第5章

Python数据分析基础库

本章学习目标：

- 学习 NumPy 库的用法、数据结构和基本操作。
- 学习 Pandas 库的用法、数据结构和基本操作。
- 学习 Matplotlib 库的用法、数据结构和基本操作。

本章介绍 Python 进行数据分析时常用的 NumPy、Pandas 和 Matplotlib 基础库。NumPy 是 Python 的一种开源数值计算扩展库，这种工具可用来存储和处理大型矩阵，比 Python 自身的嵌套列表（nested list structure）结构要高效许多；Pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的，Pandas 提供了大量的库和标准数据模型及高效、便捷地处理大型数据集所需的函数和方法；Matplotlib 是一个 Python 的 2D 绘图库，它基于各种硬拷贝格式和跨平台的交互式环境生成出版质量级别的图形。

5.1 NumPy

NumPy(Numerical Python)是一个开源的 Python 科学计算库，包含很多实用的数学函数，涵盖线性代数运算、傅里叶变换和随机数生成等功能。NumPy 允许用户进行快速交互式原型设计，可以很自然地使用数组和矩阵。它的部分功能如下。

(1) ndarray：一个具有矢量算术运算功能且节省空间的多维数组。

(2) 用于对整组数据进行快速运算的标准数学函数(无须编写循环)。

(3) 用于读/写磁盘数据的工具及操作内存映射文件的工具。

(4) 线性代数、随机数生成及傅里叶变换功能。

（5）用于集成C、C++、FORTRAN等语言的代码编写工具。

NumPy的底层算法在设计时就有着优异的性能，对于同样的数值计算任务来说，使用NumPy要比直接编写Python代码便捷得多。对于大型数组的运算来说，使用NumPy数组的存储效率和输入输出性能均优于Python中等价的基本数据结构（例如嵌套的list容器）。对于TB级的大文件来说，NumPy使用内存映射文件来处理，以达到最优的数据读写性能。这是因为NumPy能够直接对数组和矩阵进行操作，可以省略很多循环语句，其众多的数学函数也会让开发人员编写代码的工作轻松许多。不过NumPy数组的通用性不及Python提供的list容器，这是其不足之处。因此，在科学计算之外的领域，NumPy的优势也就不那么明显了。NumPy本身没有提供那么多高级的数据分析功能，理解NumPy数组及面向数组的计算将有助于更加高效地使用诸如Pandas之类的工具。下面对NumPy的数据结构和操作进行介绍。

NumPy的多维数组对象ndarray是一个快速、灵活的大数据集容器。用户可以利用这种数组对象对整块数据进行数学运算，其运算跟标量元素之间的运算一样。创建ndarray数组最简单的办法就是使用array()函数，它接收一切序列型的对象（包括其他数组），然后产生一个新的、含有传入数据的NumPy数组。这里以一个列表的转换为例：

```
In [1]: import numpy as np
        data = [6, 7.5, 8, 0, 1]
        arr1 = np.array(data)
        arr1
Out[1]: array([ 6. ,  7.5,  8. ,  0. ,  1. ])
```

ndarray是一个通用的同构数据多维容器，其中所有的元素必须是相同类型的。每一个数组都有一个shape（表示维度大小的数组）和一个dtype（用于说明数组数据类型的对象）。

```
In [2]: arr1.shape
Out[2]: (5,)
In [3]: arr1.dtype
Out[3]: dtype('float64')
```

嵌套序列（例如，由一组等长列表组成的列表）将会被转换成一个多维数组。

```
In [4]: data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
        arr2 = np.array(data2)
        arr2
Out[4]:
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
In [5]: arr2.ndim
Out[5]: 2
In [6]: arr2.shape
Out[6]: (2, 4)
```

除非显式说明，否则np.array()会尝试为新建的数组推断出一个较为合适的数据类型。数据类型保存在一个特殊的dtype对象中，例如上面的两个例子。

```
In [7]: arr1.dtype
Out[7]: dtype('float64')
In [8]: arr2.dtype
Out[8]: dtype('int32')
```

除了 np.array()外，还有一些函数可以新建数组，例如，np.zeros()和 np.ones()可以分别创建指定长度或形状全为 0 或全为 1 的数组。Empty 可以创建一个没有任何具体数值的数组。如果要用这些方法创建数组，只需传入一个表示形状的元组即可。

```
In [9]: np.zeros(8)
Out[9]: array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
In [10]: np.zeros((2, 4))
Out[10]:
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
In [11]: np.empty((2, 3, 2))
Out[11]:
array([[[  9.78249979e-322,   0.00000000e+000],
        [  0.00000000e+000,   0.00000000e+000],
        [  0.00000000e+000,   0.00000000e+000]],

       [[  0.00000000e+000,   0.00000000e+000],
        [  0.00000000e+000,   0.00000000e+000],
        [  0.00000000e+000,   0.00000000e+000]]])
```

在 NumPy 中，np.empty()会认为返回全为 0 的数组是不安全的，所以它会返回一些未初始化的很接近 0 的随机值。

ndarray 的一些常用的基本数据操作函数如表 5.1 所示。

表 5.1　ndarray 基本数据操作函数

函　　数	说　　明
array()	将输入数据(列表、元组、数组或其他序列类型)转换为 ndarray。推断出 dtype 或特别指定 dtype，默认直接赋值输入数据
asarray()	将输入转换为 ndarray。如果输入本身是一个 ndarray，就不再复制
arange()	类似于内置的 range，但返回的是一个 ndarray 而非 list
ones()，ones_like()	根据指定的形状和 dtype 创建一个全 1 数组。ones_like 以另一个数组为参数，并根据其形状和 dtype 创建一个全 1 数组
zeros()，zeros_like()	类似于 ones()和 ones_like()，只不过产生的是全 0 数组
empty()，empty_like()	创建新数组，只分配内存空间，不填充任何值
full()，full_like()	用 full value 中的所有值，根据指定的形状和 dtype 创建一个数组。full_like()使用另一个数组，用相同的形状和 dtype 创建
eye()，identity()	创建一个正方的 N×N 矩阵(对角线为 1，其余为 0)

