第5章

NumPy数据处理基础

扫码观看

学习目标

- 掌握 NumPy 数组的数据结构。
- 掌握常用随机数的生成方法。
- 掌握常用的数组运算与函数。
- 掌握 NumPy 中简单的统计函数。
- 掌握 NumPy 数组在文件中的存取方法。

NumPy 是 Python 的一个开源数值计算第三方库，可用来处理存储和计算各种数组与矩阵，比 Python 的嵌套列表高效很多。NumPy 中提供了 N 维数组对象、矩阵数据类型、随机数生成、广播函数、科学计算工具、线性代数和傅里叶变换等功能。很多 Python 科学计算的库是基于 NumPy 开发的，因此它是 Python 数据分析不可或缺的第三方库。

NumPy 是非标准库，需要提前安装。Anaconda 中集成了这个库中的模块，可以直接使用 Anaconda。也可以不安装 Anaconda。这样需要安装完官方标准的 Python 发行版后再在线或下载安装相关库。

如果采用在线安装，打开 Windows 系统的命令窗口，输入 pip install numpy 命令，完成 NumPy 相关模块的安装。受网络等因素的影响，从 PyPI 官方安装源在线安装可能会失败。这时可以通过参数－i 后面加安装源地址来尝试改用国内的安装源，以提高网络的传输速度。可以通过百度搜索国内 PyPI 镜像安装源地址。例如：

```
pip install numpy - i https://pypi.tuna.tsinghua.edu.cn/simple
```

当网络传输质量不高时，在线安装可能会出现安装中断，导致安装失败。可以下载安装模块后在本地安装。这时需要先到官方网站下载 WHL 文件。然后打开 Windows 系统的命令窗口，进入下载文件所在目录，执行 pip install 文件名.whl 命令。

本章主要介绍 NumPy 的数组结构、数据的准备、常用运算与函数、统计分析函数、数组在文件中的存取等。

5.1 数据结构

NumPy 中主要有多维数组和矩阵两种数据结构。这里只简单介绍多维数组(ndarray)对象。ndarray 是 NumPy 的数组类型，它的所有元素必须具有相同的数据类型。


扫码观看

5.1.1 利用 numpy.array()函数创建数组

NumPy 数组的创建有多种方法。在创建数组或使用 NumPy 模块相关功能之前通常先通过语句 import numpy as np 导入该模块。

利用 numpy. array(object, dtype = None, * , copy = True, order = 'K', subok = False, ndmin=0, like=None)可以根据参数 object 对象来创建数组类型(ndarray)的对象。其中，参数 object 可以是类似于数组的对象(如序列、其他数组等)；参数 dtype 表示数组元素的类型，如果没有显式指定数组的数据类型，array()函数会根据参数 object 对象，为新建的数组推断出一个较为合适的数据类型；参数 copy 用布尔值表示创建的数组是否复制 object 对象中的元素。

参数 order 指定数组的内存布局是按照 C 语言的行优先还是按照 FORTRAN 语言的列优先，可以取{"K"、"A"、"C"、"F"}中的一个，默认值为"K"。如果对象 object 不是数组，则新创建的数组按照 C 语言的行优先布局，除非 order 参数值设置为"F"才按照 Fortran 的列优先布局。如果参数 object 是一个数组，则根据 order 的取值来决定数组的内存布局。"C"表示按照 C 语言的行优先。"F"表示按照 FORTRAN 语言的列优先。order="A"表示如果输入的 object 数组对象中 order 是"F"不是"C"，则 order 按照取值"F"来操作，否则 order 按照取值"C"来操作。如果 order="K"，则保留参数 object 中的 order 方式，否则按照与输入的 object 参数最接近的方式进行内存布局。

形参列表中单独的星号(*)表示其后面的形参必须以关键参数的方式来传递实参。

1. 创建一维数组

```
>>> import numpy as np
>>> list1 = [5,6.5,9,2,3,7.8,5.6,4.9]
>>> arr1 = np.array(list1)
>>> arr1
array([5. , 6.5, 9. , 2. , 3. , 7.8, 5.6, 4.9])
>>>
```

2. 创建二维数组

```
>>> list2 = [[1,2,3,4,5],[6,7,8,9,10]]
>>> arr2 = np.array(list2)
```

```
>>> arr2
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
>>>
```

3. 创建三维或多维数组

```
>>> list3 = [[[1,2],[3,4]],[[5,6],[7,8]]]
>>> arr3 = np.array(list3)
>>> arr3
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
>>>
```

5.1.2 访问数组对象属性


扫码观看

ndarray 类的重要对象属性有 ndim、shape、size、dtype 等。ndarray. ndim 表示数组维度。ndarray. shape 表示数组各维度大小的形状元组。ndarray. size 表示数组元素的总个数，等于 shape 属性元组中各元素的乘积。ndarray. dtype 表示数组中元素的数据类型。

1. 通过 ndim 属性获取数组的维度

```
>>> arr1.ndim
1
>>> arr2.ndim
2
>>> arr3.ndim
3
>>>
```

2. 通过 shape 属性获取表示数组各维度大小的形状元组

```
>>> arr1.shape
(8,)
>>> arr2.shape
(2, 5)
>>> arr3.shape
(2, 2, 2)
>>>
```

3. 通过 size 属性获取数组中元素的总个数

```
>>> arr1.size
8
>>> arr2.size
```

```
10
>>> arr3.size
8
>>>
```

4. 通过 dtype 属性获取数组元素的数据类型

```
>>> arr1.dtype
dtype('float64')
>>> arr2.dtype
dtype('int32')
>>> arr3.dtype
dtype('int32')
>>>
```

扫码观看

5.1.3 数组对象的类型

1. 创建指定类型的数组对象

array()函数通过 dtype 参数创建指定数据类型的数组对象。例如：

```
>>> arr3 = np.array([10,20,30,40],dtype = np.float64)
>>> arr3
array([10., 20., 30., 40.])
>>>
```

其中，参数 dtype 指定了数组对象的类型。虽然原生 Python 中提供了不少数据类型，但 NumPy 需要更多、更精确的数据类型来支持科学计算。可以使用 NumPy 中的 sctypeDict.keys()来查看 NumPy 数组所支持的所有数据类型。读者可以执行下面的两行程序来查看 NumPy 数组支持的数据类型。

```
>>> import numpy as np
>>> np.sctypeDict.keys()
```

常用的类型有 bool、int、float 等后面加数字或下画线、uint 后面加数字或下画线。其中类型后面的数字表示二进制的位数。可以通过字典的值查看每个类型名称对应的实际类，例如：

```
>>> np.sctypeDict["int_"]
<class 'numpy.int32'>
>>>
```

可以看到 NumPy 数组的 int_类型实际上就是 numpy.int32 类型。

为了与 Python 原生的数据类型相区别，bool、int、float、complex、str 等类型名称末尾都加了单下画线_。

另一种描述 NumPy 数据类型的方式是用一个字符来描述，例如，字母 i 表示有符号整数，字母 f 表示浮点型数据。

2. 转换数组的数据类型

通过数组的 astype()方法转换数组的数据类型，得到新数组，原数组保持不变。例如：

```
>>> arr4 = arr2.astype(np.float64)
>>> arr4.dtype
dtype('float64')
>>> arr4
array([[ 1.,  2.,  3.,  4.,  5.],
       [ 6.,  7.,  8.,  9., 10.]])
>>> id(arr4)                          #查看数组 arr4 的开始地址
2253811340592
>>>
>>> arr2.dtype                        #原数组保持不变
dtype('int32')
>>> arr2                              #原数组保持不变
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
>>> id(arr2)                          #查看数组 arr2 的开始地址
2253921152496
>>>
```

5.1.4 创建常用数组


扫码观看

1. 利用 zeros 和 ones 创建指定形状的全 0 或全 1 数组

```
>>> np.zeros(5)
array([0., 0., 0., 0., 0.])
>>> np.zeros((2,3))
array([[0., 0., 0.],
       [0., 0., 0.]])
>>> np.ones((1,2))
array([[1., 1.]])
>>> np.ones((3,4),dtype = np.int16)
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]], dtype = int16)
>>>
```

2. 利用 eye 或 identity 创建单位阵

```
>>> np.eye(4)
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

