第 5 章

数据分析与可视化

本章重点介绍 NumPy(Numerical Python)、Pandas 和 Matplotlib 三个 Python 扩展库，如图 5-1 所示。NumPy 是科学计算的核心库。Pandas 是基于 NumPy 的数据分析工具。Matplotlib 是一款优秀的数据可视化 Python 第三方库。

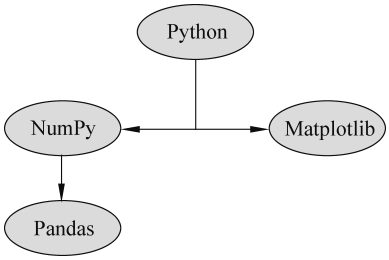

图 5-1　三个扩展库以及 Python 语言之间的关系

5.1　NumPy

NumPy 支持多维数组 ndarray 与矩阵运算，功能涵盖了随机数生成、线性代数等。NumPy 的首要任务是处理多维数组。多维数组是一个元素表，其所有元素都属于同种数据类型。

```
[[ 1., 0., 0.],
 [ 0., 1., 2.]]
```

NumPy 的维度被称为轴，上述数组由两个轴构成，第 1 个轴的长度等于 2，第 2 个轴的长度等于 3，因此这是一个 2 行 3 列的数组。ndarray 对象的主要属性如表 5-1 所示。

表 5-1　ndarray 对象的主要属性

属　　性	说　　明
ndarray.ndim	数组的维数(轴数)
ndarray.shape	数组的形状
ndarray.size	数组中元素的总数

续表

属　性	说　明
ndarray.dtype	数组中元素的数据类型
ndarray.itemsize	数组中每个元素所占内存的字节数
ndarray.data	容纳数组元素的缓冲区，通常不使用此属性

```
>>> a = np.array([[0, 1, 2],                    #数组类 ndarray 的别名 array
                  [3, 4, 5]])                   #2 行 3 列的多维数组
>>> a.ndim                                      #数组 a 的维数
2
>>> a.shape                                     #数组 a 的形状
(2, 3)
>>> a.size                                      #数组 a 的元素总数
6
>>> a.dtype                                     #数组 a 中元素的数据类型
dtype('int32')
>>> a.itemsize                                  #元素所占内存的字节数
4
```

获取函数、模块等的帮助信息，可使用 numpy.info()函数，也可以使用 help()函数。

```
>>> np.info(np.ndarray)                         #输出结果省略
>>> help(np.random)                             #输出结果省略
```

使用 dir()函数查看模块中包含的所有成员(属性与方法)，以及支持的操作。

```
>>> dir(numpy)                                  #输出结果省略
```

5.1.1　创建数组

创建 NumPy 数组主要有 4 种方法，分别是 array()、arange()、linspace()和 logspace()。

```
>>> np.array((1.5, 2.0, 3.5))                   #将元组转换为 NumPy 数组
array([1.5, 2. , 3.5])
>>> np.array(1, 2, 3)                           #初学者常犯的错误
>>> np.array([1, 2, 3])                         #写法正确
>>> np.array([(1.5, 1, 2), (2, 3, 4)])          #得到 2 行 3 列 NumPy 二维数组
array([[1.5, 1. , 2. ],
       [2. , 3. , 4. ]])
>>> np.array(range(1, 4))                       #将 range(1, 4)转换为 NumPy 数组
array([1, 2, 3])
>>> np.array([[1, 2], [3, 4]], dtype=np.float32)     #指定元素的数据类型
```

NumPy 提供的 arange()函数，其功能类似于 Python 内置函数 range()。

```
>>> np.arange(10, 30, 5)                        #生成[10, 30)范围内步长为 5 的等差数组
array([10, 15, 20, 25])
>>> np.arange(0, 2, 0.3)                        #步长为浮点数，这点与 range()函数不同
```

```
array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
>>> np.arange(6).reshape(3, 2)          #得到一个 3 行 2 列的二维数组
array([[0, 1],
       [2, 3],
       [4, 5]])
```

