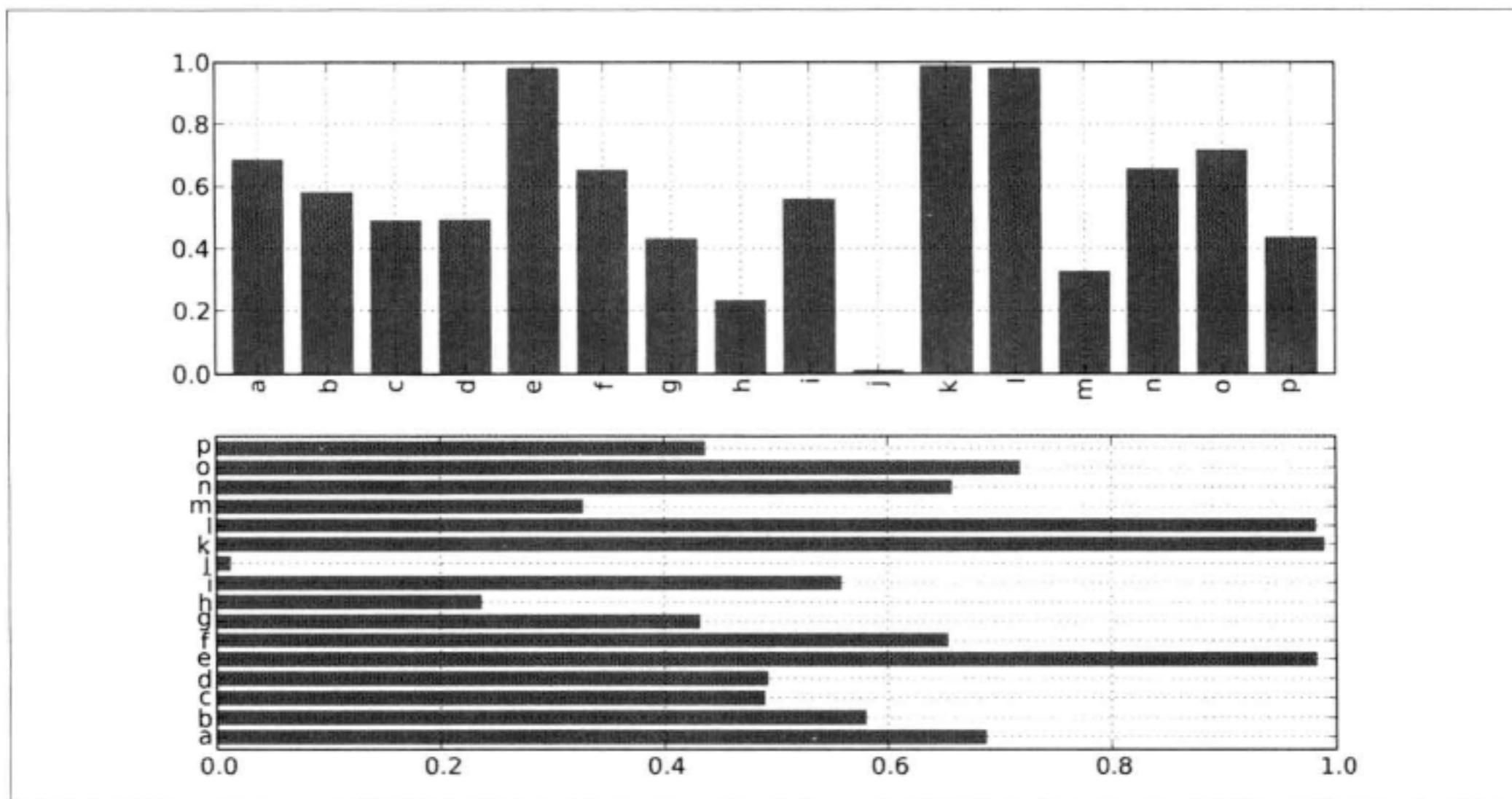


图8-15：水平和垂直柱状图示例

注意，DataFrame各列的名称“Genus”被用作了图例的标题。设置stacked=True即可为DataFrame生成堆积柱状图，这样每行的值就会被堆积在一起（如图8-17所示）：

```
In [67]: df.plot(kind='barh', stacked=True, alpha=0.5)
```



注意：柱状图有一个非常不错的用法：利用value\_counts图形化显示Series中各值的出现频率，比如`s.value_counts().plot(kind='bar')`。

再以本书前面用过的那个有关小费的数据集为例<sup>译注3</sup>，假设我们想要做一张堆积柱状图以展示每天各种聚会规模的数据点的百分比。我用read\_csv将数据加载进来，然后根据日期和聚会规模创建一张交叉表：

```
In [68]: tips = pd.read_csv('ch08/tips.csv')

In [69]: party_counts = pd.crosstab(tips.day, tips.size)

In [70]: party_counts
Out[70]:
size   1    2    3    4    5    6
day
Fri     1   16    1    1    0    0
Sat     2   53   18   13    1    0
Sun     0   39   15   18    3    1
Thur    1   48    4    5    1    3

# 1个人和6个人的聚会都比较少
In [71]: party_counts = party_counts.ix[:, 2:5]
```