```
>>> np.identity(4)
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
>>>
```

3. 创建等差数组

NumPy 中的 arange()函数用于创建等差数组对象。它的用法类似于 Python 中的 range 类。调用格式为 numpy.arange([start,] stop[,step,],dtype=None, *,like=None)。其中 start 表示开始值，默认为 0；stop 表示结束值，结果中不包含 stop 本身；step 表示步长，默认为 1；dtype 表示数组元素类型，默认从其他参数推断。start、step、dtype 三个参数可以省略。例如：

```
>>> np.arange(3,20,3)
array([ 3,  6,  9, 12, 15, 18])
>>>
```

Python 中的 range 类只能创建由整数构成的序列对象。而 NumPy 中的 arange()函数还可以创建浮点数类型的数组。例如：

```
>>> np.arange(0.5, 1.8, 0.3)
array([0.5, 0.8, 1.1, 1.4, 1.7])
>>>
```

也可以利用 numpy.linspace(start,stop,num=50,endpoint=True,retstep=False,dtype=None,axis=0)创建包含 num 个元素，且开始值为 start、结束值为 stop 的等差数列。endpoint 表示 stop 值是否作为最后一个元素。例如：

```
>>> np.linspace(0,5,10)
array([0.        , 0.55555556, 1.11111111, 1.66666667, 2.22222222,
       2.77777778, 3.33333333, 3.88888889, 4.44444444, 5.        ])
>>>
```

4. 创建等比数组

可以利用 numpy.logspace(start,stop,num=50,endpoint=True,base=10.0,dtype=None,axis=0)创建开始值为 base 的 start 次幂(base ** start)、结束值为 base 的 stop 次幂(base ** stop)的 num 个数构成的等比数列。endpoint 表示是否包含结束点。例如：

```
>>> np.logspace(0,8,16)
array([1.00000000e+00, 3.41454887e+00, 1.16591440e+01, 3.98107171e+01,
       1.35935639e+02, 4.64158883e+02, 1.58489319e+03, 5.41169527e+03,
       1.84784980e+04, 6.30957344e+04, 2.15443469e+05, 7.35642254e+05,
       2.51188643e+06, 8.57695899e+06, 2.92864456e+07, 1.00000000e+08])
>>>
```

如上例子产生从 10 的 0 次幂到 10 的 8 次幂之间 16 个数组成的等比数列数组。

5. 从字符串创建数组

可以利用 numpy.fromstring(string,dtype=float,count=-1,*,sep,like=None)函数读取字符串中的数据来创建数组。参数 string 表示输入的字符串。参数 dtype 表示数组元素类型。参数 count 是一个整数,表示从数据中最多读取 count 个 dtype 类型的元素构成数组。sep 表示参数 string 字符串中的分隔符。例如:

```
>>> np.fromstring("1,2,5,6,8,9,10",dtype = int,sep = ",")
array([ 1,  2,  5,  6,  8,  9, 10])
>>> np.fromstring("1,2,5,6,8,9,10",dtype = int,count = 2,sep = ",")
array([1, 2])
>>> np.fromstring("1,2,5,6,8,9,10",dtype = int,count = 3,sep = ",")
array([1, 2, 5])
>>>
```

6. 根据函数创建数组

函数 numpy.fromfunction(function,shape,*,dtype=<class 'float'>,like=None,**kwargs)通过在每个坐标点上执行一个指定的函数 function 来构造一个数组,参数 shape 是一个由整数构成的元组,表示各个维度上的元素个数。在坐标点(x,y,z)上,结果数组的元素值为 function(x,y,z)。例如:

```
>>> def func(x):
    return x * 2 + 1

>>> np.fromfunction(func,(5,))
array([1., 3., 5., 7., 9.])
>>>
```

上述代码表示函数 func 的参数 x 从一维坐标的 0、1、2、3、4 五个点上依次取值,根据 func 返回的值来构造一维数组。

参数 function 也可以是 lambda 表达式。例如:

```
>>> np.fromfunction(lambda x,y:x * 2 + y, (3,4))
array([[0., 1., 2., 3.],
       [2., 3., 4., 5.],
       [4., 5., 6., 7.]])
>>>
```

上述代码表示先从第一维为 0、1、2,第二维为 0、1、2、3 构成的 3 行 4 列网格位置坐标中取对应的第一维和第二维坐标分别作为 x 和 y 的值传递给 lambda 表达式,该表达式的返回值分别作为参数网格相应位置上的新值来构造一个二维数组。例如,用 lambda 表达式计算新数组的第 2 行第 3 列时,x 取 2、y 取 3,lambda 表达式返回计算结果 7。

还可以利用 NumPy 中的 asarray、ones_like、zeros_like、empty、empyt_like 等创建 ndarray 数组对象。


扫码观看

5.2 数据准备

5.2.1 随机数的生成

NumPy 中的 random 子模块用于生成各种随机数。NumPy 官方在线参考手册给出了生成各种随机数的详细介绍。下面介绍几种常用的随机数产生方法。

(1) 生成[0.0,1.0)的随机浮点数。

使用 numpy.random.rand()函数生成[0,1)的随机浮点数。例如：

```
>>> import numpy as np
>>> np.random.rand()                    #生成一个[0,1)的随机浮点数
0.8939672908405941
>>> np.random.rand(3)       #生成一个一维、共三个元素的随机浮点数数组,元素位于区间[0,1)
array([0.54350645, 0.92721516, 0.10503672])
>>> np.random.rand(2,3)                 #生成一个 2 行 3 列的二维数组,数组元素位于区间[0,1)
array([[0.72474509, 0.69509932, 0.82310355],
       [0.16464369, 0.18150546, 0.87969788]])
```

numpy.random.random_sample()、numpy.random.random()、numpy.random.ranf()和 numpy.random.sample()四个函数的功能都是返回位于[0.0,1.0)区间的一个随机的浮点数或参数中指定形状的随机浮点数数组。例如：

```
>>> np.random.random_sample()
0.8243760561191865
>>> np.random.random_sample(3)
array([0.75311019, 0.80618575, 0.19259011])
>>> np.random.random_sample((3,4)) #参数中数组的形状必须以元组的形式出现
array([[0.41438636, 0.46512609, 0.14717557, 0.92288745],
       [0.52002313, 0.6405674 , 0.64982451, 0.80266958],
       [0.97429793, 0.2897892 , 0.34625299, 0.34768561]])
>>> np.random.random_sample((2,3,4))
array([[[0.54609391, 0.12412469, 0.29738999, 0.62508189],
        [0.63528904, 0.33195217, 0.38926109, 0.310123  ],
        [0.49667031, 0.70333863, 0.76978682, 0.26887752]],

       [[0.49821545, 0.09668823, 0.66214618, 0.62025478],
        [0.54457072, 0.29458145, 0.40859359, 0.77368304],
        [0.57813389, 0.55179483, 0.08778325, 0.24025623]]])
```

(2) 使用 numpy.random.randn()生成一个具有标准正态分布的随机浮点数样本。

```
>>> np.random.randn(5)                  #生成一个标准正态分布的一维数组随机浮点数
array([ 0.1538501 ,  0.42421551, -0.17355168,  0.09019904, -0.33155756])
>>> np.random.randn(3,4)                #生成一个标准正态分布的 3 行 4 列二维数组随机浮点数
array([[  1.09882567,  0.67002068, -1.84222623,  1.53957494],
       [-0.14725161,  0.14962733,  0.22269968,  0.38329739],
       [  0.66025437,  0.18853493,  0.38823973,  0.98848714]])
```

(3) 使用 numpy.random.randint()函数生成随机整数。

randint()函数形参格式为：randint(low,high=None,size=None,dtype=int)。其功能是生成位于[low,high)区间内、离散均匀分布的、指定个数的整数值。如果 high 值为 None,则生成的位于[0,low)区间内的整数值。例如：

```
>>> np.random.randint(5)            #生成一个[0,5)的随机整数
2
>>> np.random.randint(50,size=5)  #生成个数为5,值在[0,50)的一维整数数组
array([23, 17, 24, 27, 34])
>>> np.random.randint(10,20)        #生成一个位于[10,20)区间的随机整数
18
>>>      #生成一个3行4列的数组,元素的值是[10,50)区间里的随机整数
>>> np.random.randint(10,50,(3,4))
array([[30, 45, 36, 30],
       [16, 41, 18, 44],
       [43, 16, 37, 11]])
```

(4) 使用 np.random.uniform()生成一个均匀分布的随机采样。

调用格式为 uniform(low=0.0,high=1.0,size=None)。其功能是生成在[low,high)区间内均匀分布的 size 个浮点数。返回值是单个标量或多个浮点数组成的数组。例如：