扫一扫

视频讲解

5.1.1　ndarray 的数据类型

dtype(数据类型)是一个特殊的对象，它含有 ndarray 将一块内存解释为特定数据类型所需的信息。

```
In [12]: arr3 = np.array([1, 2, 3], dtype = np.float64)
         arr3
Out[12]: array([ 1.,  2.,  3.])
In [13]: arr4 = np.array([1, 2, 3], dtype = np.int32)
         arr4
Out[13]: array([1, 2, 3])
```

dtype 是 NumPy 如此强大和灵活的原因之一。在多数情况下，它直接映射到相应的机器表示，这使得“读写磁盘上的二进制数据流”及“集成低级语言代码”等工作变得更加简单。数值型 dtype 的命名形式相同：一个类型名（例如 float 或 int），后面跟一个用于表示各元素位长的数字。标准的双精度浮点值（即 Python 中的 float 对象）需要占用 8B（即 64b）。因此，该类型在 NumPy 中记作 float64。

可以用 astype 方法显式更改数组的 dtype。

```
In [14]: arr5 = np.array([1, 2, 3])
  arr5.dtype
Out[14]: dtype('int32')
In [15]: arr6 = arr5.astype(np.float64)
  arr6
Out[15]: array([ 1.,  2.,  3.])
```

扫一扫

视频讲解

5.1.2 数组和标量之间的运算

用数组表达式代替循环的方法，通常被称作矢量化（vectorization）。大小相等的数组之间的任何算术运算都会应用到元素集。

```
In [16]: arr = np.array([[1. ,2. , 3.], [4. ,5. ,6]])
          arr * arr
Out[16]:
array([[  1.,   4.,   9.],
       [ 16.,  25.,  36.]])
In [17]: arr - arr
Out[17]:
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

同样，数组和标量的运算也会将那个标量传播到各个元素。

```
In [18]: 1 / arr
Out[18]:
array([[ 1.        ,  0.5       ,  0.33333333],
       [ 0.25      ,  0.2       ,  0.16666667]])
In [19]: arr * * 0.5
Out[19]:
array([[ 1.        ,  1.41421356,  1.73205081],
       [ 2.        ,  2.23606798,  2.44948974]])
```

扫一扫

视频讲解

5.1.3　索引和切片

NumPy索引和切片是一个内容丰富的主题，因为选取数据子集或单个元素的方式有很多。首先，一维数组的切片索引基本和Python列表的切片索引功能一致。

```
In [20]: arr = np.arange(10)
         arr
Out[20]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
In [21]: arr[4]
Out[21]: 4
In [22]: arr[3:7]
Out[22]: array([3, 4, 5, 6])
In [23]: arr[3:5] = 12
         arr
Out[23]: array([ 0,  1,  2, 12, 12,  5,  6,  7,  8,  9])
```

如上所示，当将一个标量赋值给一个切片时（例如，arr[3:5]＝12），该值会自动传播到整个选区。因为数组切片是原始数组视图，这就意味着如果做任何修改，原始数组都会跟着更改。

```
In [24]: arr_slice = arr[3:5]
         arr_slice[1] = 100
         arr
Out[24]: array([  0,   1,   2,  12, 100,   5,   6,   7,   8,   9])
In [25]: arr_slice[:] = 250
         arr
Out[25]: array([  0,   1,   2, 250, 250,   5,   6,   7,   8,   9])
```

对于高维数组来说，能做的事情更多。在一个二维数组中，各索引位置上的元素不再是标量而是一维数组。

```
In [26]: arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
         arr[2]
Out[26]: array([7, 8, 9])
```

因此，可以对各个元素进行递归访问，但这样需要做的事情有点多。用户可以传入一个以逗号隔开的索引列表来选取单个元素。也就是说，下面这两种方式是等价的。

```
In [27]: arr[1][2]
Out[27]: 6
In [28]: arr[1, 2]
Out[28]: 6
```

花式索引是利用整数数组进行索引，假设有一个8×4的数组：

```
In [29]: arr = np.empty((8,4))
         for i in range(8):
             arr[i] = i
         arr
Out[29]:
array([[ 0.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  1.],
       [ 2.,  2.,  2.,  2.],
       ...,
       [ 5.,  5.,  5.,  5.],
       [ 6.,  6.,  6.,  6.],
       [ 7.,  7.,  7.,  7.]])
```

为了以特定的顺序选取行子集，只需传入一个用于指定顺序的整数列表或 ndarray 即可。

```
In [30]: arr[[4, 3, 0, 6]]
Out[30]:
array([[ 4.,  4.,  4.,  4.],
       [ 3.,  3.,  3.,  3.],
       [ 0.,  0.,  0.,  0.],
       [ 6.,  6.,  6.,  6.]])
```

使用负数索引将会从末尾开始选取行。

```
In [31]: arr[[ -3, -5, -7]]
Out[31]:
array([[ 5.,  5.,  5.,  5.],
       [ 3.,  3.,  3.,  3.],
       [ 1.,  1.,  1.,  1.]])
```

当一次传入多个数组时，它返回的是一个一维数组，其中的元素对应各个索引元组。

```
In [32]: arr = np.arange(32).reshape((8,4))
         arr
Out[32]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       ...,
       [20, 21, 22, 23],
       [24, 25, 26, 27],
       [28, 29, 30, 31]])
In [33]: arr[[1,5,7,2], [0,3,1,2]]
Out[33]: array([ 4, 23, 29, 10])
```

它选出的元素其实是(1,0)、(5,3)、(7,1)和(2,2)这些位置的元素。这个花式索引的结果可能和某些用户预测的不太一样，选取矩阵的行列子集应该是矩形区域的形式才

对。下面是得到该结果的一个办法：

```
In [34]: arr[[1,5,7,2]][:,[0,3,1,2]]
Out[34]:
array([[ 4,  7,  5,  6],
       [20, 23, 21, 22],
       [28, 31, 29, 30],
       [ 8, 11,  9, 10]])
```

另外一个办法就是使用 np.ix_()函数，它可以将两个一维数组转换成一个用于选取方形区域的索引器。

```
In [35]: arr[np.ix_([1,5,7,2], [0,3,1,2])]
Out[35]:
array([[ 4,  7,  5,  6],
       [20, 23, 21, 22],
       [28, 31, 29, 30],
       [ 8, 11,  9, 10]])
```

注意：花式索引和切片不一样，它是将数据复制到新的数组中。

扫一扫

视频讲解

5.1.4　数组转置和轴对换

转置(transpose)是重塑的一种特殊形式，它返回的是源数据的视图(不会进行任何复制操作)。数组不仅有 transpose()方法，还有一个特殊的 T 属性。