使用 linspace()[①]函数和 logspace()函数分别创建等差和等比数组。

```
>>> np.linspace(0, 1, 5)               #生成[0, 1]范围内 5 个浮点数组成的等差数组
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
>>> np.logspace(0, 1, 5).round(3)      #生成由 5 个浮点数组成的等比数组
array([ 1.   ,  1.778,  3.162,  5.623, 10.   ])
```

上述代码中等比数组的 5 个浮点数,分别等于 10 的 0 次方、0.25 次方、0.5 次方、0.75 次方和 1 次方。

3 个特殊函数 zeros()、ones()和 empty(),分别生成全零数组、全 1 数组和空数组。

```
>>> np.zeros((2, 3))                   #生成 2 行 3 列的全零数组
array([[0., 0., 0.],
       [0., 0., 0.]])
>>> np.empty((2, 3))                   #生成 2 行 3 列的空数组,只申请内存空间
array([[0., 0., 0.],                   #不执行初始化操作,其元素值不确定
       [0., 0., 0.]])
```

5.1.2 算术运算与线性代数

数与数组、数组与数组之间的算术运算[②],是在元素级别上进行的。

```
>>> a = np.array([3, 4, 5])
>>> b = np.array([1, 2, 3])
>>> a * 2
array([6, 8, 10])
>>> b > 2
array([False, False, True])
```

使用@运算符(Python 版本号在 3.5 及以上)或 dot()函数实现通常意义上的矩阵乘法。

```
>>>a = np.array([[1, 1],
                 [0, 1]])
>>>b = np.array([[2, 0],
                 [3, 4]])
>>>a * b                              #只是对应位置上的元素相乘
array([[2, 0],
       [0, 4]])
>>>a @ b                              #通常意义上的矩阵乘法,等价于 a.dot(b)
```

① linspace 指线性空间(Linear Space)。

② 算术运算包括加法、减法、乘法、除法、幂等。

```
array([[5, 4],
       [3, 4]])
```

通过指定轴参数 axis，使计算操作作用在指定的轴上，如图 5-2 所示。

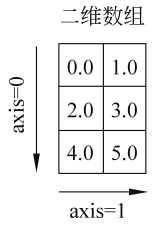

图 5-2　二维数组

```
>>> a = np.array([[0, 1],
                  [2, 3],
                  [4, 5]])
>>> a.mean(axis=None)                    #数组所有元素的平均值,等价于 a.mean()
2.5
>>> a.mean(axis=0)                       #垂直于 x 轴的方向上求数组 a 的平均值
array([2., 3.])
>>> a.mean(axis=1)                       #垂直于 y 轴的方向上求数组 a 的平均值
array([0.5, 2.5, 4.5])
>>> a.std().round(3)                     #标准差(Standard Deviation)
1.708
>>> a = np.array([[1, 4, 8],
                  [5, 1, 4],
                  [0, 7, 9]])
>>> np.sort(a, axis=None)                #先将数组扁平化,然后再排序
array([0, 1, 1, 4, 4, 5, 7, 8, 9])       #原数组不变,即不是原地排序(In-place)
>>> a.sort()                             #原地排序,数组被改变,等价于 a.sort(axis=1)
>>> a
array([[1, 4, 8],
       [1, 4, 5],
       [0, 7, 9]])
>>> a.cumsum(axis=1)                     #垂直于 y 轴的方向上求累加和
array([[1, 5, 13],                       #累加的 cumulative
       [ 1, 5, 10],
       [ 0, 7, 16]], dtype=int32)
```

可使用 average()函数计算数组的加权平均。

```
>>> a = np.array([[0, 1, 2],
                  [3, 4, 5]])
>>> weight = [0.3, 0.7]                  #权重数组
>>> np.average(a, axis=0, weights=weight)
array([2.1, 3.1, 4.1])
```

```
>>> a = np.array([[1., 2.],
                  [3., 4.]])
>>> a.transpose()                          #矩阵的转置(Transpose),等价于 a.T
array([[1., 3.],
       [2., 4.]])
>>> np.diagonal(a)                         #矩阵 a 的主对角线(Diagonal)
array([1., 4.])
>>> np.linalg.inv(a)                       #求逆(Inverse)矩阵
array([[-2. ,  1. ],                       #linalg =linear algebra 线性代数
       [ 1.5, -0.5]])
```