```
>>> np.random.uniform(1,5,3)
array([2.3024662 , 1.00442023, 4.69085634])
>>> np.random.uniform(1,5)
1.348537831590019
>>>
```

(5) 使用 np.random.choice()从数组中随机抽取元素。

调用格式为 choice(a,size=None,replace=True,p=None)。其功能是从给定的一维数组中随机抽取样本。如果 a 为一维数组,从 a 的元素中抽取样本；如果 a 是一个整数,随机样本从 np.arange(a)中抽取。size 值如果为 None,则返回一个随机样本；size 为整数,表示返回样本的个数；size 为整数元组,该元组的元素表示返回的样本在各维度上的数量,如(m,n),则共返回 m*n 个元素。replace=True 表示可以重复取相同元素,否则不可以重复取相同元素。p 是与 a 的形状大小相同的一维数组,其元素值总和为 1,用来规定 a 中每个元素被选取的概率。如果没有指定 p,则 a 中每个元素被选中的概率相同。例如：

```
>>> np.random.choice(a=5, size=3, replace=True)
array([2, 2, 2])
>>> np.random.choice(a=5, size=3, replace=False)
array([4, 1, 3])
>>> direction = ["up", "down", "left", "right"]
>>> np.random.choice(a=direction,size=(2,2),replace=True)
array([['down', 'right'],
       ['up', 'down']], dtype='<U5')
```

```
>>> np.random.choice(a = direction,size = (2,2),replace = False)
array([['left', 'down'],
       ['up', 'right']], dtype = '<U5')
>>> np.random.choice(a = direction,size = 2,replace = False)
array(['down', 'left'], dtype = '<U5')
>>> np.random.choice(a = direction,size = 2,replace = False,p = [0.3, 0.4, 0.2, 0.1])
array(['up', 'left'], dtype = '<U5')
>>>
```

从上述随机数产生的结果可以看出，调用随机数生成函数，每次会得到一个不同的随机数。原因是调用这些函数时，计算机以当前时间为随机数发生器的种子值。每次调用随机数生成函数的时间不同，因此产生的结果看起来是一个随机数。又例如：

```
>>> np.random.randint(100)
15
>>> np.random.randint(100)
64
```

从如上两个例子可以看出，没有提供种子值，则每次产生不同的随机数。

可以通过如下三种方式来设置随机数发生器的种子值，使得相同条件下每次调用随机数生成函数时产生相同的值。

方式 1：使用 numpy.random.seed()函数设置种子值

如果每次调用随机数生成函数前，都通过 numpy.random.seed()传递相同的种子值，则每次执行随机数生成函数时将得到相同的结果。例如：

```
>>> np.random.seed(10)
>>> np.random.randint(100)
9
>>> np.random.seed(10)
>>> np.random.randint(100)
9
```

方式 2：使用 numpy.random.RandomState 类设置种子值

```
>>> rnd = np.random.RandomState(seed = 0)
>>> rnd.rand(2,2)
array([[0.5488135 , 0.71518937],
       [0.60276338, 0.54488318]])
>>> rnd.randint(5,10,size = (2,2))
array([[6, 8],
       [7, 9]])
>>>
```

这种方式会产生一个伪随机数序列。在相同的种子值下，同一次序位置上生成相同的随机数。例如：

```
>>> rnd = np.random.RandomState(seed = 0)
>>> rnd.rand(2,2)                          #第 1 个次序位置
array([[0.5488135 , 0.71518937],
```

```
       [0.60276338, 0.54488318]])
>>> rnd.rand(2,2)                          #第 2 个次序位置
array([[0.4236548 , 0.64589411],
       [0.43758721, 0.891773  ]])
>>> rnd.rand(2,2)                          #第 3 个次序位置
array([[0.96366276, 0.38344152],
       [0.79172504, 0.52889492]])
>>>
>>> rnd = np.random.RandomState(seed = 0)  #用相同的种子重新开始随机数发生器
>>> rnd.rand(2,2)                          #第 1 个次序位置
array([[0.5488135 , 0.71518937],
       [0.60276338, 0.54488318]])
>>> rnd.rand(2,2)                          #第 2 个次序位置
array([[0.4236548 , 0.64589411],
       [0.43758721, 0.891773  ]])
>>> rnd.rand(2,2)                          #第 3 个次序位置
array([[0.96366276, 0.38344152],
       [0.79172504, 0.52889492]])
>>>
```

方式 3：使用 np.random.default_rng()函数设置种子值

这是 NumPy 提供的一种新的随机数种子设置和随机数生成方式，新的程序推荐使用此方式。这里只列举几个简单的例子。这种方式也会产生一个伪随机数序列。在相同的种子值下，同一次序位置上生成相同的随机数。例如：

```
>>> rng = np.random.default_rng(seed = 0)
>>> rng.integers(10,100,size = (2,2))      #第 1 个次序位置
array([[86, 67],
       [56, 34]], dtype = int64)
>>> rng.integers(10,100,size = (2,2))      #第 2 个次序位置
array([[37, 13],
       [16, 11]], dtype = int64)
>>> rng.integers(10,100,size = (2,2))      #第 3 个次序位置
array([[25, 83],
       [68, 92]], dtype = int64)
>>>
>>> rng = np.random.default_rng(seed = 0)  #用相同的种子重新开始随机数发生器
>>> rng.integers(10,100,size = (2,2))      #第 1 个次序位置
array([[86, 67],
       [56, 34]], dtype = int64)
>>> rng.integers(10,100,size = (2,2))      #第 2 个次序位置
array([[37, 13],
       [16, 11]], dtype = int64)
>>> rng.integers(10,100,size = (2,2))      #第 3 个次序位置
array([[25, 83],
       [68, 92]], dtype = int64)
>>>
```

5.2.2 NumPy 数组在文本文件中的存取

1. 将数组写入 txt 文件或 csv 文件

可以利用 numpy.savetxt 将数组保存到以 txt 为扩展名的文本文件中或以 csv 为扩展名的文本文件中。函数定义如下：

```
numpy.savetxt(fname, X, fmt = '%.18e', delimiter = ' ', newline = '\n',
              header = '', footer = '', comments = '# ', encoding = None)
```

各参数简要含义如下：

fname——保存的文件名；

X——一维或二维数组数据；

fmt——输出数据的格式字符串或格式字符串序列；

delimiter——分隔列数据的字符串；

newline——分隔行的字符串；

header——将在文件开头写入的字符串；

footer——将在文件尾写入的字符串；

comments——注释字符串，放在 header 或 footer 字符串前面表示注释；

encoding——表示输出到文件的字符编码。

【例 5.1】 生成 5 行 6 列的数组，数组元素是[0,1)区间里的随机浮点数，屏幕上打印输出该数组，并将该数组保存到文件 array.txt 中。保存到文件中的数组元素保留 5 位小数，同一行中的元素之间以逗号分隔。

程序源代码如下：

```
#example5_1.py
#coding = utf - 8
import numpy as np
#生成 5 行 6 列的数组,数组元素是[0,1)区间里的随机浮点数
a = np.random.rand(5,6)
print(a)
#将数组 a 保存到 array.txt 文件,文件名前可以包含路径名
#fmt = '%0.5f'表示保留 5 位小数
#delimiter = ','指定以逗号作为同一行中元素之间的分隔符
np.savetxt('array.txt',a,fmt = '%0.5f',delimiter = ',')
```

执行程序 example5_1.py，在源程序文件所在目录下生成一个 array.txt 文件。该文件中保存 5 行 6 列浮点数数据，每行的元素之间以逗号分隔。

如果将程序中 np.savetxt('array.txt',a,fmt='%0.5f',delimiter=',')修改为 np.savetxt('array.csv',a,fmt='%0.5f',delimiter=',')，执行程序后，将在源程序文件所在目录下生成一个 array.csv 文件，该文件中保存 5 行 6 列浮点数数据。

2. 从 txt 文件或 csv 文件读取数据构成数组

可以利用 numpy.loadtxt 从 txt 或 csv 文件中读取数据，返回数组。函数定义如下：