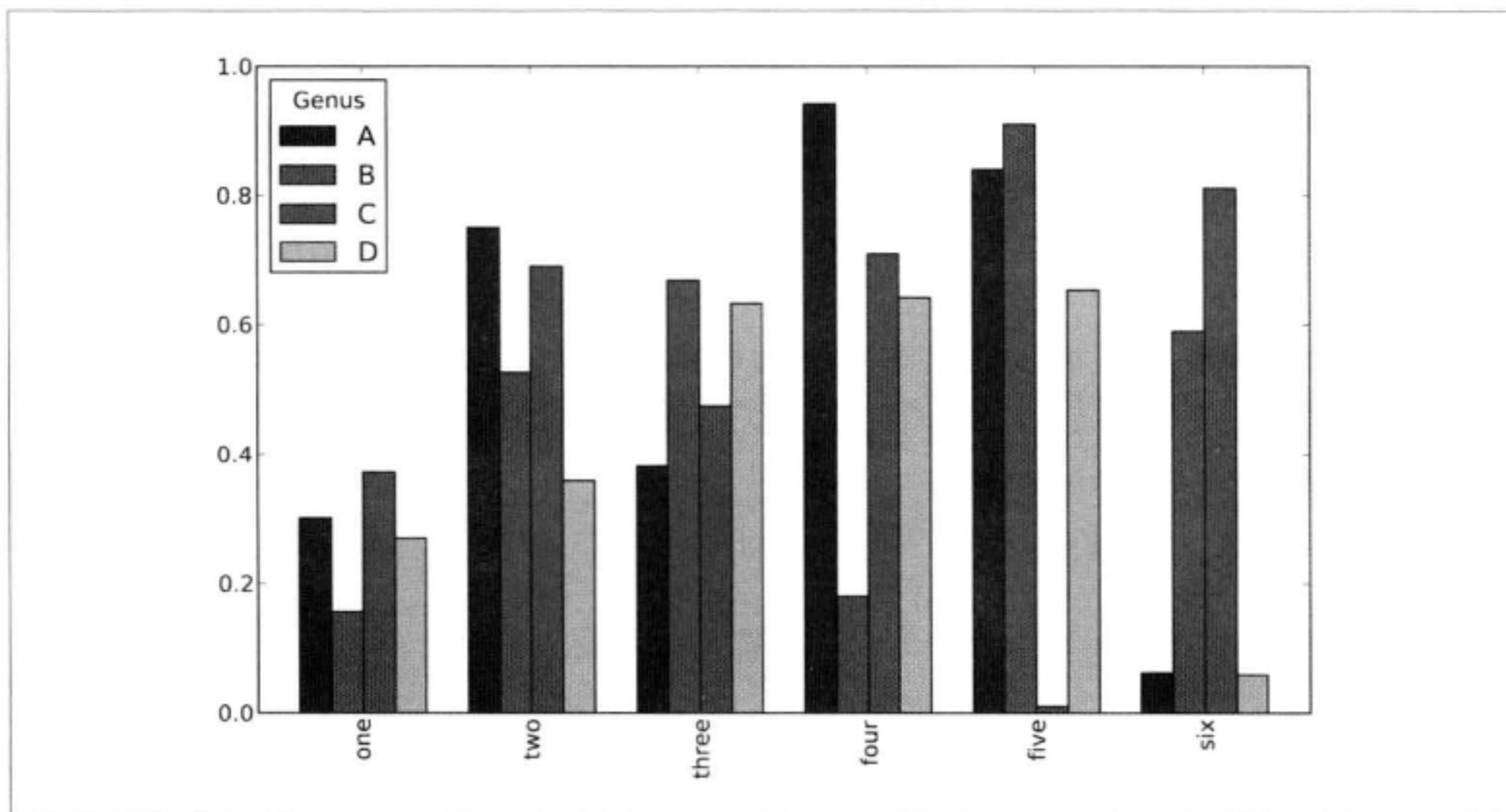


图8-16：DataFrame柱状图示例

译注3：本书前面没有用过这个数据集，读者不用找了。



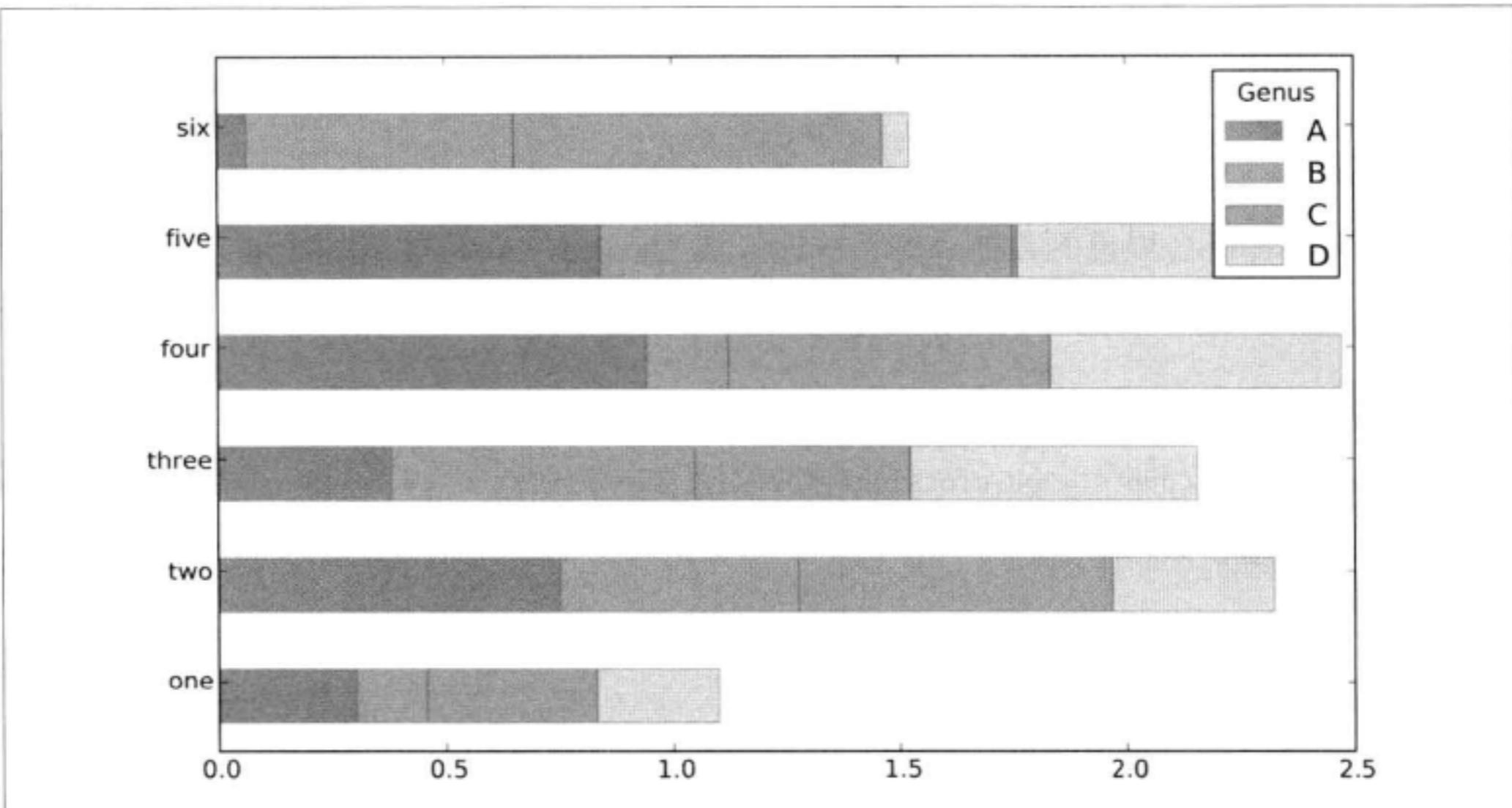


图8-17：DataFrame堆积柱状图示例

然后进行规格化，使得各行的和为1（必须转换成浮点数，以避免Python 2.7中的整数除法问题），并生成图表（如图8-18所示）：

```
# 规格化成“和为1”
In [72]: party_pcts = party_counts.div(party_counts.sum(1).astype(float), axis=0)

In [73]: party_pcts
Out[73]:
size      2          3          4          5
day
Fri    0.888889  0.055556  0.055556  0.000000
Sat    0.623529  0.211765  0.152941  0.011765
Sun    0.520000  0.200000  0.240000  0.040000
Thur   0.827586  0.068966  0.086207  0.017241

In [74]: party_pcts.plot(kind='bar', stacked=True)
```

于是，通过该数据集就可以看出，聚会规模在周末会变大。

## 直方图和密度图

直方图（histogram）是一种可以对值频率进行离散化显示的柱状图。数据点被拆分到离散的、间隔均匀的面元中，绘制的是各面元中数据点的数量。再以前面那个小费数据为例，通过Series的hist方法，我们可以生成一张“小费占消费总额百分比”<sup>译注4</sup>的直方图（如图8-19所示）：