```
In [36]: arr = np.arange(15).reshape(5,3)
         Arr
Out[36]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])

In [37]: arr.T
Out[37]:
array([[ 0,  3,  6,  9, 12],
       [ 1,  4,  7, 10, 13],
       [ 2,  5,  8, 11, 14]])
```

在进行矩阵计算时，经常需要用到该操作，例如，利用 np.dot()计算矩阵内积。

```
In [38]: arr = np.random.randn(6,3)
np.dot(arr.T, arr)
Out[38]:
array([[  9.03630405,   0.49388948,  -1.54587135],
       [  0.49388948,   2.25164741,   1.93791071],
       [ -1.54587135,   1.93791071,  10.55460651]])
```

对于高维数组来说，transpose()需要得到一个由轴编号组成的元组才能对这些轴进

行转置。

```
In [39]: arr = np.arange(16).reshape((2, 2, 4))
Arr
Out[39]:
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],

       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]]])
In [40]: arr.transpose((1, 0, 2))
Out[40]:
array([[[ 0,  1,  2,  3],
        [ 8,  9, 10, 11]],

       [[ 4,  5,  6,  7],
        [12, 13, 14, 15]]])
```

扫一扫

视频讲解

5.1.5 利用数组进行数据处理

NumPy 数组可以将很多数据处理任务表述为简洁的数组表达式（否则需要编写循环）。矢量化数组运算要比 Python 方式快一两个数量级，尤其是对于各种数值运算来说。例如，np.meshgrid()函数接收两个一维数组，并产生两个二维矩阵（对应两个数组中所有的(x,y)对）。

```
In [41]: points = np.arange( - 5, 5, 0.01)      # 1000 个间隔相等的点
         xs, ys = np.meshgrid(points, points)
        ys
Out[41]:
array([[ - 5.  , - 5.  , - 5.  , ..., - 5.  , - 5.  , - 5.  ],
       [ - 4.99, - 4.99, - 4.99, ..., - 4.99, - 4.99, - 4.99],
       [ - 4.98, - 4.98, - 4.98, ..., - 4.98, - 4.98, - 4.98],
       ...,
       [ 4.97,  4.97,  4.97, ...,  4.97,  4.97,  4.97],
       [ 4.98,  4.98,  4.98, ...,  4.98,  4.98,  4.98],
       [ 4.99,  4.99,  4.99, ...,  4.99,  4.99,  4.99]])
```

假设在一组值上计算函数 sqrt(x^2 + y^2)，这时对函数的求值运算就好办了，把这两个数组当作两个浮点数编写表达式即可。

```
In [42]: import matplotlib.pyplot as plt
         z = np.sqrt(xs * * 2 + ys * * 2)
         z
Out[42]:
array([[ 7.07106781,  7.06400028,  7.05693985, ...,  7.04988652,
         7.05693985,  7.06400028],
       [ 7.06400028,  7.05692568,  7.04985815, ...,  7.04279774,
         7.04985815,  7.05692568],
```

```
       [ 7.05693985,  7.04985815,  7.04278354, ...,  7.03571603,
         7.04278354,  7.04985815],
       ...,
       [ 7.04988652,  7.04279774,  7.03571603, ...,  7.0286414 ,
         7.03571603,  7.04279774],
       [ 7.05693985,  7.04985815,  7.04278354, ...,  7.03571603,
         7.04278354,  7.04985815],
       [ 7.06400028,  7.05692568,  7.04985815, ...,  7.04279774,
         7.04985815,  7.05692568]])
In [43]: plt.imshow(z, cmap = plt.cm.gray)
         plt.colorbar()
       plt.title('Image plot of $\sqrt{x^2 + y^2}$ for a grid of values')
Out[43]: <matplotlib.text.Text at 0x1086aa90>
```

函数值的图形化结果如图 5.1 所示。

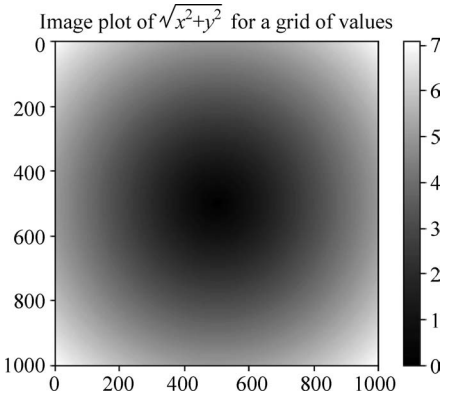

图 5.1　根据网格对函数求值的结果

5.1.6　数学和统计方法

用户可以通过数组上的一组数学函数对整个数组或某个轴向的数据进行统计计算。

```
In [44]: arr = np.random.randn(5, 4)        #产生正态分布数据
         arr.mean()
Out[44]: -0.24070480645161735
In [45]: np.mean(arr)
Out[45]: -0.24070480645161735
In [46]: arr.sum()
Out[46]: -4.8140961290323467
```

mean()和 sum()这类函数可以接收一个 axis 参数(用于计算该轴向上的统计值),最终结果是一个少一维的数组。

```
In [47]: arr.mean(axis = 1)
Out[47]: array([-0.26271711, -0.50185429,  0.38508322, -0.25435201, -0.56968384])
In [48]: arr.sum(0)
Out[48]: array([ 0.81837351, -2.17245972, -4.01616748,  0.55615755])
```

像 cumsum()和 cumprod()之类的方法则不聚合，而是产生一个由中间结果组成的数组。

```
In [49]: arr = np.array([[0,1,2], [3,4,5], [6,7,8]])
         arr.cumsum(0)
Out[49]:
array([[ 0,  1,  2],
       [ 3,  5,  7],
       [ 9, 12, 15]], dtype = int32)
In [50]: arr.cumprod(1)
Out[50]:
array([[  0,   0,   0],
       [  3,  12,  60],
       [  6,  42, 336]], dtype = int32)
```

5.2 Pandas

Pandas 的名称来自面板数据（panel data）和 Python 数据分析（data analysis）。Pandas 是一种基于 NumPy 的数据分析包，最初由 AQR Capital Management 于 2008 年 4 月作为金融数据分析工具开发出来，并于 2009 年底开源，目前由专注于 Python 数据包开发的 PyData 开发小组继续维护。Pandas 提供了大量的高效操作大型数据集所需的函数和方法，它是使 Python 成为强大而高效的数据分析工具的重要因素之一。

5.2.1 Pandas 数据结构

1. Series

Series 是一种类似于一维数组的对象，它由一组数据及与之相关的一组数据标签（即索引）组成。只有一组数据可产生最简单的 Series。