```
numpy.loadtxt(fname, dtype = <class 'float'>, comments = '#', delimiter = None, converters =
None, skiprows = 0, usecols = None, unpack = False, ndmin = 0, encoding = 'bytes', max_rows =
None, *, like = None)
```

各参数的简要含义如下：

（1）fname 表示要读取的文件或文件名。

（2）dtype 表示结果数组的数据类型，默认为 float 类型。

（3）comments 表示标记注释的字符串。

（4）delimiter 表示文件中分隔值的字符串。

（5）converters 表示一个将列号映射到函数的字典，字典中的函数将该列字符串解析为所需的值。

（6）skiprows 表示读取数据时跳过的行数。

（7）usecols 表示整数或整数序列，表示需要读取的列号（列编号从 0 开始）。

（8）unpack 表示是否将返回的数组转置，若为 True，返回的数组将被转置，方便将每列数据组成的数组分别赋值给一个单独的变量或作为序列中单独的一个元组。

（9）ndmin 取整数 0、1 或 2，表示返回数组至少具有的维数；如果小于这个维数，则一维轴会被压缩以增加维数。

（10）encoding 表示用于解码文件中字符的编码。

【例 5.2】 读取 array.txt 文件中的数据构成数组，并打印输出。读取指定列的元素，并打印输出。

程序源代码如下：

```
#example5_2.py
#coding = utf - 8
import numpy as np

#读取文本文件的内容,并按照正常的行列成为二维数组的元素
#根据文本文件中元素之间的分隔符指定 delimiter 参数的值
a = np.loadtxt('array.txt',delimiter = ',',dtype = np.float32)
print('文本文件中保存的原始二维数组信息如下:')
print(a)

#使用 unpack = True 得到转置后的数组
b = np.loadtxt('array.txt',delimiter = ',',unpack = True,
               dtype = np.float32)
print('使用 unpack = True 得到转置后的数组如下:')
print(b)
print('以下打印二维数组 b 的每行,也就是文本文件的每列:')
for i in range(len(b)):
    print(b[i])

#读取指定列的值
c1,c3 = np.loadtxt('array.txt',delimiter = ',',unpack = True,
                   usecols = (1,3),dtype = np.float32)
print('以下打印列信息构成的数组:')
```

```
print(c1)
print(c3)
```

程序 example5_2.py 的运行结果如下：

```
文本文件中保存的原始二维数组信息如下:
[[0.35759 0.89229 0.60431 0.08043 0.86266 0.15912]
 [0.07057 0.93454 0.68374 0.89964 0.2887  0.9041 ]
 [0.83064 0.48475 0.61559 0.17935 0.0997  0.97764]
 [0.66439 0.01323 0.23512 0.34811 0.43549 0.9095 ]
 [0.11388 0.14762 0.18302 0.47503 0.35288 0.3873 ]]
使用 unpack = True 得到转置后的数组如下:
[[0.35759 0.07057 0.83064 0.66439 0.11388]
 [0.89229 0.93454 0.48475 0.01323 0.14762]
 [0.60431 0.68374 0.61559 0.23512 0.18302]
 [0.08043 0.89964 0.17935 0.34811 0.47503]
 [0.86266 0.2887  0.0997  0.43549 0.35288]
 [0.15912 0.9041  0.97764 0.9095  0.3873 ]]
以下打印二维数组 b 的每行,也就是文本文件的每列:
[0.35759 0.07057 0.83064 0.66439 0.11388]
[0.89229 0.93454 0.48475 0.01323 0.14762]
[0.60431 0.68374 0.61559 0.23512 0.18302]
[0.08043 0.89964 0.17935 0.34811 0.47503]
[0.86266 0.2887  0.0997  0.43549 0.35288]
[0.15912 0.9041  0.97764 0.9095  0.3873 ]
以下打印列信息构成的数组:
[0.89229 0.93454 0.48475 0.01323 0.14762]
[0.08043 0.89964 0.17935 0.34811 0.47503]
```

只需要将 example5_2.py 程序文件中的 array.txt 文件名改为以 csv 为扩展名的文件名，就可以读取 csv 文件中的数据构成数组。

扫码观看

5.3 常用数组运算与函数

5.3.1 数组的索引

1. 一维数组索引操作

一维数组通过“数组名[索引号]”的方式来提取特定位置上元素的值或重新设置特定位置上元素的值。位置索引值从 0 开始计数。例如：

```
>>> import numpy as np
>>> np.random.seed(1000)
>>> a = np.random.randint(1,100,10)
>>> a
array([52, 88, 72, 65, 95, 93,  2, 62,  1, 90])
>>> a[5]
93
>>> a[6] = a[6] * 15
```

```
>>> a
array([52, 88, 72, 65, 95, 93, 30, 62,  1, 90])
>>>
```

2. 二维数组索引操作

二维数组通过“数组名[i,j]”或“数组名[i][j]”的方式获取第 i 行 j 列元素的值或重新设置第 i 行 j 列元素的值。例如：

```
>>> np.random.seed(1000)
>>> x = np.random.randint(1,100,size = (3,4))
>>> x
array([[52, 88, 72, 65],
       [95, 93,  2, 62],
       [ 1, 90, 46, 41]])
>>> x[2,0]
1
>>> x[2][0]
1
>>> x[2,0] = 5
>>> x[1,2] = x[1,2] * 3
>>> x
array([[52, 88, 72, 65],
       [95, 93,  6, 62],
       [ 5, 90, 46, 41]])
>>>
```

5.3.2 数组的切片


扫码观看

数组切片是选取原数组指定位置上的元素构成一个数组。数组切片是原始数组的视图，数据并不会被复制，即视图上的任何修改都会直接体现到原数组上或基于同一个原数组的其他切片上。

1. 一维数组的切片

数组名[start：end：step]用来进行一维数组的切片，从 start 位置上的数开始(包括 start)，到 end 位置结束(不包括 end)，每次增长的步长为 step。例如：

```
>>> import numpy as np
>>> np.random.seed(1000)
>>> a = np.random.randint(1,100,10)
>>> a
array([52, 88, 72, 65, 95, 93,  2, 62,  1, 90])
>>> a1 = a[2:6]
>>> a1
array([72, 65, 95, 93])
>>> a2 = a[1:8:3]
>>> a2
```

```
array([88, 95, 62])
>>> a2[1] = 99                                  #在 a2 中修改某位置上的值
>>> a2
array([88, 99, 62])
>>> a1                                          #a1 中体现出了修改后的结果
array([72, 65, 99, 93])
>>> a                                           #原数组 a 中体现出了修改后的结果
array([52, 88, 72, 65, 99, 93,  2, 62,  1, 90])
>>>
```

2. 二维数组的切片

数组名[start1：end1：step1，start2：end2：step2]用来进行二维数组的切片。其中start1、end1 和 step1 分别表示数组第一维切片开始位置、结束位置和增长的步长；start2、end2 和 step2 分别表示数组第二维切片开始位置、结束位置和增长的步长。end1 和 end2 位置的元素均不包含在切片结果中。

```
>>> import numpy as np
>>> np.random.seed(1000)
>>> x = np.random.randint(1,100,size = (5,6))
>>> x
array([[52, 88, 72, 65, 95, 93],
       [ 2, 62,  1, 90, 46, 41],
       [93, 92, 37, 61, 43, 59],
       [42, 21, 31, 89, 31, 29],
       [31, 78, 83, 29, 86, 94]])
>>> x1 = x[1:4, 2:5]
>>> x1
array([[ 1, 90, 46],
       [37, 61, 43],
       [31, 89, 31]])
>>> x2 = x[1:4:2, : ]
>>> x2
array([[ 2, 62,  1, 90, 46, 41],
       [42, 21, 31, 89, 31, 29]])
>>> x1[2,0] = 999                               #修改 x1 中的元素值
>>> x1
array([[  1,  90,  46],
       [ 37,  61,  43],
       [999,  89,  31]])
>>> x2                                          #x2 中直接体现了修改的结果
array([[  2,  62,   1,  90,  46,  41],
       [ 42,  21, 999,  89,  31,  29]])
>>> x                                           #原数组 x 中直接体现了修改的结果
array([[ 52,  88,  72,  65,  95,  93],
       [  2,  62,   1,  90,  46,  41],
       [ 93,  92,  37,  61,  43,  59],
       [ 42,  21, 999,  89,  31,  29],
       [ 31,  78,  83,  29,  86,  94]])
>>>
```

数组名[i]可以用来获取第 i 行的所有元素构成新数组，维数比原数组减 1。数组名[:,j]可以用来获取第 j 列的所有元素构成新数组，维数比原数组减 1。