<sup>译注4</sup>：仔细观察数据可以发现，实际并不是这样的，因为这里的小费可能不在消费总额里面。仅当做一个例子即可，不必深究。



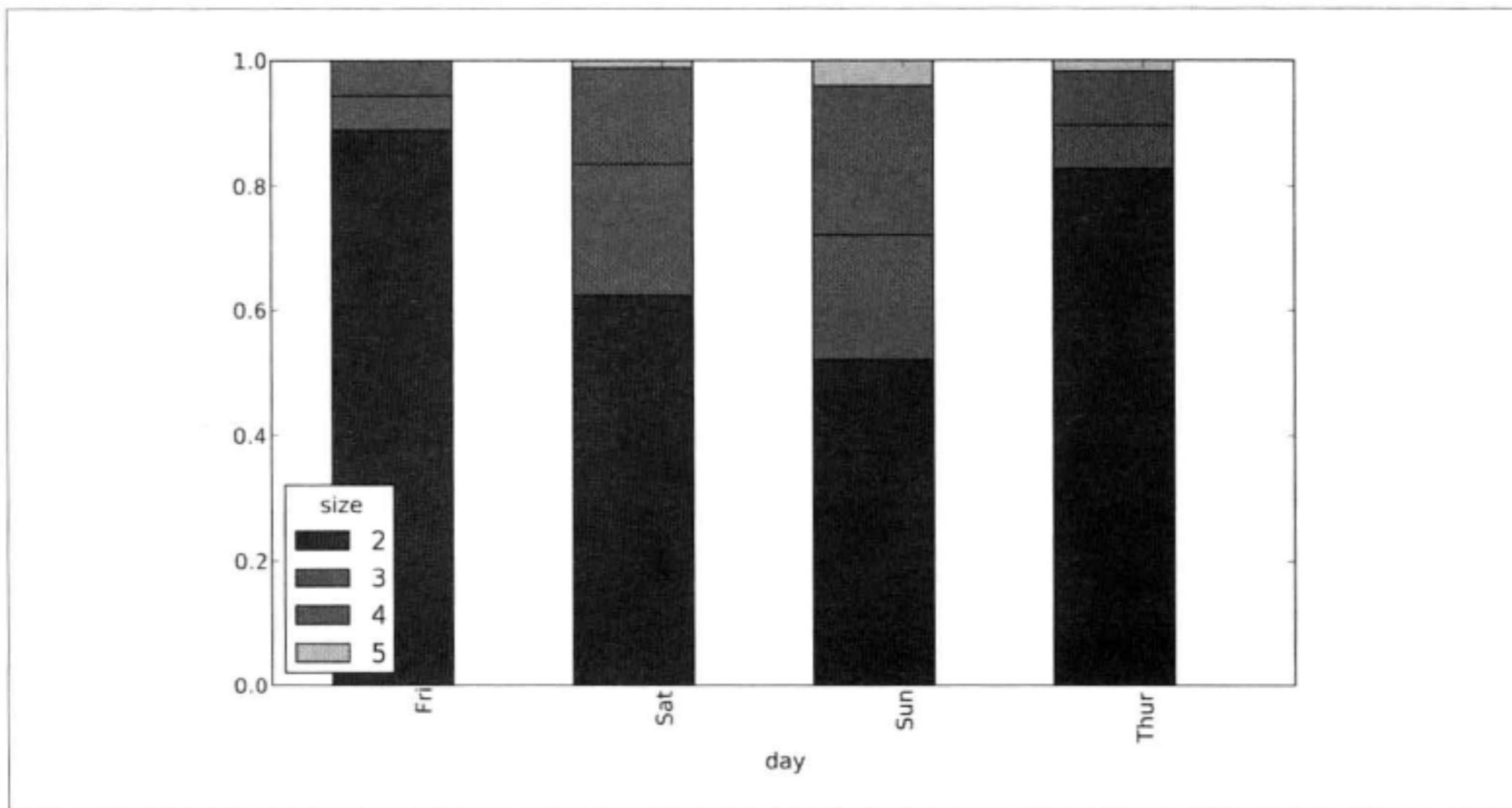


图8-18：每天各种聚会规模的比例

```
In [76]: tips['tip_pct'] = tips['tip'] / tips['total_bill']  
In [77]: tips['tip_pct'].hist(bins=50)
```

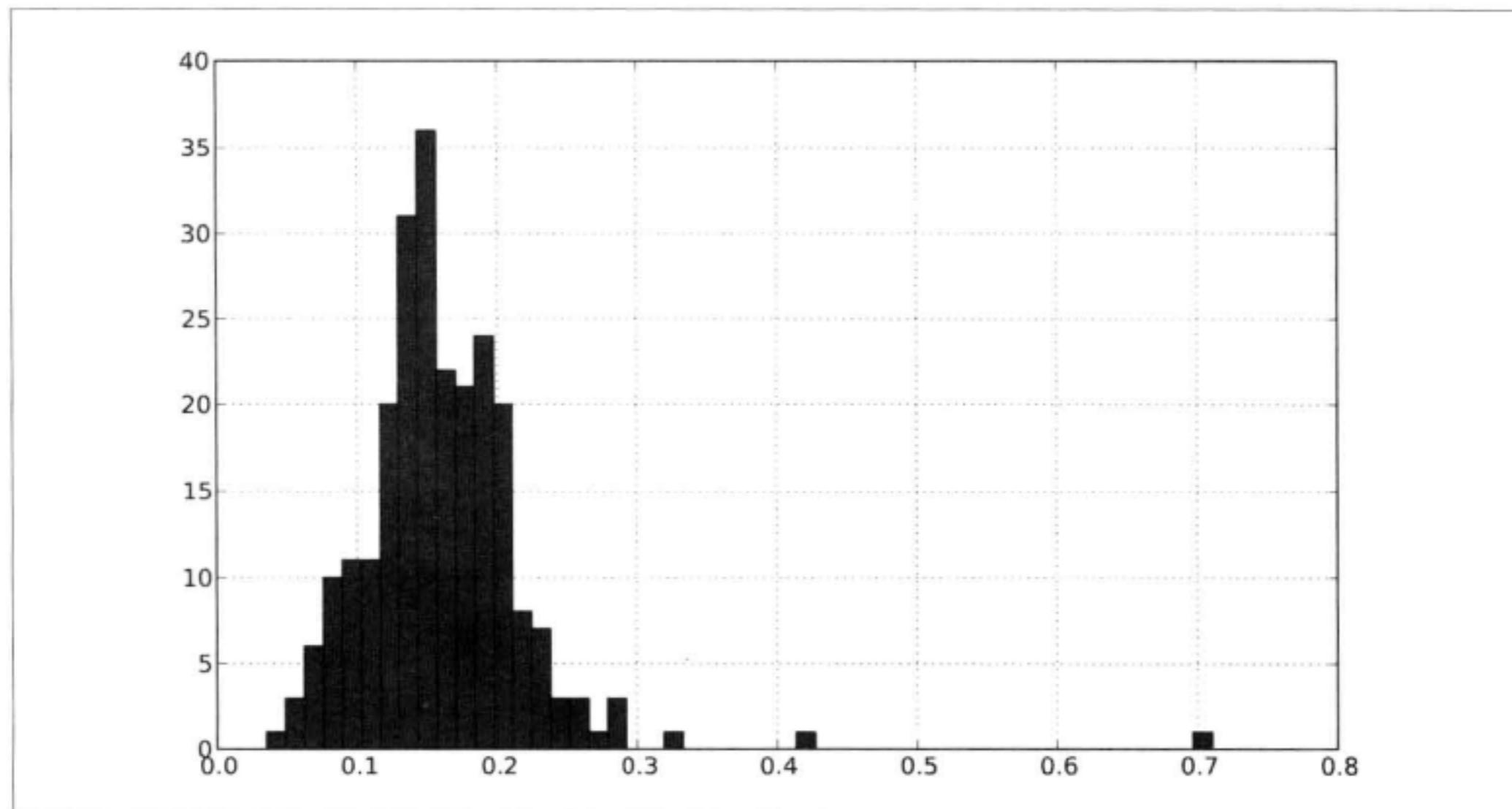


图8-19：小费百分比的直方图

与此相关的一种图表类型是密度图，它是通过计算“可能会产生观测数据的连续概率分布的估计”而产生的。一般的过程是将该分布近似为一组核（即诸如正态（高斯）分



布之类的较为简单的分布）。因此，密度图也被称作KDE（Kernel Density Estimate，核密度估计）图。调用plot时加上kind='kde'即可生成一张密度图（标准混合正态分布KDE），如图8-20所示：

```
In [79]: tips['tip_pct'].plot(kind='kde')
```