```
In [1]: import pandas as pd
         from pandas import Series, DataFrame
         obj = Series([4, 7, -5, 3])
         obj
Out[1]:
0    4
1    7
2   -5
3    3
dtype: int64
```

Series 的字符串表现形式为索引在左边，值在右边。由于没有为数据指定索引，会自动创建一个 0～(N－1)(N 为数据长度)的整数型索引。用户可以通过 Series 的 values 和 index 属性获取其数组表示形式和索引对象。

```
In [2]: obj.values
Out[2]: array([ 4,  7, -5,  3], dtype = int64)
In [3]: obj.index
Out[3]: RangeIndex(start = 0, stop = 4, step = 1)
```

通常，需要创建的 Series 带有一个可以对各个数据点进行标记的索引。

```
In [4]: obj2 = Series([4,3, -5,7], index = ['d','b','a','c'])
        obj2
Out[4]:
d    4
b    3
a   -5
c    7
dtype: int64
```

与普通的 NumPy 数组相比，可以通过索引的方式选取 Series 中的单个或一组值。

```
In [5]: obj2['a']
Out[5]: -5
In [6]: obj2[['c', 'a', 'd']]
Out[6]:
c    7
a   -5
d    4
dtype: int64
```

2. DataFrame

DataFrame 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的类型（数值型、字符串、布尔型等）。DataFrame 既有行索引，又有列索引，可以看作是由 Series 组成的字典（共用同一个索引）。跟其他类似的数据结构相比，DataFrame 中面向行和面向列的操作基本是平衡的。构建 DataFrame 的方法很多，最常见的就是直接传入一个由等长列表或 NumPy 数组组成的字典。

```
In [7]: data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
                'year':[2000, 2001, 2002, 2001, 2002],
                'pop':[1.5, 1.7, 3.6, 2.4, 2.9]}
    frame = DataFrame(data)
    frame
Out[7]:
   pop   state   year
0  1.5    Ohio  2000
1  1.7    Ohio  2001
2  3.6    Ohio  2002
3  2.4  Nevada  2001
4  2.9  Nevada  2002
```

如果指定了列序列，DataFrame 的列就会按照指定的顺序进行排列。

```
In [8]: DataFrame(data, columns = ['year', 'state', 'pop'])
Out[8]:
   year   state   pop
0  2000    Ohio  1.5
1  2001    Ohio  1.7
2  2002    Ohio  3.6
3  2001  Nevada  2.4
4  2002  Nevada  2.9
```

5.2.2 Pandas 文件操作

1. Pandas 读取文件

Pandas 提供了一些用于将表格型数据读取为 DataFrame 对象的函数。表 5.2 对它们进行了总结，其中 read_csv()和 read_table()可能会是今后用得最多的。

表 5.2 Pandas 读取文件的函数

函　数	说　明
read_csv()	从文件、URL、文件型对象中加载带分隔符的数据。默认分隔符为逗号
read_table()	从文件、URL、文件型对象中加载带分隔符的数据。默认分隔符为制表符("\t")
read_fwf()	读取定宽列格式数据(也就是说没有分隔符)
read_clipboard()	读取剪贴板中的数据，可以看作是 read_table()的剪贴板。它在将网页转换为表格时非常有用

2. Pandas 导出文件

Pandas 导出文件的函数如表 5.3 所示。

表 5.3 Pandas 导出文件的函数

函　数	说　明
to_csv(file_path,sep=' ',index=True,header=True)	file_path 表示文件路径 sep 表示分隔符 index 代表是否导出行序号 header 代表是否导出列序号
to_excel(file_path,sep=' ',index=True,header=True)	file_path 表示文件路径 sep 表示分隔符 index 代表是否导出行序号 header 代表是否导出列序号

5.2.3 数据处理

在数据分析中，数据清洗是在数据价值链中最关键的步骤。数据清洗就是处理缺失数据及清除无意义的信息。对于垃圾数据来说，即使是通过最好的分析，也将产生错误的结果，并误导业务本身。

对缺失值的处理有数据补齐、删除对应行、不处理等几种方法。

```
In [9]:
import pandas as pd
import numpy as np
from pandas import DataFrame
data = {'Tom':[170, 26, 30],'Mike':[175, 25, 28],'Jane':[170, 26,np.nan],'Tim':[175, 25, 28]}
data1 = DataFrame(data).T
data1.drop_duplicates()
data1
```

该段代码的输出结果如下：

```
Out [9]:
          0     1     2
Jane   170.0   26.0   NaN
Mike   175.0   25.0   28.0
Tim    175.0   25.0   28.0
Tom    170.0   26.0   30.0
```

方法一：删除有缺失值的行。

```
In [10]:
data2 = data1.dropna()
data2
```

删除后的结果如下：

```
Out [10]:
           0     1     2
Mike   175.0   25.0   28.0
Tim    175.0   25.0   28.0
Tom    170.0   26.0   30.0
```

通过使用 dropna()方法后，可以看到第 1 行存在缺失值，故被删掉了。

方法二：对缺失值进行填充有很多方法，比较常用的有均值填充、中位数填充、众数填充等。以下采用均值填充。

```
In [11]:
data3 = data1.fillna(data1.mean())
data3
```

结果如下：

```
Out [11]:
           0     1           2
Jane   170.0   26.0   28.666667
Mike   175.0   25.0   28.000000
Tim    175.0   25.0   28.000000
Tom    170.0   26.0   30.000000
```

5.2.4 层次化索引

层次化索引是 Pandas 的一个重要功能，它能使一个轴上有多个（两个以上）索引级别，即它能以低维度形式处理高维度数据。

```
In [12]:
import pandas as pd
import numpy as np
from pandas import Series,DataFrame
data = Series(np.random.randn(10),
              index = [['a','a','a','b','b','b','c','c','d','d'],
                       [1,2,3,1,2,3,1,2,2,3]])
data
Out[12]:
a  1   -0.088594
   2    0.316611
   3    1.383978
b  1    0.215510
   2   -0.111913
   3   -0.580355
c  1   -0.048050
   2   -0.054285
d  2   -0.136860
   3   -1.578472
dtype: float64
```

这就是带有多重索引的 Series 格式化输出。下面看一下它的索引：