```
>>> x3 = x[1]                                    #取 x 中的第 1 行，得到一维数组
>>> x3
array([ 2, 62,  1, 90, 46, 41])
>>> x4 = x[:,3]                                  #取 x 中的第 3 列，得到一维数组
>>> x4
array([65, 90, 61, 89, 29])
```

5.3.3 改变数组的形状


扫码观看

1. reshape()方法生成指定形状的数组视图

reshape()方法在不改变数组数据的情况下，改变数组形状。

其参数指定新数组的形状。reshape()方法不产生新的数据，只返回数组的一个视图，原数组的形状保持不变。例如：

```
>>> import numpy as np
>>> a = np.arange(1,16)
>>> a
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> b = a.reshape(3,5)
>>> b
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
>>> a                                            #原数组保持不变
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> b[0][0] = 100                                #修改数组 b 中元素的值
>>> b
array([[100,   2,   3,   4,   5],
       [  6,   7,   8,   9,  10],
       [ 11,  12,  13,  14,  15]])
>>> a                                            #数组 a 中也看到了修改的结果
array([100,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
        14,  15])
>>>
```

2. 用 resize()方法直接修改原数组的形状

resize()方法也可以修改数组形状。与 reshape()方法不同，resize()方法会直接修改原数组的形状。例如：

```
>>> import numpy as np
>>> a = np.arange(1,16)
>>> a
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
>>> a.resize(3,5)
>>> a
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
```

3. 通过设置数组的 shape 属性值直接修改原数组的形状

也可以通过设置数组的 shape 属性值达到修改数组形状的目的，该方法也是直接改变现有数组的形状。例如：

```
>>> import numpy as np
>>> a = np.arange(1,16)
>>> a
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>> a.shape = (3,5)
>>> a
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
>>>
```

5.3.4 数组对角线上替换新元素值

用 NumPy 中的 fill_diagonal()函数可以替换任意维数的给定数组主对角线上的元素值。函数的调用格式为 fill_diagonal(a,val,wrap=False)。其中 a 是一个至少为二维的数组，其主对角线将被标量值 val 替换；wrap 表示是否对行数大于列数的数组对角线循环替换，默认为 False。下面列举 fill_diagonal()函数的常用案例。

```
>>> import numpy as np
>>> a = np.arange(9).reshape(3,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> np.fill_diagonal(a, 9)
>>> a
array([[9, 1, 2],
       [3, 9, 5],
       [6, 7, 9]])
>>>
>>> a = np.arange(12).reshape(3,4)
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> np.fill_diagonal(a,100)
>>> a
```

```
array([[100,   1,   2,   3],
       [  4, 100,   6,   7],
       [  8,   9, 100,  11]])
>>>
>>> a = np.arange(12).reshape(4,3)
>>> a
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
>>> np.fill_diagonal(a,100)
>>> a
array([[100,   1,   2],
       [  3, 100,   5],
       [  6,   7, 100],
       [  9,  10,  11]])
>>>
```

读者可以通过 fill_diagonal()函数的帮助文档了解参数 wrap 的用法和非主对角线的元素替换方法。

5.3.5 用 np.newaxis 或 None 插入一个维度

可以用 np.newaxis 或 None 在数组中插入一个新的维度。例如：

```
>>> import numpy as np
>>> a = np.array([1,2,3])                    #创建一个一维数组
>>> a
array([1, 2, 3])
>>> a.shape
(3,)
>>> a_1 = a[:, np.newaxis]                   #通过 np.newaxis 插入一个维度
>>> a_1
array([[1],
       [2],
       [3]])
>>> a_1.shape                                #得到一个 3 行 1 列的二维数组
(3, 1)
>>>
>>> a_2 = a[:, None]                         #也可以使用 None 来插入一个维度
>>> a_2
array([[1],
       [2],
       [3]])
>>> a_2.shape
(3, 1)
>>>
>>> a_3 = a[np.newaxis, : ]                  #也可以在前面插入一个维度
>>> a_3
array([[1, 2, 3]])
>>> a_3.shape                                #得到一个 1 行 3 列的二维数组
```

```
(1, 3)
>>>
>>> a_4 = a[None, : ]
>>> a_4
array([[1, 2, 3]])
>>> a_4.shape
(1, 3)
>>>
```

实际上，np.newaxis 在功能上等价于 None，是 None 的一个别名。例如：

```
>>> type(np.newaxis)
<class 'NoneType'>
>>> None == np.newaxis
True
>>>
```

下面再来看一个利用 np.newaxis 或 None 插入维度的方式从二维数组中取一列构成新的二维数组的例子。

```
>>> Y = np.arange(0,9).reshape(3,3)
>>> Y
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> y_0 = Y[:, 0]
>>> y_0
array([0, 3, 6])
>>> y_0.shape                          #是一个一维数组
(3,)
>>> y_0[:, np.newaxis]                 #通过 np.newaxis 插入一个维度
array([[0],
       [3],
       [6]])
>>> y_0[:, np.newaxis].shape           #获得一个 3 行 1 列的二维数组
(3, 1)
>>>
>>> y_0[np.newaxis, : ]
array([[0, 3, 6]])
>>> y_0[np.newaxis, : ].shape          #获得一个 1 行 3 列的二维数组
(1, 3)
>>>
```

下面来看一下如何直接获取一列或多列组成一个二维数组。

```
>>> Y[:, 0]                            #得到一个一维数组
array([0, 3, 6])
>>> Y[:, [0]]                          #把列号放在一个序列中，得到一个二维数组
array([[0],
       [3],
       [6]])
>>>
```

```
>>> Y[:, [0,2]]                          #也可以把多个列号放在一个序列中
array([[0, 2],
       [3, 5],
       [6, 8]])
>>>
```

5.3.6 数组的基本运算

1. 数组与标量之间的运算

数组与标量之间的运算将直接作用到每个元素。例如：

```
>>> import numpy as np
>>> np.random.seed(500)
>>> a1 = np.random.randint(1,10,size = (3,4))
>>> np.random.seed(1000)
>>> a2 = np.random.randint(1,10,size = (3,4))
>>> a1
array([[8, 2, 2, 9],
       [8, 2, 2, 6],
       [3, 3, 4, 7]])
>>> a2
array([[4, 8, 8, 1],
       [2, 1, 9, 5],
       [5, 5, 3, 9]])
>>> a1 * 2
array([[16,  4,  4, 18],
       [16,  4,  4, 12],
       [ 6,  6,  8, 14]])
>>> a1
array([[8, 2, 2, 9],
       [8, 2, 2, 6],
       [3, 3, 4, 7]])
>>> a1/2
array([[4. , 1. , 1. , 4.5],
       [4. , 1. , 1. , 3. ],
       [1.5, 1.5, 2. , 3.5]])
```

2. 大小相等的数组之间的算术运算

大小相等的数组之间的任何算术运算也会应用到元素级。例如：

```
>>> a1 + a2
array([[12, 10, 10, 10],
       [10,  3, 11, 11],
       [ 8,  8,  7, 16]])
>>>
>>> np.add(a1,a2)                        #通过函数进行计算
array([[12, 10, 10, 10],
       [10,  3, 11, 11],
       [ 8,  8,  7, 16]])
>>> a1                                   #原数组保持不变
```

```
array([[8, 2, 2, 9],
       [8, 2, 2, 6],
       [3, 3, 4, 7]])
>>> a2                                    #原数组保持不变
array([[4, 8, 8, 1],
       [2, 1, 9, 5],
       [5, 5, 3, 9]])
```

3. 数据离散化（分箱）

NumPy 中的 digitize()函数可以对数据进行分箱离散化操作，格式如下：

```
numpy.digitize(x, bins, right = False)
```

其中，参数 x 是一个 NumPy 数组，是待离散化的数据；参数 bins 是一维单调数组，必须是升序或者降序，作为离散化的参考对象；参数 right 表示间隔是包含左边界的值还是包含右边界的值，其默认值为 False 表示不包含右边界的值，而是包含左边界的值。当 bins 中的值按照升序排列，如果 right＝False，则比较 bins[i－1] <＝ x < bins[i]，否则比较 bins[i－1] < x <＝ bins[i]；当 bin 中的值按照降序排列，如果 right＝False，则比较 bins[i－1] > x >＝ bins[i]，否则比较 bins[i－1] >＝ x > bins[i]。也就是当 right＝False 时，比较的区间不包括大的边界；当 right＝True 时，比较的区间包括大的边界。这里的每个区间被称为一个箱子。

返回值为 x 中的每个元素在 bins 中各个分段（箱子）的位置（索引）所构成的数组。例如：