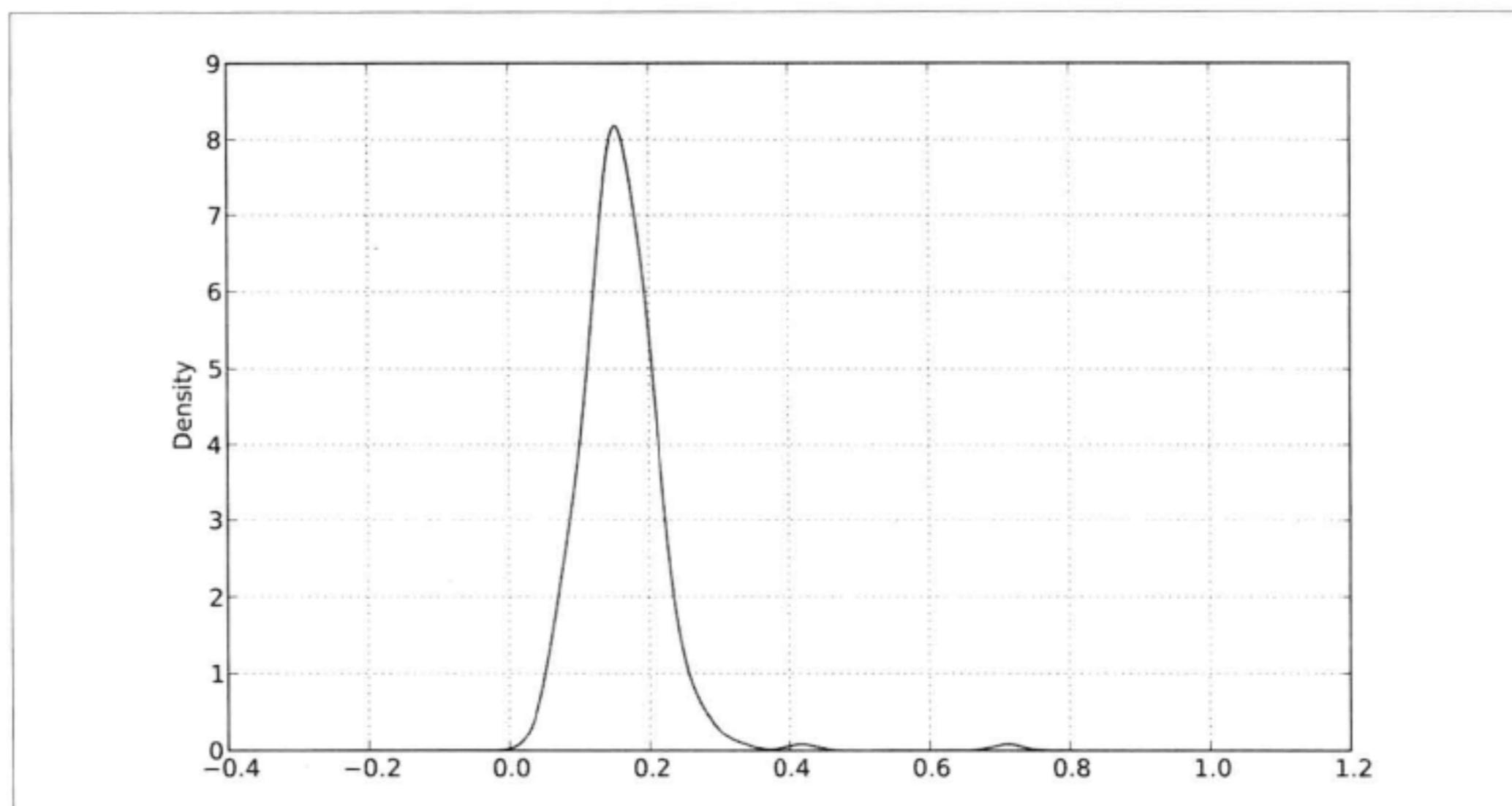


图8-20：小费百分比的密度图

这两种图表常常会被画在一起。直方图以规格化形式给出（以便给出面元化密度），然后再在其上绘制核密度估计。接下来来看一个由两个不同的标准正态分布组成的双峰分布（如图8-21所示）：

```
In [81]: comp1 = np.random.normal(0, 1, size=200) # N(0, 1)
```

```
In [82]: comp2 = np.random.normal(10, 2, size=200) # N(10, 4)
```

```
In [83]: values = Series(np.concatenate([comp1, comp2]))
```

```
In [84]: values.hist(bins=100, alpha=0.3, color='k', normed=True)
Out[84]: <matplotlib.axes.AxesSubplot at 0x5cd2350>
```

```
In [85]: values.plot(kind='kde', style='k--')
```

## 散布图

散布图（scatter plot）是观察两个一维数据序列之间的关系的有效手段。matplotlib的scatter方法是绘制散布图的主要方法。在下面这个例子中，我加载了来自statsmodels项目的macrodata数据集，选择其中几列，然后计算对数差：



```

In [86]: macro = pd.read_csv('ch08/macrodata.csv')

In [87]: data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]

In [88]: trans_data = np.log(data).diff().dropna()

In [89]: trans_data[-5:]
Out[89]:
    cpi      m1  tbilrate  unemp
198 -0.007904  0.045361 -0.396881  0.105361
199 -0.021979  0.066753 -2.277267  0.139762
200  0.002340  0.010286  0.606136  0.160343
201  0.008419  0.037461 -0.200671  0.127339
202  0.008894  0.012202 -0.405465  0.042560

```

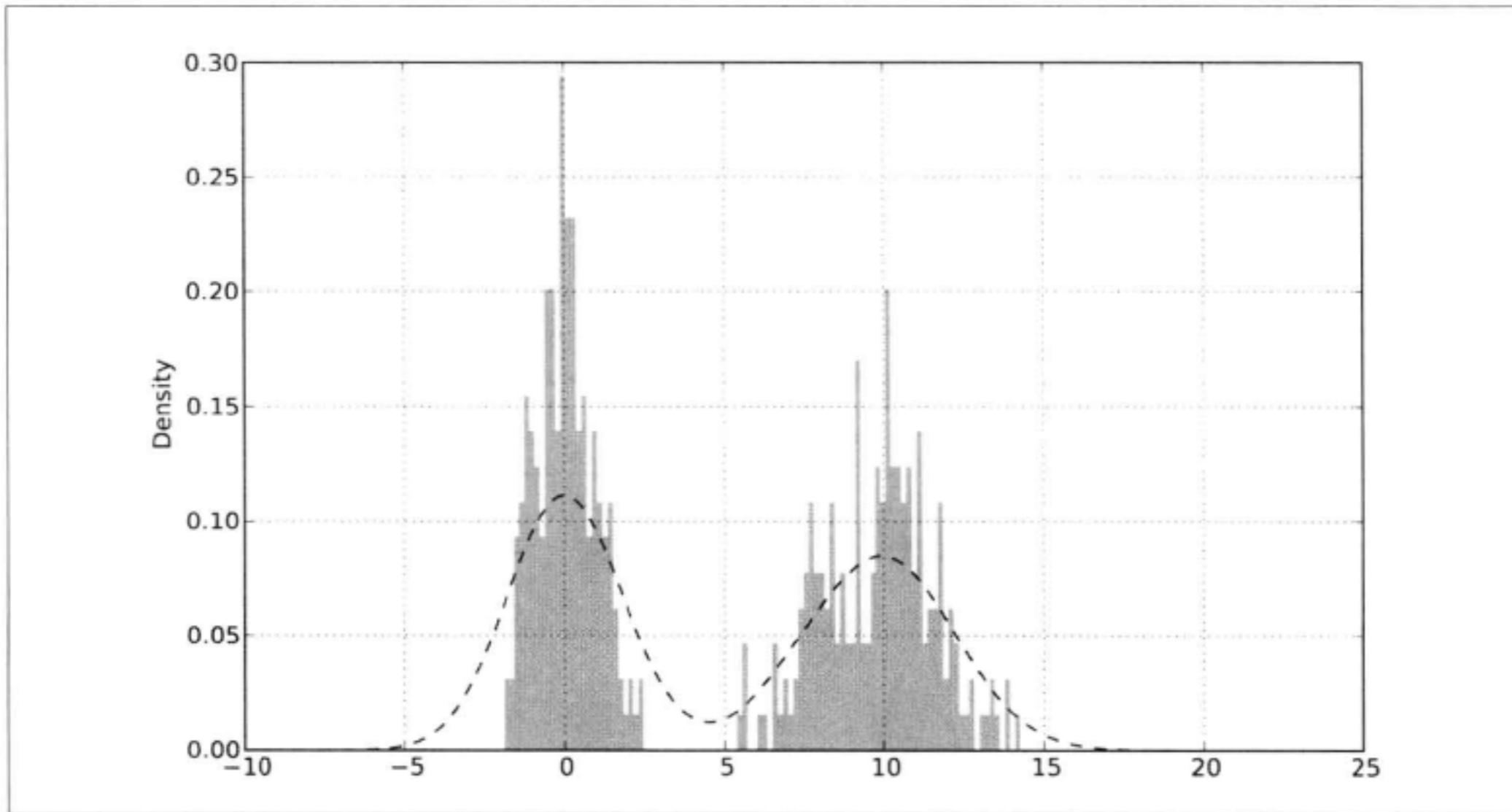


图8-21：带有密度估计的规格化直方图

利用plt.scatter即可轻松绘制一张简单的散布图（如图8-22所示）：

```

In [91]: plt.scatter(trans_data['m1'], trans_data['unemp'])
Out[91]: <matplotlib.collections.PathCollection at 0x43c31d0>

In [92]: plt.title('Changes in log %s vs. log %s' % ('m1', 'unemp'))