```
In [13]:
data.index
Out[13]:
MultiIndex(levels = [['a', 'b', 'c', 'd'], [1, 2, 3]],
           labels = [[0, 0, 0, 1, 1, 1, 2, 2, 3, 3], [0, 1, 2, 0, 1, 2, 0, 1, 1, 2]])
```

对于一个层次化索引的对象来说，选取一个数据集很简单。

```
In [14]:
data['b']
Out[14]:
1    0.215510
2   -0.111913
3   -0.580355
In [15]:
data['b':'c']
Out[15]:
b  1    0.215510
   2   -0.111913
   3   -0.580355
c  1   -0.048050
   2   -0.054285
dtype: float64
In [16]:
```

```
data[['b','d']]
Out[16]:
b  1    0.215510
   2   -0.111913
   3   -0.580355
d  2   -0.136860
   3   -1.578472
dtype: float6
```

甚至还可以在“内层”中进行选取。

```
In [17]:
data[:,2]
Out[17]:
a    0.316611
b   -0.111913
c   -0.054285
d   -0.136860
dtype: float64
```

层次化索引在数据重塑和基于分组的操作中扮演着重要的角色。例如，一个数据可以通过它的 unstack()方法被重新安排到一个 DataFrame 中。

```
In [18]:
data.unstack()
Out[18]:
          1         2         3
a -0.088594  0.316611  1.383978
b  0.215510 -0.111913 -0.580355
c -0.048050 -0.054285       NaN
d       NaN -0.136860 -1.578472
```

对于一个 DataFrame 对象来说，每条轴都可以有分层索引。

```
In [19]:
df = DataFrame(np.arange(12).reshape((4,3)),
                index = [['a','a','b','b'],[1,2,1,2]],
                        columns = [['Ohio','Ohio','Colorado'],
                                   ['green','red','green']])
Df
Out[19]:
     Ohio     Colorado
    green red    green
a 1     0   1        2
  2     3   4        5
b 1     6   7        8
  2     9  10       11
```

各层都可以有名字(字符串或其他 Python 对象)。如果指定名称，它就会显示在控制台输出中。

```
In [20]:
df.index.names = ['key1','key2']
df.columns.names = ['state','color']
df
Out[20]:
state       Ohio     Colorado
color      green red    green
key1 key2
a    1         0   1         2
     2         3   4         5
b    1         6   7         8
     2         9  10        11
```

由于有了分部的索引，所以可以很轻松地选取列分组。

```
In [21]:
df['Ohio']
Out[21]:
color       green  red
key1 key2
a    1          0    1
     2          3    4
b    1          6    7
     2          9   10
```

5.2.5 分级顺序

1. 重新分级排序

有时需要重新调整某条轴上各级别的顺序，或者根据指定级别的值对数据进行重新排序。Swaplevel()接收两个级别的编号或名称，并返回一个互换级别的新对象（但数据不会发生变化）。

```
In [22]:
df.swaplevel('key1','key2')
Out[22]:
state       Ohio     Colorado
color      green red    green
key2 key1
1    a         0   1         2
2    a         3   4         5
1    b         6   7         8
2    b         9  10        11
```

在交换级别时经常会用到 sortlevel()，它根据单个级别中的值对数据进行排序，这样最终结果就是有序的了。

```
In [23]:
df.sortlevel(1)
Out[23]:
```

```
state       Ohio      Colorado
color      green red     green
key1 key2
a     1         0   1         2
b     1         6   7         8
a     2         3   4         5
b     2         9  10        11
```

2. 根据级别汇总统计

许多对 DataFrame 和 Series 的描述和汇总统计都有一个 level 选项，它用于指定在某条轴上求和的级别。

```
In [24]:
df.sum(level = 'key2')
Out[24]:
state  Ohio      Colorado
color green red     green
key2
1          6   8        10
2         12  14        16
```

5.2.6　使用 DataFrame 的列

有时希望将 DataFrame 的一个或多个列索引当成行用，或者将 DataFrame 的行索引变成列。

```
In [25]:
Df = DataFrame({'a':range(7),'b':range(7,0, -1),
                'c':['one','one','one','two','two','two','two'],
                'd':[0,1,2,0,1,2,3]})
Df
Out[25]:
   a  b    c  d
0  0  7  one  0
1  1  6  one  1
2  2  5  one  2
3  3  4  two  0
4  4  3  two  1
5  5  2  two  2
6  6  1  two  3
```

DataFrame 的 set_index()方法会将一个或多个列转换为行索引，并创建一个新的 DataFrame。

```
In [26]:
df1 = df.set_index(['c','d'])
df1
Out[26]:
```

```
        a  b
c   d
one 0   0  7
    1   1  6
    2   2  5
two 0   3  4
    1   4  3
    2   5  2
    3   6  1
```

DataFrame 的 reset_index()方法会将层次化索引的级别转移到列里面去。

```
In [27]:
df1.reset_index()
Out[27]:
     c  d  a  b
0  one  0  0  7
1  one  1  1  6
2  one  2  2  5
3  two  0  3  4
4  two  1  4  3
5  two  2  5  2
6  two  3  6  1
```

5.3 Matplotlib

Matplotlib 可以通过绘图帮助用户找出异常值，进行必要的数据转换，得出有关模型的 idea 等，其是 Python 数据分析重要的可视化工具。

5.3.1 figure 和 subplot

Matplotlib 的图像都位于 figure 中，可以用 plt.figure()创建一个新的 figure。

```
In [28]:
import matplotlib.pyplot as plt
fig = plt.figure()          # 创建一个新的 figure，会弹出一个空窗口
```

plt.figure()的一些选项，特别是 figuresize，可以确保图片保存到磁盘上时具有一定的大小和纵横比。plt.gcf()可得到当前 figure 的引用，必须用 add_subplot()创建一个或多个 subplot 才可以绘图。

```
In [29]:
ax1 = fig.add_subplot(2,2,1)
```

以上代码的意思是该图像是 2×2 的（即有 4 个 subplot），且当前选中的是 4 个 subplot 中的第一个（编号从 1 开始）。如果要把后面的也创建并显示出来，那么可以用如下代码。