5.1.3 通用函数

NumPy 提供的通用函数包括 sin()、cos()、exp()等。这些函数的操作对象是数组中的元素而不是数组本身,而且生成一个新数组作为输出。

```
>>> a = np.array( [0, 1, 2] )
>>> np.exp(a).round(3)                     #求 e 的幂,指数为数组元素
array([1.   , 2.718, 7.389])
>>> a = np.random.rand(1, 3).round(3)      #生成 1 行 3 列[0, 1)内的随机数数组
>>> a
array([ [0.044, 0.572, 0.454] ])
>>> np.sin(a).round(3)
array([[0.044, 0.541, 0.439],
       [0.749, 0.107, 0.358]])
```

Python 语言的内置模块 random 有一个 randint()函数。

```
>>> import random
>>> a = random.randint(1, 5)               #生成[1, 5]范围内的随机整数
>>> a
4
>>> b = np.random.randint(1, 5)            #生成[1, 5)范围内的随机整数,不包括 5
>>> b
1
```

尽量使用 np.random.randint()函数,而不是 random.randint()函数,因为前者的效率更高。

函数 random.random()、np.random.random()、np.random.rand()和 np.random.randint()都能生成随机数。random()是内置模块 random 的一个函数,可生成[0, 1)范围内的浮点数。

```
>>> random.random()
0.2611025605918177
```

函数 np.random.random()与 random.random()的功能相同。

np.random.rand()函数可创建具有指定形状的、[0, 1]范围内的随机数数组。

```
>>> np.random.rand(3).round(3)          #创建 1 行 3 列的数组
array([0.417, 0.9  , 0.576])
>>> np.random.rand(1, 2)                #创建 1 行 2 列的数组
array([ [0.83704391, 0.15624938] ])
```

np.random.randint()函数可创建具有指定形状的随机整数数组。

```
>>> np.random.randint(5, size=(2,4))    #生成 2 行 4 列[0, 5)范围内的整数数组
array([[4, 3, 0, 3],
       [3, 2, 1, 4]])
>>> np.random.randint(1, [3, 5])        #生成 1 行 2 列的整数数组
array([1, 2])                           #两个元素分别来自[1,3)和[1,5)
>>> np.random.randint([5, 7], 10)       #生成 1 行 2 列的整数数组
array([5, 8])                           #两个元素分别来自[5,10)和[7,10)
```

NumPy 的常用函数还包括 add()、sqrt()、var()、np.random.choice()、square()、sign()、log()、floor()、argmax()、argmin()等。

5.1.4　索引、切片和迭代

类似于 Python 语言的字符串、列表和元组，一维数组也可以被索引、切片和迭代。

```
>>> a = np.array([5, 6, 7, 8, 9])
>>> a[2]
7
>>> a[2:4]
array([7, 8])
>>> a[::-1]                             #将数组 a 逆序输出
array([9, 8, 7, 6, 5])
>>> a                                   #数组 a 本身并没改变
array([5, 6, 7, 8, 9])
>>> for i in a:                         #迭代，输出省略
    print(i ** (1/2))
```

多维数组的每个轴对应一个索引，这些索引之间以逗号分隔。

```
>>> a = np.array([[ 0,  1,  2,  3],
                  [10, 11, 12, 13],
                  [20, 21, 22, 23],
                  [30, 31, 32, 33]])
>>> a[2, 3]                             #得到第 3 行第 4 列的元素
23
>>> a[0:4, 1]                           #得到第 2 列第 1~4 行的所有元素
array([ 1, 11, 21, 31])
>>> a[:, 1]                             #得到第 2 列的所有元素，等价于 a[0:4, 1]
array([ 1, 11, 21, 31])
```