```

在探索式数据分析工作中，同时观察一组变量的散布图是很有意义的，这也被称为散布图矩阵（scatter plot matrix）。纯手工创建这样的图表很费工夫，所以pandas提供了一个能从DataFrame创建散布图矩阵的scatter\_matrix函数。它还支持在对角线上放置各变量的直方图或密度图。结果如图8-23所示：

```
In [93]: pd.scatter_matrix(trans_data, diagonal='kde', color='k', alpha=0.3)
```



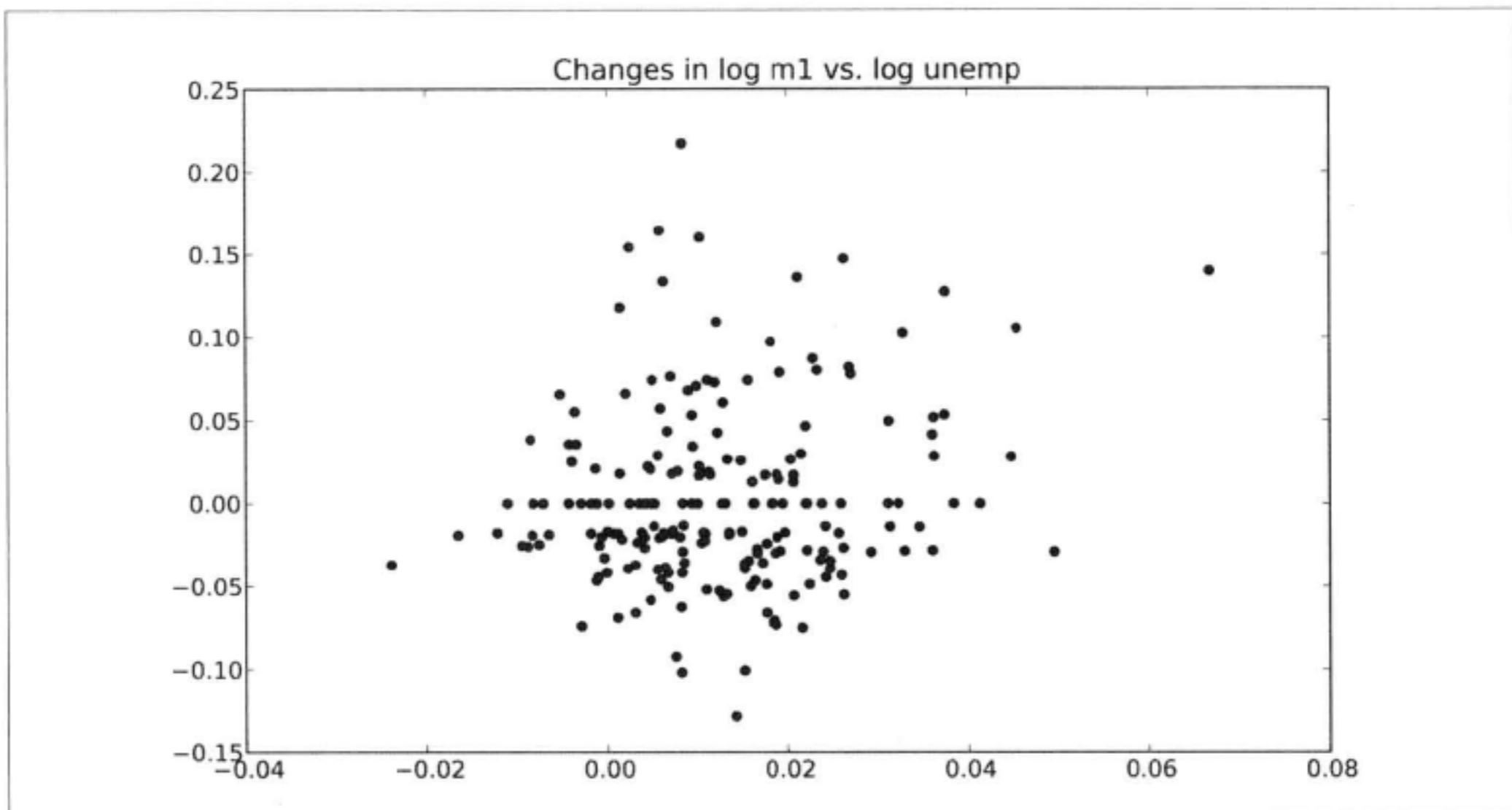


图8-22：一张简单的散布图

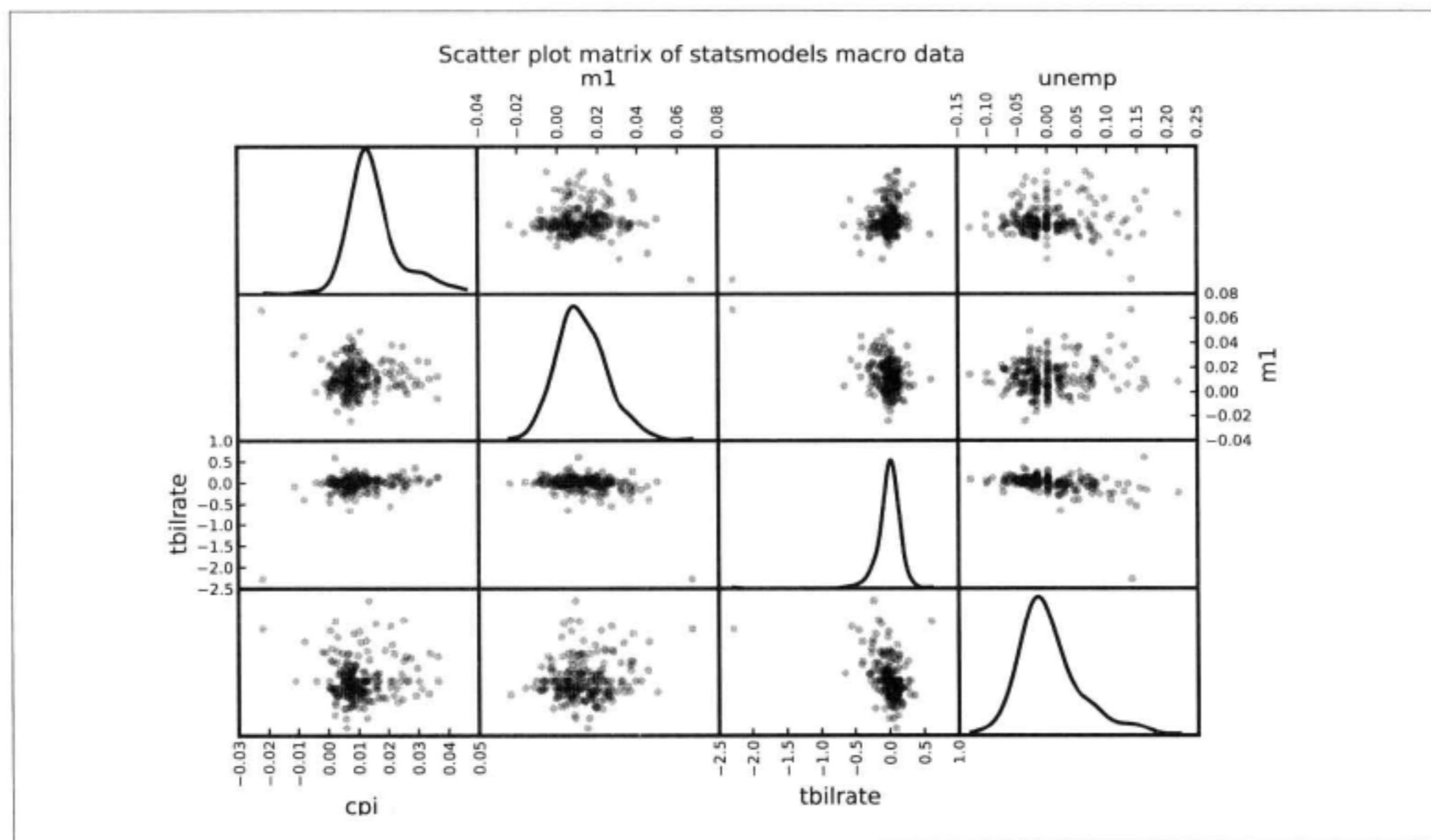


图8-23：statsmodels macro data的散布图矩阵

## 绘制地图：图形化显示海地地震危机数据

Ushahidi是一家非营利软件公司，人们可以通过短信向其提供有关自然灾害和地缘政治事件的信息。这些数据集会被发布在他们的网站 (<http://community.ushahidi.com/>)