```
In [30]:
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
ax4 = fig.add_subplot(2,2,4)
```

这几行代码运行的结果如图 5.2 所示。

```
Out[30]:
```

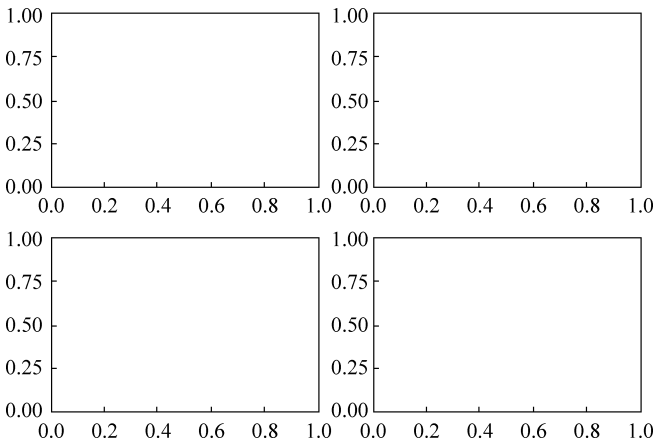

图 5.2　有 4 个 subplot 的 figure

这时如果执行一条绘图命令 plt.plot([ ]),Matplotlib 就会在最后一个用过的 subplot(没有则创建一个)上进行绘制。因此,执行下列代码可以得到如图 5.3 所示的结果。

```
In [31]:
from numpy.random import randn
plt.plot(randn(50).cumsum(), 'k--')
Out[31]:
```

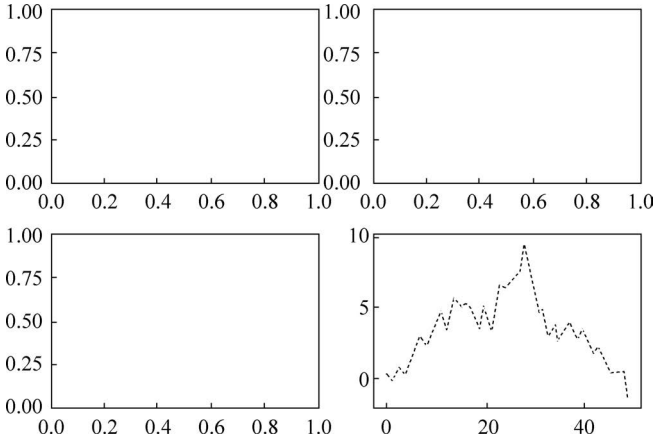

图 5.3　进行绘制操作后的图

'k--'是一个线型选项，用于告诉 Matplotlib 绘制黑色虚线图。前面那些由 fig.add_subplot() 返回的是 AxesSubplot 对象，直接调用其实例方法就可以在其他空着的格子里面绘图。

```
In [32]:
import numpy as np
ax1.hist(randn(100),bins = 20,color = 'k', alpha = 0.3)
ax2.scatter(np.arange(30),np.arange(30) + 3 * randn(30))
```

结果如图 5.4 所示。

```
Out[32]:
```

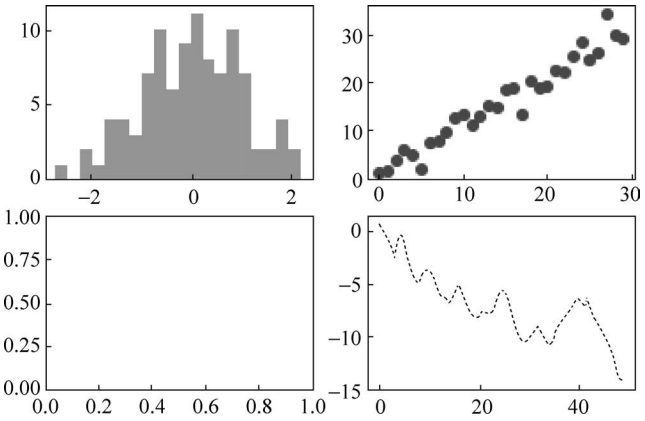

图 5.4　连续绘制后的图

可以在 Matplotlib 中找到各种图标类型。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 [33]:
fig, axes = plt.subplots(2,3)
axes
```

输出结果如下：

```
Out[33]:
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000000012486278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000013EFF780>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000000161A67B8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000000161FF588>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000016265AC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000000162BE400>]],
      dtype = object)
```

这是非常实用的，因为可以轻松地对 axes 数组进行索引，就好像一个二维数组一样，例如 axes[0,1]。用户还可以通过 sharex 和 sharey 指定 subplot 应该具有相同的 X 轴或 Y 轴。在比较相同范围内的数据时，这也是非常实用的，否则 Matplotlib 会自动缩放各

图表的界限。关于 subplots 的更多信息如表 5.4 所示。

表 5.4　pyplot.subplots()的参数

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 会影响所有的 subplot)
sharey	所有 subplot 应该使用相同的 Y 轴刻度(调节 ylim 会影响所有的 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

5.3.2　调整 subplot 周围的间距

在默认情况下,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。间距与图像的高度和宽度有关,因此,如果调整了图像大小,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,代码如下:

```
In [34]:
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)
```

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,在这个例子中将间距收缩到 0,如图 5.5 所示。

```
Out[34]:
```

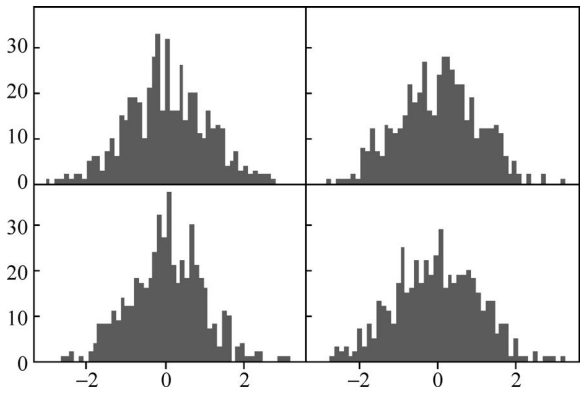

图 5.5　各 subplot 之间没有间距

由图 5.5 不难看出其中的轴标签重叠了。Matplotlib 不会检查轴标签是否重叠,所以对于这种情况来说,用户只能自己设定刻度位置和刻度标签。

5.3.3 颜色、注释和线型

Matplotlib 的 plot()函数可以接收一组(x,y)坐标及表示颜色和线型的字符串缩写。常用的颜色都有一个缩写词，如果要使用其他颜色，可以使用指定其 RGB 值的方式。例如，要根据 x 和 y 绘制红色虚线，可以执行如下代码：

```
In [35]:
plt.plot(x,y,'r -- ')
```