假如数组 a 有 3 个轴，a[-1]等价于 a[-1，:，:]。也就是说，如果只指定前面几个轴，后面的轴不指定的话，那么切片将包含剩余轴上的所有元素，此时也可以用“...”表示剩余轴上

的所有元素。

```
>>> a[-1]                                    #数组 a 的最后一行
array([30, 31, 32, 33])
>>> a[-1, :]                                 #等价于 a[-1]
array([30, 31, 32, 33])
>>> a[-1, ...]                               #等价于 a[-1]和 a[-1, :]
array([30, 31, 32, 33])
```

“...”具有伸缩性,它可自动生成代码所需的任意多个冒号。假如数组 x 有 5 个轴,则:

(1) x[1,2,...]等价于 x[1,2,:,:,:];

(2) x[4,...,5,:]等价于 x[4,:,:,5,:]。

除了通过整数或切片进行索引外,还可以使用整数数组进行索引。

```
>>> a = np.array([0, 1, 4, 9, 16, 25])
>>> i = np.array([1, 1, 5, 3])
>>> a[i]                                     #数组 a 中位置 i 处的元素
array([1, 1, 25, 9])
>>> i = np.array([[3, 4], [2, 3]])           #二维索引数组
>>> a[i]                                     #数组 a[i]与数组 i 的形状相同
array([[ 9, 16],
       [4, 9]])
```

借助于布尔索引可以显式地指定数组中需要的项和不需要的项。

```
>>> a = np.array([[ 0,  1,  2,  3],
                  [ 4,  5,  6,  7],
                  [ 8,  9, 10, 11]])
>>> b = a > 4
>>> b                                        #布尔数组 b 与数组 a 的形状相同
array([[False, False, False, False],
       [False,  True,  True,  True],
       [True,  True,  True,  True]])
>>> a[b]
array([5,  6,  7,  8,  9, 10, 11])           #返回由选定元素组成的一维数组
>>> a[b] = 0                                 #将布尔索引与赋值操作相结合
>>> a
array([[0, 1, 2, 3],
       [4, 0, 0, 0],
       [0, 0, 0, 0]])
```

使用布尔索引指定数组的维度:

```
>>> a = np.array([[ 0,  1,  2,  3],
                  [4,  5,  6,  7],
                  [8,  9,  10, 11]])
>>> b1 = np.array([False, True, True])
>>> a[b1]                                    #选择数组的第 2 行和第 3 行
array([[4,  5,  6,  7],
```

```
        [8,  9, 10, 11]])
>>> b2 = np.array([True, False, True, False])
>>> a[:, b2]                              #选择数组的第 1 列和第 3 列
array([[ 0, 2],
       [ 4, 6],
       [ 8, 10]])
>>> a[b1, b2]                             #得到主对角线上的元素,第 2、3 行与第 1、3 列
array([ 4, 10])
```

在默认情况下,迭代操作是在多维数组的第 1 个轴进行的。

```
>>> a = np.array([[ 0,  1,  2,  3],
                  [10, 11, 12, 13]])
>>> for row in a:
    print(row)
```

上述代码的输出结果:

```
[0 1 2 3]
[10 11 12 13]

>>> a = np.array([[0, 1, 2],
                  [3, 4, 5]])
>>> for ele in a.flat:                    #属性 flat 可将多维数组扁平化为一维数组
    print(ele, end=" ")                   #属性 flat 并不改变原数组的形状
```

上述代码的输出结果:

```
0 1 2 3 4 5
```

5.1.5　形状变换

数组的形状(Shape)由各个轴以及轴上的元素个数决定。

```
>>> a = np.array([[0, 1],
                  [2, 3],
                  [4, 5]])
>>> a.shape
(3, 2)
```