*research/datasets/*) 上以供分析和图形化。我下载了2010年海地地震及其余震期间搜集的数据。在本节中，我将告诉你如何利用pandas以及其他目前已经学过的工具处理这些数据，以便为分析和图形化工作做准备。从上面的链接下载好这个CSV文件之后，就可以用read\_csv将其加载到DataFrame中了：

```
In [94]: data = pd.read_csv('ch08/Haiti.csv')
```

```
In [95]: data
Out[95]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3593 entries, 0 to 3592
Data columns:
Serial          3593 non-null values
INCIDENT TITLE  3593 non-null values
INCIDENT DATE   3593 non-null values
LOCATION        3593 non-null values
DESCRIPTION     3593 non-null values
CATEGORY        3587 non-null values
LATITUDE        3593 non-null values
LONGITUDE       3593 non-null values
APPROVED        3593 non-null values
VERIFIED        3593 non-null values
dtypes: float64(2), int64(1), object(7)
```

现在来处理一下这些数据，看看哪些是我们想要的。每一行表示一条从某人的手机上发送的紧急或其他问题的报告。每条报告都有一个时间戳和位置（经度和纬度）：

```
In [96]: data[['INCIDENT DATE', 'LATITUDE', 'LONGITUDE']][:10]
Out[96]:
    INCIDENT DATE  LATITUDE  LONGITUDE
0  05/07/2010 17:26  18.233333 -72.533333
1  28/06/2010 23:06  50.226029  5.729886
2  24/06/2010 16:21  22.278381  114.174287
3  20/06/2010 21:59  44.407062  8.933989
4  18/05/2010 16:26  18.571084 -72.334671
5  26/04/2010 13:14  18.593707 -72.310079
6  26/04/2010 14:19  18.482800 -73.638800
7  26/04/2010 14:27  18.415000 -73.195000
8  15/03/2010 10:58  18.517443 -72.236841
9  15/03/2010 11:00  18.547790 -72.410010
```

CATEGORY字段含有一组以逗号分隔的代码，这些代码表示消息的类型：

```
In [97]: data['CATEGORY'][:6]
Out[97]:
0      1. Urgences | Emergency, 3. Public Health,
1  1. Urgences | Emergency, 2. Urgences logistiques
2  2. Urgences logistiques | Vital Lines, 8. Autre |
3                      1. Urgences | Emergency,
4                      1. Urgences | Emergency,
5  5e. Communication lines down,
Name: CATEGORY
```



只要仔细观察一下上面这个数据摘要，就能发现有些分类信息缺失了，因此我们需要丢弃这些数据点。此外，调用describe还能发现数据中存在一些异常的地理位置：

```
In [98]: data.describe()
Out[98]:
      Serial      LATITUDE      LONGITUDE
count  3593.000000  3593.000000  3593.000000
mean   2080.277484    18.611495   -72.322680
std    1171.100360     0.738572    3.650776
min     4.000000    18.041313   -74.452757
25%   1074.000000    18.524070   -72.417500
50%   2163.000000    18.539269   -72.335000
75%   3088.000000    18.561820   -72.293570
max   4052.000000    50.226029   114.174287
```

清除错误位置信息并移除缺失分类信息是一件很简单的事情：

```
In [99]: data = data[(data.LATITUDE > 18) & (data.LATITUDE < 20) &
...:                 (data.LONGITUDE > -75) & (data.LONGITUDE < -70)
...:                 & data.CATEGORY.notnull()]
```

现在，我们想根据分类对数据做一些分析或图形化工作，但是各个分类字段中可能含有多个分类。此外，各个分类信息不仅有一个编码，还有一个英文名称（可能还有一个法语名称）。因此需要对数据做一些规整化处理。首先，我编写了两个函数<sup>译注5</sup>，一个用于获取所有分类的列表，另一个用于将各个分类信息拆分为编码和英语名称：

```
def to_cat_list(catstr):
    stripped = (x.strip() for x in catstr.split(','))
    return [x for x in stripped if x]

def get_all_categories(cat_series):
    cat_sets = (set(to_cat_list(x)) for x in cat_series)
    return sorted(set.union(*cat_sets))

def get_english(cat):
    code, names = cat.split('.')
    if '|' in names:
        names = names.split(' | ')[1]
    return code, names.strip()
```

你可以测试一下get\_english函数是否工作正常：

```
In [101]: get_english('2. Urgences logistiques | Vital Lines')
Out[101]: ('2', 'Vital Lines')
```

接下来，我做了一个将编码跟名称映射起来的字典，这是因为我们等会儿要用编码进行

---

译注5：读者就当做两个吧。



分析。后面我们在修饰图表时也会用到这个（注意，这里用的是生成器表达式，而不是列表推导式）：

```
In [102]: all_cats = get_all_categories(data.CATEGORY)

# 生成器表达式
In [103]: english_mapping = dict(get_english(x) for x in all_cats)

In [104]: english_mapping['2a']
Out[104]: 'Food Shortage'

In [105]: english_mapping['6c']
Out[105]: 'Earthquake and aftershocks'
```

根据分类选取记录的方式有很多，其中之一是添加指标（或哑变量）列，每个分类一列。为此，我们首先抽取出唯一的分类编码，并构造一个全零DataFrame（列为分类编码，索引跟data的索引一样）：

```
def get_code(seq):
    return [x.split('.')[0] for x in seq if x]

all_codes = get_code(all_cats)
code_index = pd.Index(np.unique(all_codes))
dummy_frame = DataFrame(np.zeros((len(data), len(code_index))), index=data.index,
                        columns=code_index)
```

如果一切顺利，`dummy_frame`应该是这样的：

```
In [107]: dummy_frame.ix[:, :6]
Out[107]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3569 entries, 0 to 3592
Data columns:
 1      3569 non-null values
 1a     3569 non-null values
 1b     3569 non-null values
 1c     3569 non-null values
 1d     3569 non-null values
 2      3569 non-null values
dtypes: float64(6)
```

你可能已经想到了，现在应该将各行中适当的项设置为1，然后再与`data`进行连接：

```
for row, cat in zip(data.index, data.CATEGORY):
    codes = get_code(to_cat_list(cat))
    dummy_frame.ix[row, codes] = 1

data = data.join(dummy_frame.add_prefix('category_'))
```

现在`data`有了一些新的列：



```
In [109]: data.ix[:, 10:15]
Out[109]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3569 entries, 0 to 3592
Data columns:
category_1      3569 non-null values
category_1a     3569 non-null values
category_1b     3569 non-null values
category_1c     3569 non-null values
category_1d     3569 non-null values
dtypes: float64(5)
```

接下来开始画图吧！由于这是空间坐标数据，因此我们希望把数据绘制在海地的地图上。`basemap`工具集 (<http://matplotlib.github.com/basemap>, `matplotlib`的一个插件) 使得我们能够用Python在地图上绘制2D数据。`basemap`提供了许多不同的地球投影以及一种将地球上的经纬度坐标投影转换为二维`matplotlib`图的方式。经过一遍又一遍地尝试，我编写了下面这个函数，它可以绘制出一张简单的黑白海地地图：

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

def basic_haiti_map(ax=None, lllat=17.25, urlat=20.25,
                     lllon=-75, urlon=-71):
    # 创建极球面投影的Basemap实例。
    m = Basemap(ax=ax, projection='stere',
                 lon_0=(urlon + lllon) / 2,
                 lat_0=(urlat + lllat) / 2,
                 llcrnrlat=lllat, urcrnrlat=urlat,
                 llcrnrlon=lllon, urcrnrlon=urlon,
                 resolution='f')
    # 绘制海岸线、州界、国界以及地图边界。
    m.drawcoastlines()
    m.drawstates()
    m.drawcountries()
    return m
```

现在的问题是，如何让返回的这个`Basemap`对象知道该怎样将坐标转换到画布上。我编写了下面的代码来绘制数据。对于每一个分类，我在数据集中找到了对应的坐标，并在适当的`subplot`中绘制一个`Basemap`，转换坐标，然后通过`Basemap`的`plot`方法绘制点：

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
fig.subplots_adjust(hspace=0.05, wspace=0.05)

to_plot = ['2a', '1', '3c', '7a']

lllat=17.25; urlat=20.25; lllon=-75; urlon=-71

for code, ax in zip(to_plot, axes.flat):
    m = basic_haiti_map(ax, lllat=lllat, urlat=urlat,
                         lllon=lllon, urlon=urlon)
```



```
cat_data = data[data['category_%s' % code] == 1]

# 计算地图的投影坐标。
x, y = m(cat_data.LONGITUDE, cat_data.LATITUDE)

m.plot(x, y, 'k.', alpha=0.5)
ax.set_title('%s: %s' % (code, english_mapping[code]))
```