```
>>> import numpy as np
>>> x = np.array([ - 0.1, 0.2, 6.0, 3.0, 4.0, 15])
>>> bins = np.array([0.0, 1.0, 2.5, 4.0, 10.0])
>>> inds = np.digitize(x, bins)
```

由于 right 参数采用默认值 False，并且 bins 中的值按照升序排列，因此 bins 产生了索引为 0 的(－∞，0.0)的区间、索引为 1 的[0.0，1.0)区间、索引为 2 的[1.0，2.5)区间、索引为 3 的[2.5，4.0)区间、索引为 4 的[4.0，10.0)区间、索引为 5 的[10.0，∞)区间。

x 中的第一个元素－0.1 位于索引为 0 的区间中，因此在返回的数组中对应位置的值为 0；第二个元素 0.2 位于索引为 1 的区间中，因此在返回的数组中对应位置的值为 1；其他元素的位置值以此类推。

用如下代码来查看保存了 digitize()函数返回值的变量 inds，它里面的元素分别表示 x 中各元素位于 bins 中的哪个区间。

```
>>> inds
array([0, 1, 4, 3, 4, 5], dtype = int64)
>>>
```

继续上面的例子，为 right 分别赋予不同的值，如下所示：

```
>>> inds = np.digitize(x, bins, right = False)
>>> inds
```

```
array([0, 1, 4, 3, 4, 5], dtype = int64)
>>> inds = np.digitize(x, bins, right = True)
>>> inds
array([0, 1, 4, 3, 3, 5], dtype = int64)
>>>
```

digitize(x,bins,right = False)的执行过程相当于两层的嵌套循环。外层循环顺序遍历数组 x 中的元素,对每取出的一个元素 x[i],内层循环顺序遍历数组 bins,返回 x[i]位于 bins 数组中元素构成的某个区间索引值,比较规则如下:如果 bins 数组中的元素是升序的,且 right=False,那么如果满足 bins[j-1]<=x[i]<bins[j],返回的数组中就在末尾增加 j 作为元素值,然后回到外层循环继续上面的操作;如果 bins 数组元素是降序的,且 right=False,那么如果满足 bins[j-1]>x[i]>= bins[j],返回的数组中就在末尾增加 j 作为元素值。当数组 x 遍历完之后,就返回由 bins 中的索引值 j 组成的数组。

从 NumPy 的 1.10.0 版本开始,NumPy 中的 digitize()函数的功能可以由 searchsorted()函数来实现。searchsorted()函数使用二进制搜索存放这些值的箱子位置,与以前的线性搜索相比,这种方法对于更大数量的箱子具有更好的伸缩性。

numpy.searchsorted(bins,x,side='left',sorter=None)中的参数 bins 和 x 与 digitize()函数中的参数 bins 和 x 的含义相同。参数 side 有 left 和 right 两个可选值,为 left 时表示分段(箱子)不包含左侧值,但包含右侧值;为 right 时表示分段(箱子)不包含右侧值,但包含左侧值。限于篇幅,对参数 sorter 不做详细介绍,有兴趣的读者可以参考官方文档。下面给出几个简单的例子来说明其用法和功能。

```
>>> np.searchsorted([1,2,3,4,5], [3,4.5,6])
array([2, 4, 5], dtype = int64)
>>> np.searchsorted([1,2,3,4,5], [3,4.5,6],side = 'left')
array([2, 4, 5], dtype = int64)
>>> np.searchsorted([1,2,3,4,5], [3,4.5,6],side = 'right')
array([3, 4, 5], dtype = int64)
>>>
```

对于单调递增的 bins 来说,np.digitize(x,bins,right=False)和 np.searchsorted(bins,x,side='right')返回相同的结果。

4. NumPy 数组的常用计算函数

NumPy 数组的常用计算函数及其说明如表 5.1 所示。

表 5.1 NumPy 数组的常用计算函数及其说明

函 数	说 明
add()	将两个数组中对应位置的元素相加
subtract()	将两个数组中对应位置的元素相减
multiply()	将两个数组中对应位置的元素相乘
divide()	将两个数组中对应位置的元素相除

续表

函　数	说　明
maximum()	返回两个数组中对应位置上的最大值作为元素构成新数组
minimum()	返回两个数组中对应位置上的最小值作为元素构成新数组
greater(),greater_equal,less(),less_equal(),equal(),not_equal()	执行元素级的比较运算，最终产生布尔型数组

numpy.sum()和 numpy.average()可以分别求得数组元素的和与均值。numpy.std()和 numpy.var()分别用于求数组的标准差和方差。numpy.max()和 numpy.min()用于求得最大值和最小值。numpy.sort()用于对数组进行排序。例如：

```
>>> import numpy as np
>>> np.random.seed(1000)
>>> x = np.random.randint(1,10,size = (3,4))
>>> x
array([[2, 8, 2, 6],
       [7, 6, 8, 9],
       [9, 3, 3, 7]])
>>> np.sum(x)
70
>>> np.sum(x,axis = 0)                    #每列相加
array([18, 17, 13, 22])
>>> np.sum(x,axis = 1)                    #每行相加
array([18, 30, 22])
>>> np.average(x)
5.833333333333333
>>> np.average(x,axis = 0)                #每列求均值
array([6.      , 5.66666667, 4.33333333, 7.33333333])
>>> np.average(x,axis = 1)                #每行求均值
array([4.5, 7.5, 5.5])
>>>
>>> np.var(x)
6.472222222222222
>>> np.var(x,axis = 0)                    #对每列上的元素求方差
array([8.66666667, 4.22222222, 6.88888889, 1.55555556])
>>> np.var(x,axis = 1)                    #对每行上的元素求方差
array([6.75, 1.25, 6.75])
>>> np.std(x)
2.544056253745625
>>> np.std(x,axis = 0)                    #对每列上的元素求标准差
array([2.94392029, 2.05480467, 2.62466929, 1.24721913])
>>> np.std(x,axis = 1)                    #对每行上的元素求标准差
array([2.59807621, 1.11803399, 2.59807621])
>>>
```

5.3.7 数组的排序


扫码观看

利用 np.sort()函数可以对数组进行排序，根据原数组生成一个排序后的数组，原数组保持不变。例如：

```
>>> import numpy as np
>>> np.random.seed(1000)
>>> x = np.random.randint(1,10,size = (3,4))
>>> x
array([[4, 8, 8, 1],
       [2, 1, 9, 5],
       [5, 5, 3, 9]])
>>> np.sort(x)                          #默认按各行分别排序
array([[1, 4, 8, 8],
       [1, 2, 5, 9],
       [3, 5, 5, 9]])
>>> x                                   #原数组保持不变
array([[4, 8, 8, 1],
       [2, 1, 9, 5],
       [5, 5, 3, 9]])
>>> np.sort(x,axis = 1)                 #按行排序
array([[1, 4, 8, 8],
       [1, 2, 5, 9],
       [3, 5, 5, 9]])
>>> x                                   #原数组保持不变
array([[4, 8, 8, 1],
       [2, 1, 9, 5],
       [5, 5, 3, 9]])
>>> np.sort(x,axis = 0)                 #按列排序
array([[2, 1, 3, 1],
       [4, 5, 8, 5],
       [5, 8, 9, 9]])
>>> np.sort(x,axis = None)              #axis = None 时，按数组展开的元素排序
array([1, 1, 2, 3, 4, 5, 5, 5, 8, 8, 9, 9])
>>>
```

5.3.8 数组的组合


扫码观看

numpy.hstack()和 numpy.vstack()分别用于数组的水平组合和垂直组合。numpy.concatenate()既可以用于水平组合，也可以用于垂直组合。例如：

```
>>> import numpy as np
>>> a = np.arange(6).reshape(2,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5]])
>>> b = a + 10
>>> b
```