可通过各种命令更改数组的形状[①]。ravel()、reshape()以及转置(.T)都返回修改后的数组,但不改变原数组。

```
>>> a.ravel()                             #返回扁平化的数组
array([0, 1, 2, 3, 4, 5])
>>> a.reshape(2, 3)                       #由 3 行 2 列转换为 2 行 3 列的新数组
array([[0, 1, 2],
       [3, 4, 5]])
```

① 数组的扁平化,既可以使用数组的 flat 属性,也可以使用数组的 ravel()方法。

```
>>> a.T                                    #转置(Transpose)
array([[0, 2, 4],
       [1, 3, 5]])
```

读者自行验证原数组 a 并没有发生改变。resize()方法也可以改变数组的形状,不同之处是它改变数组本身。当改变数组的形状时,如果将某个维度指定为－1,则系统会自动推导该维度的长度。

```
>>> a = np.array([0, 1, 2, 3, 4, 5])
>>> a.reshape(3, -1)                       #根据第 1 维的长度 3,推导出第 2 维的长度 2
array([[0, 1],
       [2, 3],
       [4, 5]])
```

5.1.6 堆叠与分割

两个数组既可以沿着水平方向堆叠在一起,也可以沿着垂直方向堆叠在一起。

```
>>> a = np.array([[4, 3],
                  [9, 3]])
>>> b = np.array([[7, 0],
                  [9, 5]])
>>> np.hstack((a, b))                      #水平堆叠(Horizontal)
array([[4, 3, 7, 0],
       [9, 3, 9, 5]])
>>> np.column_stack((a, b))
array([[4, 3, 7, 0],
       [9, 3, 9, 5]])
>>> np.c_[a, b]                            #c = column 列
array([[4, 3, 7, 0],
       [9, 3, 9, 5]])
>>> np.vstack((a, b))                      #垂直堆叠(Vertical)
array([[4, 3],                             #尝试代码 np.r_[a, b],r = row 行
       [9, 3],                             #尝试代码 np.row_stack((a, b))
       [7, 0],
       [9, 5]])
```

函数 np.hstack()、np.vstack()、np.row_stack()和 np.column_stack(),以及两个 CClass 实例 np.r_与 np.c_的用法非常灵活,读者没有把握时要多实践。

数组能堆叠在一起,也能拆开。水平分割用 hsplit()函数;垂直分割用 vsplit()函数;既水平分割又垂直分割用 array_split()函数。

5.1.7 广播

广播(Broadcasting)用于解决在不同形状的数组之间进行算术运算的问题。在某些约束条件下,较小的数组在较大的数组上“广播”,以使两个数组的形状相互兼容。

```
>>> a = np.array([[0],
```

```
                [1],
                [2]])
>>> b = np.array([0, 1, 2])
>>> a + b
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])
```

上述两个数组的加法，其计算过程如下。

```
array([  [0 + 0, 0 + 1, 0 + 2]
         [1 + 0, 1 + 1, 1 + 2]
         [2 + 0, 2 + 1, 2 + 2]  ])
```

5.2　Pandas

Pandas 拥有 Series 和 DataFrame 两种重要的数据结构，如表 5-2 所示，可用于对噪声数据等进行清洗，以方便后续的数据分析与处理。

表 5-2　Pandas 的两种数据结构

名　称	维　度	说　明
Series	一维	带有标签的同种数据类型组成的一维数组。与 list 和 numpy.array 类似，但是 list 中的元素可以是不同的数据类型；而 array 与 Series 只允许存储相同的数据类型
DataFrame	二维	带有标签的异种数据类型组成的二维数组

5.2.1　Series

1. 创建 Series

创建 Series 的基本语法格式如下。

```
pandas.Series(data=None, index=None, name=None)
```

- data：输入数据，接收 array 或 dict；
- index：索引，接收 array 或 list，必须与数据 data 的长度相同；
- name：Series 对象的名称，接收字符串。

(1) 参数 data 是 NumPy 数组。

```
>>>import pandas as pd
>>>s = pd.Series(np.arange(3), index=['a', 'b', 'c'], name='ndarray')
>>>print(s)
```

上述代码的输出结果：

```
a    0
b    1
c    2
```