最终结果如图8-24所示。

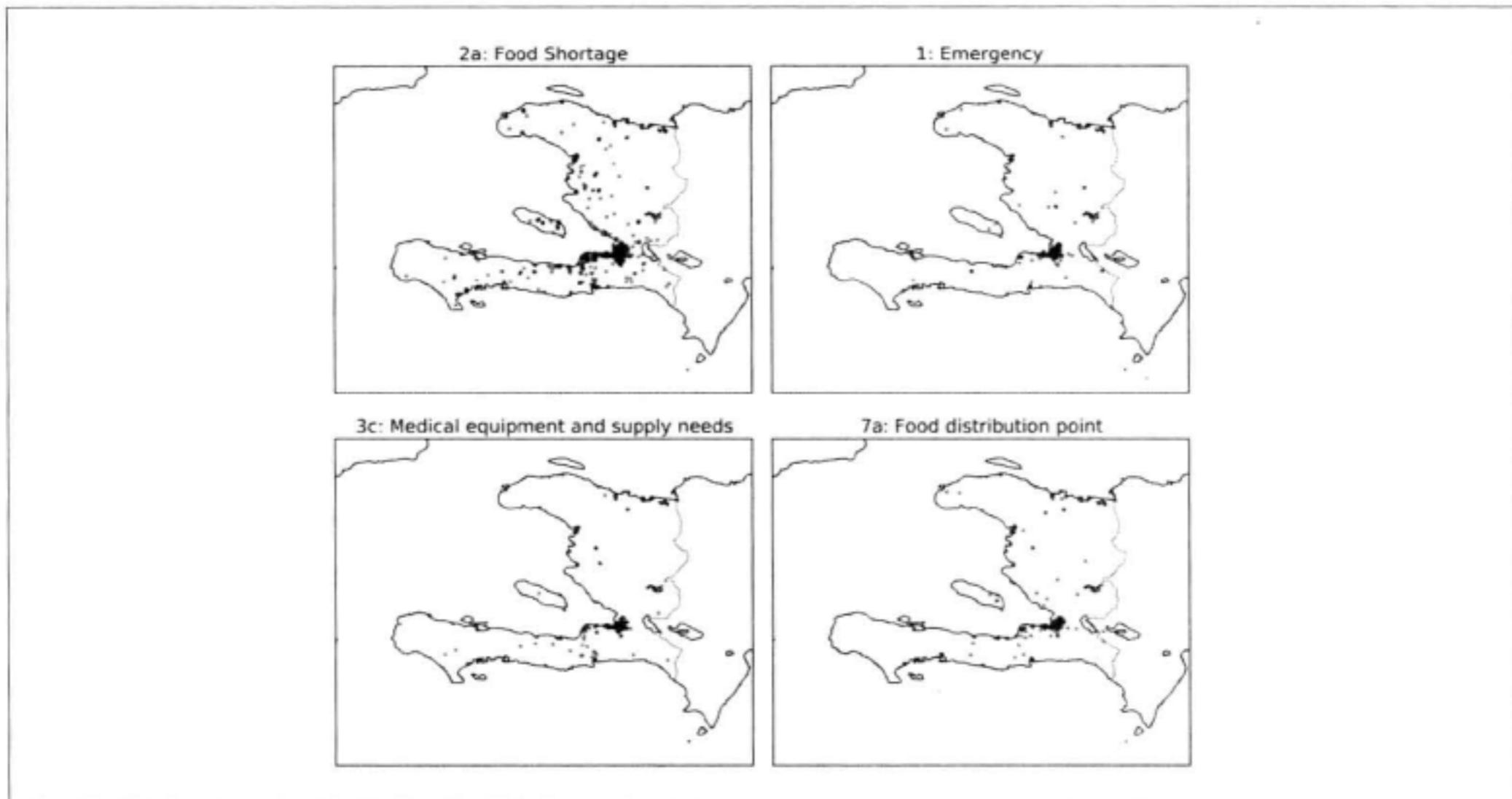


图8-24：海地地震的4类数据

从图中可以看出，大部分数据都集中在人口最稠密的城市——太子港。`basemap`还可以叠加来自`shapefile`的地图数据。我先下载了一个带有太子港道路的`shapefile`（参见 [http://cegrp.cga.harvard.edu/haiti/?q=resources\\_data](http://cegrp.cga.harvard.edu/haiti/?q=resources_data)）。`Basemap`对象有一个非常方便的`readshapefile`方法，于是在解压完道路数据文件之后，我只在代码中加以下几行就可以了：

```
shapefile_path = 'ch08/PortAuPrince_Roads/PortAuPrince_Roads'
m.readshapefile(shapefile_path, 'roads')
```