```
array([[10, 11, 12],
       [13, 14, 15]])
>>> np.hstack((a,b))                          #水平组合
array([[ 0,  1,  2, 10, 11, 12],
       [ 3,  4,  5, 13, 14, 15]])
>>> np.concatenate((a,b),axis = 1)            #水平组合
array([[ 0,  1,  2, 10, 11, 12],
       [ 3,  4,  5, 13, 14, 15]])
>>>
>>> np.vstack((a,b))                          #垂直组合
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [10, 11, 12],
       [13, 14, 15]])
>>> np.concatenate((a,b),axis = 0)            #垂直组合
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [10, 11, 12],
       [13, 14, 15]])
>>>
```

numpy.column_stack()用于按列组合。numpy.row_stack()用于按行组合。用于一维数组时，column_stack()将每个一维数组作为一列，合并为一个二维数组，而 hstack()将数组合并为一个一维数组。例如：

```
>>> import numpy as np
>>> x = np.arange(4)
>>> y = x + 10
>>> x
array([0, 1, 2, 3])
>>> y
array([10, 11, 12, 13])
>>> np.column_stack((x,y))
array([[ 0, 10],
       [ 1, 11],
       [ 2, 12],
       [ 3, 13]])
>>> np.hstack((x,y))
array([ 0, 1, 2, 3, 10, 11, 12, 13])
>>>
```

用于二维数组时，column_stack()与 hstack()均按列进行组合，两者功能相同。例如：

```
>>> a = np.arange(6).reshape(2,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5]])
>>> b = a + 10
>>> b
```

```
array([[10, 11, 12],
       [13, 14, 15]])
>>> np.column_stack((a,b))
array([[ 0,  1,  2, 10, 11, 12],
       [ 3,  4,  5, 13, 14, 15]])
>>> np.hstack((a,b))
array([[ 0,  1,  2, 10, 11, 12],
       [ 3,  4,  5, 13, 14, 15]])
>>>
```

无论用于一维数组还是用于二维数组，row_stack()和 vstack()均按行进行组合，两者功能相同。例如：

```
>>> x
array([0, 1, 2, 3])
>>> y
array([10, 11, 12, 13])
>>> np.row_stack((x,y))
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13]])
>>> np.vstack((x,y))
array([[0, 1, 2, 3],
       [10, 11, 12, 13]])
>>> np.row_stack((a,b))
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [10, 11, 12],
       [13, 14, 15]])
>>> np.vstack((a,b))
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [10, 11, 12],
       [13, 14, 15]])
>>>
```

numpy.r_[ ]实现行叠加。numpy.c_[ ]实现列叠加。例如：

```
>>> import numpy as np
>>> x = np.arange(4)
>>> y = x + 10
>>> x
array([0, 1, 2, 3])
>>> y
array([10, 11, 12, 13])
>>> np.c_[x,y]                          #注意:这里是方括号
array([[ 0, 10],
       [ 1, 11],
       [ 2, 12],
       [ 3, 13]])
>>> np.r_[x,y]                          #np.r_[]用于一维数组,创建一个一维数组
array([ 0,  1,  2,  3, 10, 11, 12, 13])
```

```
>>>
>>> a = np.arange(6).reshape(2,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5]])
>>> b = a + 10
>>> b
array([[10, 11, 12],
       [13, 14, 15]])
>>> np.c_[a,b]                          #注意:这里是方括号
array([[ 0,  1,  2, 10, 11, 12],
       [ 3,  4,  5, 13, 14, 15]])
>>> np.r_[a,b]                          #np.r_[]用于二维数组,对数组纵向堆叠
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [10, 11, 12],
       [13, 14, 15]])
>>>
```

5.3.9 数组的分割

NumPy 中的 hsplit()、vsplit()、split()和 array_split()函数均可以实现对数组的分割。这些函数将一个数组分割为多个子数组，返回由这些子数组构成的列表。使用这些函数分割得到的子数组和被分割的数组具有相同的维度。

扫码观看

1. numpy.hsplit()和 numpy.vsplit()

numpy.hsplit(ary,indices_or_sections)将数组沿着水平方向分割为相同大小的 sections 个子数组或在 indices 序列的元素值确定的列之前做分割，并由这些子数组构成一个列表。各子数组与原数组具有相同的行数，各子数组的列数之和等于原数组的列数。

numpy.vsplit(ary,indices_or_sections)将数组沿着垂直方向分割为相同大小的 sections 个子数组或在 indices 序列的元素值确定的行之前做分割，并由这些子数组构成列表。例如：

```
>>> import numpy as np
>>> a = np.arange(12).reshape(3,4)
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> np.hsplit(a,2)
[array([[0, 1],
       [4, 5],
       [8, 9]]), array([[ 2,  3],
       [ 6,  7],
       [10, 11]])]
>>> np.hsplit(a,(1,3))                  #在第 1 列和第 3 列之前做分割
[array([[0],
```

```
       [4],
       [8]]), array([[ 1,  2],
       [ 5,  6],
       [ 9, 10]]), array([[ 3],
       [ 7],
       [11]])]
>>> x = np.vsplit(a,3)
>>> x
[array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10, 11]])]
>>> x[0]
array([[0, 1, 2, 3]])
>>> x = np.vsplit(a,[2,])                    #在第 2 行之前做分割
x
[array([[0, 1, 2, 3],
       [4, 5, 6, 7]]), array([[ 8,  9, 10, 11]])]
>>>
```

2. numpy.split()和 numpy.array_split()

numpy.split()和 numpy.array_split()均可以分别进行水平或垂直方向上的分割，并由分割得到的子数组构成一个列表。

numpy.split(ary,indices_or_sections,axis=0)中如果指定切分的份数 sections，则每份上必须可以平均切分，否则会报错。也可以根据 indices 序列元素值指定的索引在每个索引之前做分割。例如：

```
>>> np.split(a,2,axis = 1)                   #横向切为两份
[array([[0, 1],
       [4, 5],
       [8, 9]]), array([[ 2,  3],
       [ 6,  7],
       [10, 11]])]
>>> np.split(a, (3,),axis = 1)               #在第 3 列之前进行分割,切分成两份
[array([[ 0,  1,  2],
       [ 4,  5,  6],
       [ 8,  9, 10]]), array([[ 3],
       [ 7],
       [11]])]
>>> np.split(a,3,axis = 0)                   #纵向分为三份
[array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10, 11]])]
>>> np.split(a, (2,))                        #在第 2 行之前进行分割,切分成两份
[array([[0, 1, 2, 3],
       [4, 5, 6, 7]]), array([[ 8,  9, 10, 11]])]
>>> np.split(a, [1, 3],axis = 1)             #在第 1 列和第 3 列前进行分割
[array([[0],
       [4],
       [8]]), array([[ 1,  2],
       [ 5,  6],
       [ 9, 10]]), array([[ 3],
```

```
       [ 7],
       [11]])]
>>>
```

也可以采用 hsplit()和 vsplit()分别实现上述功能。

numpy.array_split(ary,indices_or_sections,axis=0)中如果指定切分份数,可以不需要保证平均切分。如果不能平均切分且原数组被切分维度上的元素个数为 L,则前 L % n 个子数组大小为 L//n + 1,其余子数组的大小为 L//n。也可以按照 indices 序列元素值指定的索引,在索引位置前做分割。例如:

```
>>> np.array_split(a,2,axis = 0)
[array([[0, 1, 2, 3],
       [4, 5, 6, 7]]), array([[ 8,  9, 10, 11]])]
>>> np.array_split(a,(1,3),axis = 1)      #在第 1 列和第 3 列之前做分割
[array([[0],
       [4],
       [8]]), array([[ 1,  2],
       [ 5,  6],
       [ 9, 10]]), array([[ 3],
       [ 7],
       [11]])]
>>>
```

5.3.10 随机打乱数组中的元素顺序

numpy.random 子模块中的 shuffle()和 permutation()函数均可以随机打乱数组元素的顺序,也就是进行洗牌操作。shuffle()函数是对参数中的数组对象原地操作,直接打乱原数组; permutation()函数返回打扰参数中数组元素后得到的新数组,参数中的原数组保持不变。

1. numpy.random.shuffle()

numpy.random.shuffle(x)改变数组 x 中的元素顺序,是对数组 x 的原地洗牌,不返回新的数组对象。如果数组 x 为多维数组,只打乱第一维(只对第一维洗牌)。如果 x 是一个二维数组,只对行(第一维)进行打乱操作。例如:

```
>>> import numpy as np
>>> x = np.arange(12)
>>> x
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> np.random.shuffle(x)
>>> x
array([11, 10,  2,  4,  3,  5,  7,  0,  8,  9,  6,  1])
>>>
>>> x = np.arange(12).reshape((4,3))
>>> x
array([[ 0,  1,  2],
       [ 3,  4,  5],
```