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

```
In [36]:
plt.plot(x,y,linestyle = ' -- ', color = 'r')
```

在 Matplotlib 绘制的图形中可以添加两类注释：指向性注释和无指向性注释。用一个箭头指向要注释的地方，再写上一段话的行为，称为指向性注释。Matplotlib 使用函数 plt.annotate()来实现这个功能，而无指向性注释使用 text()函数实现。annotate()函数的语法结构如下：

```
plt.annotation(s, xy, xytext = None, xycoords = 'data', textcoords = None, arrowprops = None,
annotation_clip = None, ** kwargs)
```

主要参数解释如下。

s：字符串，注释信息内容。

xy：(float,float)，箭头点所在的坐标位置。

xytext：(float,float)，注释内容的坐标位置。

xycoords：被注释点的坐标系属性(xycoords 的值为 data，以被注释的坐标点 xy 为参考)。

textcoords：设置注释文本的坐标系属性(textcoords 选择为相对于被注释点 xy 的偏移量)。

arrowprops：dict，设置指向箭头的参数（arrowstyle：设置箭头的样式；color：设置箭头的颜色；connectionstyle：设置箭头的形状为直线或曲线)。

无指向性的注释文本使用 matplotlib.pyplot.text()函数进行添加，该函数会在图中指定的位置添加注释内容而无指向箭头。函数的语法结构如下：

```
plt.text(x,y,s,family,fontsize,style,color, ** kwargs)
```

主要参数解释如下。

x,y：代表注释内容位置。

s：代表注释文本内容。

family：设置字体，自带的可选项有{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}。

fontsize：字体大小。

style：设置字体样式，可选项有{'normal'，'italic'(斜体)，'oblique'(斜体)}。

下面实例使用柱状图展示我国 Top5 城市富裕家庭数量分布并用 text()函数标注家庭数量。

```
import matplotlib.pyplot as plt
import numpy as np
# 构建数据
Y2020 = [15600,12700,11300,4270,3620]
Y2021 = [17400,14800,12000,5200,4020]
cities = ['北京','上海','香港','深圳','广州']
bar_width = 0.4
half_bar_width = 0.2
# 中文乱码的处理
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False
# 绘图
plt.bar(np.arange(5) - half_bar_width, Y2020, label = '2020', color = 'royalblue', alpha
= 0.8, width = bar_width)
plt.bar(np.arange(5) + half_bar_width, Y2021, label = '2021', color = 'goldenrod', alpha
= 0.8, width = bar_width)
plt.xlabel('Top5 城市')
plt.ylabel('家庭数量')
plt.title('财富家庭数 Top5 城市分布')
plt.xticks(np.arange(5),cities)
# 为每个条形图添加数值标签
for x2020,y2020 in enumerate(Y2020):
    plt.text(x2020 - bar_width, y2020 + 100, '%s' % y2020)
for x2021,y2021 in enumerate(Y2021):
    plt.text(x2021, y2021 + 100, '%s' % y2021)
plt.legend()
plt.show()
```

上述代码的执行结果如图 5.6 所示。

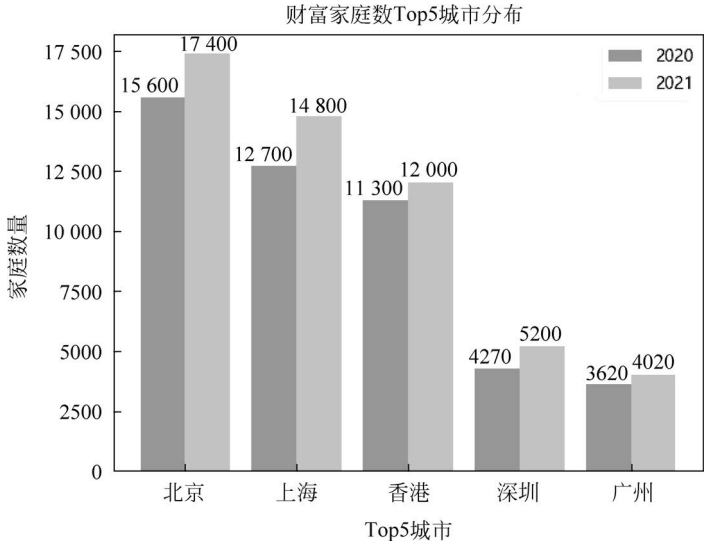

图 5.6　我国 Top5 城市富裕家庭数分布柱状图

5.3.4 刻度标签和图例

对于大多数的图表装饰项而言，其实现方式主要有两种，即使用过程型的 pyplot 接口和更为面向对象的原生 Matplotlib API。设计 pyplot 接口的目的就是实现交互式作用，它含有诸如 xlim()、xticks()和 xticklabels()之类的方法，分别控制图表的范围、刻度位置和刻度标签等。其使用方式有以下两种。

（1）调用时不带参数，则返回当前的参数值。例如，plt.xlim()返回当前 X 轴的绘图范围。

（2）调用时带参数，则设置参数。例如，plt.xlim([0,100])会将 X 轴的范围设置为 0～100。

这些方法都是对当前或最近创建的 AxesSubplot 起作用，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()。为了说明轴的自定义，创建一个简单的图像并绘制一段随机漫步图例，如图 5.7 所示。

```
In [37]:
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.plot(randn(1000).cumsum())
Out[37]:
```