在对经纬度边界进行了一番尝试之后，我做了一张反映食物短缺情况的图片，如图8-25所示。



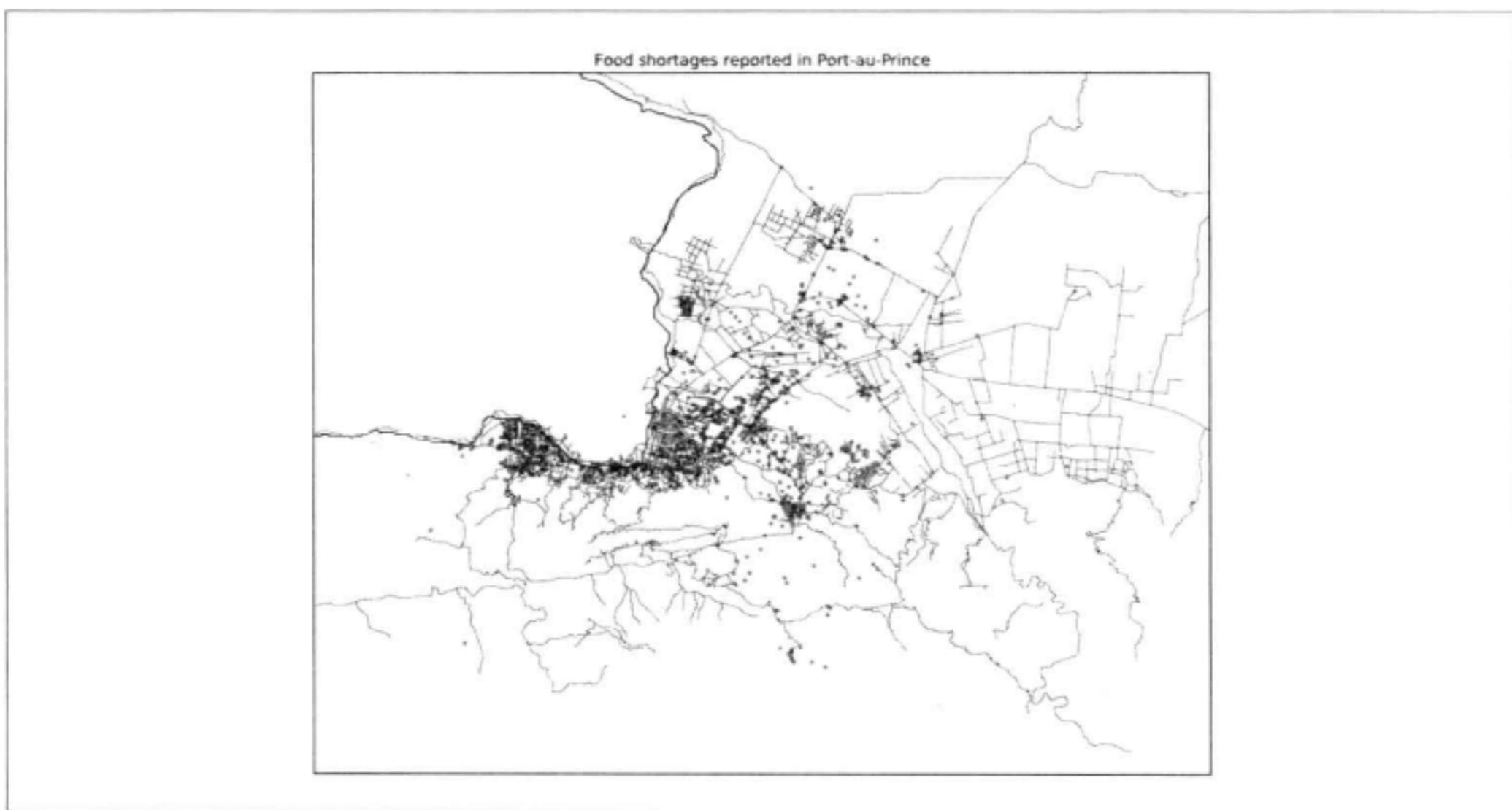


图8-25：海地大地震期间，太子港的食物短缺报告

## Python图形化工具生态系统

用Python创建图形的方式非常多（根本罗列不完）。除了开源库，商业库也不少。

本书主要涉及的是matplotlib，因为它是Python领域中使用最广泛的绘图工具。虽然matplotlib是Python科学计算生态系统的重要组成部分，但它在统计图表的创建和展示方面仍然有许多缺点。MATLAB用户可能会对matplotlib感到熟悉，而R用户（尤其是使用ggplot2和trellis的那些）可能就会比较郁闷了（至少目前是）。虽然matplotlib可以为Web应用创建漂亮的图表，但这通常需要耗费大量的精力，因为它原本是为印刷而设计的。先不管美不美观，至少它足以应付大部分需求。在pandas中，我跟其他开发人员一直都在寻求使数据分析中的大部分绘图工作变得更简单的办法。

广泛使用的图形化工具很多。这里我只列举几个，但建议你研究一下整个生态系统。

### Chaco

Chaco (<http://code.enthought.com/chaco/>) 是由Enthought开发的一个绘图工具包，它既可以绘制静态图又可以生成交互式图形，如图8-26所示。它非常适合用复杂的图形化方式表达数据的内部关系。跟matplotlib相比，Chaco对交互的支持要好得多，而且渲染速度很快。如果要创建交互式的GUI应用程序，它确实是个不错的选择。



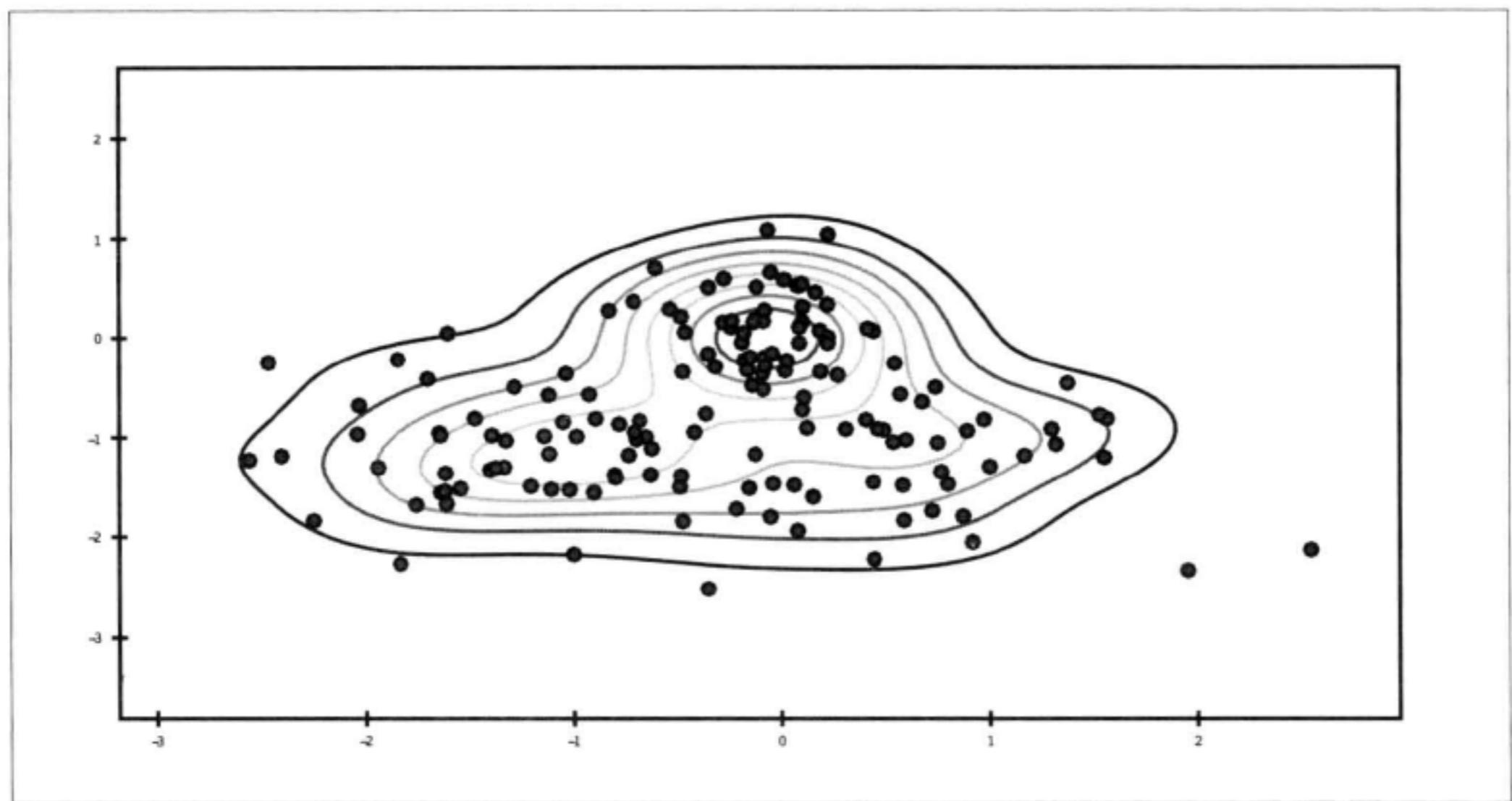


图8-26：Chaco图示例

## mayavi

mayavi项目（由Prabhu Ramachandran、Gaël Varoquaux等人开发）是一个基于开源C++图形库VTK的3D图形工具包。跟matplotlib一样，mayavi也能集成到IPython以实现交互式使用。通过鼠标和键盘操作，图形可以被平移、旋转、缩放。在第12章中，我用mayavi制作了一张有关广播的插图。我没有给出任何调用mayavi的代码，但你可以在网上找到很多文档和示例。我相信它能成为WebGL（以及相关产品）的替代品，虽然其生成的图形很难以交互的形式共享。

## 其他库

当然，Python领域中还有许多其他的图形化库和应用程序：PyQwt、Veusz、gnuplotpy、biggles等。我就曾经见过PyQwt被用在基于Qt框架（ PyQt）的GUI应用程序中。许多库都还在不断地发展（有些已经被用在大型应用程序当中了）。近几年来，我发现了一个总体趋势：大部分库都在向基于Web的技术发展，并逐渐远离桌面图形技术。下面我要就这个问题多说几句。

## 图形化工具的未来

基于Web技术（比如JavaScript）的图形化是必然的发展趋势。毫无疑问，许多基于Flash或JavaScript的静态或交互式图形化工具已经出现了很多年。而且类似的新工具包（如



d3.js及其分支项目)一直都在不断涌现。相比之下,非Web式的图形化开发工作在近几年中减慢了许多。Python以及其他数据分析和统计计算环境(如R)都是如此。

于是,开发方向就变成了实现数据分析和准备工具(如pandas)与Web浏览器之间更为紧密的集成。我希望这个思路今后能成为Python以及非Python用户之间富有成效的协作手段。