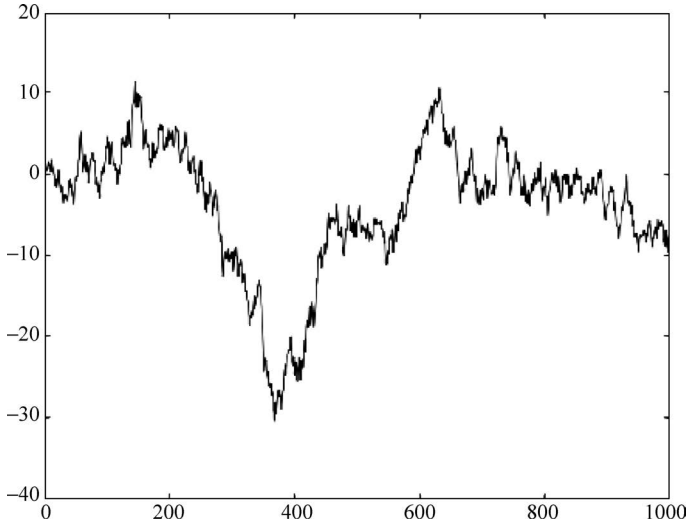

图 5.7 随机漫步图例

如果要修改 X 轴刻度，最简单的办法就是使用 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，在默认情况下这些位置也就是刻度标签。

5.3.5 添加图例

图例(legend)是另外一种用于表示图标元素的重要工具。添加图例的最简单方式，

就是在添加 subplot 时传入 label 参数。

```
In [38]:
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.plot(randn(1000).cumsum(),'k',label = 'one')
```

当需要对图中的线进行注解时，可用下面这样的代码添加图例。

```
In [39]:
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.plot(randn(1000).cumsum(),'k',label = 'one')
ax.plot(randn(1000).cumsum(),'k--',label = 'two')
ax.plot(randn(1000).cumsum(),'k.',label = 'three')
ax.legend(loc = 'best')
```

这几行代码得到的效果如图 5.8 所示。用户可以通过 loc 参数来指定图例所在的位置，'best'表示它会自动找一个最佳位置。

```
Out[39]:
```

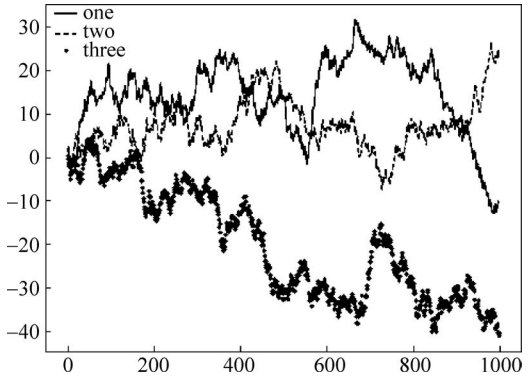

图 5.8　在最佳位置添加图例

5.3.6　将图表保存到文件

利用 plt.savefig()方法可以将当前图表保存到文件。该方法相当于 figure 对象的 savefig()实例方法。例如，要将图表保存为 SVG 格式文件，需要用如下代码：

```
In [40]:
plt.savefig('figpath.svg')
```

文件类型是通过文件扩展名推断出来的。因此，如果用户使用的是.jpg，就会得到一个 JPG 格式的文件。在发布图片时最常用到的两个选项是 dpi(控制“每英寸点数”)和 bbox_inches(剪除当前图表周围的空白部分)。如果用户想得到一个指定分辨率的文件，

可以用下面的语句：

```
In [41]:
plt.savefig('figpath.svg', dpi = xxx, bbox_inches = 'tight')
```

dpi 表示想要得到的分辨率，bbox_inches＝'tight'表示得到的图片带有最小的白边。figure.savefig()方法的参数说明如表 5.5 所示。

表 5.5　figure.savefig()方法的参数说明

参　数	说　明
fname	含有文件路径的字符串，或者 Python 的文件型对象，图像格式由文件扩展名推断出
dpi	图像的分辨率（每英寸点数），默认等于 100
facecolor	图像的背景颜色，默认为白色
edgecolor	图像四周的颜色，默认为白色
format	设置文件格式，例如 png、pdf、svg、jpg
bbox_inches	图像需要保存的部分。如果设置为'tight'，则会尝试剪掉图像周围的空白部分

本章小结

本章介绍 Python 数据分析的常用库：NumPy 数值计算库是数据分析的基础，它将数据转换为数组进行计算；Pandas 是 Python 数据分析的标准库，里面包含了很多数据分析的工具；Matplotlib 是将数据可视化的库，可以让用户对数据有一个更加直观、清晰的认识。

扫一扫

自测题

本章习题

1. 创建一个长度为 10 的一维全为 0 的 ndarray 对象，然后让第 3 个元素等于 5。

2. 创建一个包含 0 到 15 的一维 NumPy 数组，然后将其重塑为一个 4×4 的二维数组，并计算每列的平均值。

3. 创建一个 1×100 的随机数组，计算数组的最大值、最小值和标准差。

4. 创建一个 6×6 的二维数组，其中包含从 0 到 35 的整数。然后，提取数组中所有位于奇数行和偶数列的元素，并将这些元素重新组成一个新的数组。

5. 创建一个 8×8 的二维数组，其中包含从 1 到 64 的整数。然后，执行以下切片操作：

（1）提取数组的左上角 3×3 子数组。

（2）提取数组的右下角 3×3 子数组。

（3）提取数组的中心 4×4 子数组。

6. 创建一个 Pandas Series 对象，包含从 1 到 10 的整数。然后，执行以下操作：

(1) 计算所有元素的平方并存储在一个新的 Series 中。

(2) 过滤出所有大于 50 的元素。

(3) 计算原始 Series 中所有元素的总和。

7. 创建一个包含表 5.6 中数据的 Pandas DataFrame，然后，打印 DataFrame 的基本信息和统计摘要。

表 5.6　第 5 章习题 7 的数据

Name	Age	City
Alice	24	New York
Bob	30	Los Angeles
Charlie	22	Chicago
David	35	Miami
Eva	29	Seattle

8. 若有一个 CSV 文件，名为 student.csv，包含有学生的姓名、学号、年龄、性别等信息，使用 pandas 库从该文件中读取数据，并过滤出所有年龄大于 25 岁的学生信息。

9. 使用表 5.7 中的数据创建一个 DataFrame，将 "Salary" 列中的美元符号和逗号去除，并将其转换为数值类型。

表 5.7　第 5 章习题 9 的数据

Name	Salary
Alice	$ 60000
Bob	$ 75000
Charlie	$ 50000
David	$ 85000
Eva	$ 70000

10. 使用表 5.8 中的数据创建一个 DataFrame，查找并填充 "Age" 列中的缺失值，使用平均值填充。

表 5.8　第 5 章习题 10 的数据

Name	Age	City
Alice	24	New York
Bob	NaN	Los Angeles
Charlie	22	Chicago
David	35	Miami
Eva	NaN	Seattle