```
       [ 6,  7,  8],
       [ 9, 10, 11]])
>>> np.random.shuffle(x)
>>> x
array([[ 3,  4,  5],
       [ 0,  1,  2],
       [ 9, 10, 11],
       [ 6,  7,  8]])
>>>
```

2. numpy.random.permutation()

numpy.random.permutation(x)对参数中的数组 x 元素顺序进行打乱操作(洗牌)得到一个新的数组,参数中的数组 x 本身保持不变。如果 x 为多维数组,只对第一维的顺序进行打乱操作。如果 x 是一个二维数组,只对行进行洗牌。例如:

```
>>> import numpy as np
>>> x = np.arange(12)
>>> x
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> y = np.random.permutation(x)
>>> y
array([ 1,  5,  8,  6,  3, 11,  9,  0,  7,  2,  4, 10])
>>> x
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>>
>>> x = np.arange(12).reshape((4,3))
>>> x
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
>>> y = np.random.permutation(x)
>>> y
array([[ 0,  1,  2],
       [ 9, 10, 11],
       [ 6,  7,  8],
       [ 3,  4,  5]])
>>> x
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
>>>
```

5.3.11 多维数组的展开

利用数组的 ravel()或 flatten()可以将数组展平为一维数组。ravel()方法返回原数组的视图,在新数组或原数组任一对象上对元素的修改均可在另一个对象上看到修改后

的结果。flatten()方法返回数组的一个 copy，原数组和新数组相互独立，原数组或新数组对元素的修改互不影响。例如：

```
>>> import numpy as np
>>> x = np.arange(12).reshape(3,4)
>>> x
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> y = x.ravel()                          #用 ravel 展平数组，产生原数组的一个视图
>>> y
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> y[0] = 50                              #修改新数组中的一个元素
>>> y
array([50,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> x                                      #原数组中相应位置体现了修改的结果
array([[50,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> z = x.flatten()                        #用 flatten 展开数组，生成原数组的副本
>>> z
array([50,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> z[1] = 100                             #对新数组元素的修改
>>> z
array([ 50, 100,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11])
>>> x                                      #原数组相应位置上的值没有变化
array([[50,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

5.3.12 其他常用函数与对象

1. numpy.where(condition,[x,y])函数

参数 condition、x 和 y 都是类似于数组的对象（如数组、列表等）。根据条件 condition，返回从 x 或 y 中选择的元素构成的数组。对应位置上，如果 condition 中的元素为 True，则结果数组中取 x 中相应位置的元素；否则，结果数组中取 y 中相应位置的元素。如果参数只有 condition，此函数返回的是 np.asarray(condition).nonzero()的结果。

下面给出 where()函数的几个使用实例。

```
>>> import numpy as np
>>> v = np.arange(9)
>>> v
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> np.where(v > 6, v + 100, v * 10)
array([  0,  10,  20,  30,  40,  50,  60, 107, 108])
```

```
>>> v                                    #v本身保持不变
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>>
>>> c = np.array([[True,False,False],[False,True,False]])
>>> c
array([[ True, False, False],
       [False,  True, False]])
>>> v = np.arange(6).reshape((2,3))
>>> v
array([[0, 1, 2],
       [3, 4, 5]])
>>> np.where(c,v,v + 10)
array([[ 0, 11, 12],
       [13,  4, 15]])
>>>
```

当参数中只有 condition，没有 x 和 y 时，等价于 np.asarray(condition).nonzero()。例如：

```
>>> import numpy as np
>>> c = np.array([5,0,8,6,0,1])
>>> np.where(c)
(array([0, 2, 3, 5], dtype = int64),)
>>>
```

如上代码返回的是非 0 元素的位置索引，表示 0、2、3 和 5 索引位置上的元素为非 0。

```
>>> np.where(c > 5)
(array([2, 3], dtype = int64),)
>>>
```

如上代码返回的是大于 5 的元素索引值 2 和 3。

当 condition 中的数组为一个 n 维数组时，返回的是一个由 n 个数组构成的元组。元组中的每个数组表示满足条件的元素在每个原数组中每个维度上的索引值。例如：

```
>>> c = np.array([[5,0,8],[6,0,1]])
>>> c
array([[5, 0, 8],
       [6, 0, 1]])
>>> np.where(c > 5)
(array([0, 1], dtype = int64), array([2, 0], dtype = int64))
>>>
```

如上代码中返回的第一个数组表示第一个维度(行)上的索引，分别为 0 和 1；第二个数组表示第二个维度(列)上的索引，分别为 2 和 0。然后从两个结果数组中依次取出索引值构成位置坐标：从第一个数组取第 0 个元素值 0，从第二个数组取第 0 个元素值 2，构成坐标(0,2)表示第 0 行第 2 列；从第一个数组取出第 1 个元素 1，从第二个数组取出第 1 个元素 0，构成坐标(1,0)表示第 1 行第 0 列。这两个位置上的元素分别为 8 和 6，均满足大于 5。

当 numpy.where()函数中的三个参数均为一维数组时，该函数的作用等价于以下生

成式：

```
[xValue  if  c  else  yValue  for c, xValue, yValue  in  zip(condition, x, y)]
```

2. numpy.piecewise()函数

函数 piecewise(x,condlist,funclist,*args,**kw)根据 x 中的元素满足 condlist 中的哪个条件选择执行 funclist 中对应的操作来生成新数组对象中的元素。

参数 x 表示要进行操作的标量或类似于数组的对象(如数组、列表等)；condlist 是一个由布尔数组或布尔标量构成的列表；funclist 是一个由形式为 f(x[,*args,**kw])的函数或标量构成的列表，与 condlist 中的元素一一对应。当 condlist 中的某个条件为 True 时，执行 funclist 上对应位置的操作；*args 表示接收不定个数的位置参数，然后传递给 f(x[,*args,**kw])；**kw 表示接收不定个数的关键参数，然后传递给 f(x[,*args,**kw])。最终返回一个与 x 相同形状的数组，其元素是根据 condlist 中条件所对应的 funclist 中的 f 函数计算得到的结果，如果 x 中的元素不满足 condlist 中的任何条件，则返回 0。例如：

```
>>> x = np.arange(9)
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> np.piecewise(x,[x<3, x>6],[-1, 1])
array([-1, -1, -1,  0,  0,  0,  0,  1,  1])
>>>
>>> def f1(x):
    return x+10

>>> def f2(x):
    return x+100

>>> np.piecewise(x,[x<3, x>6],[f1, f2])
array([ 10,  11,  12,   0,   0,   0,   0, 107, 108])
>>>
>>> np.piecewise(x,[x<3, x>6],[lambda k : k+10, lambda k : k+100])
array([ 10,  11,  12,   0,   0,   0,   0, 107, 108])
>>>
>>> x = np.arange(9).reshape(3,3)
>>> x
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> np.piecewise(x,[x<3, x>6],[lambda k : k+10, lambda k : k+100])
array([[ 10,  11,  12],
       [  0,   0,   0],
       [  0, 107, 108]])
>>>
```

3. numpy.clip()函数

函数 numpy.clip(a,a_min,a_max,out=None)的作用是剪辑数组中的元素，将元素

限制在 a_min 和 a_max 之间，小于 a_min 的元素用 a_min 来替换，大于 a_max 的元素用 a_max 来替换。

参数 a 是一个类似于数组(array_like)的对象，包含待剪辑的元素。a_min 表示最小值，可以为标量、数组或 None；如果为 None，不对较低区间的边缘进行剪辑。a_max 表示最大值，可以为标量、数组或 None；其他含义与 a_min 类似。a_min 和 a_max 不能同时为 None。out 为一个数组对象，是可选参数。如果有该参数，执行结果将被保存到这个数组对象中。它可能就是输入数组，用来表示在原数组中做剪裁操作。该对象的形状必须正确，以确保能够正确容纳输出结果。例如：

```
>>> a = np.arange(9)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> np.clip(a,2,6)
array([2, 2, 2, 3, 4, 5, 6, 6, 6])
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> b = np.clip(a,2,6)
>>> b
array([2, 2, 2, 3, 4, 5, 6, 6, 6])
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> np.clip(a,2,6,out = a)
array([2, 2, 2, 3, 4, 5, 6, 6, 6])
>>> a
array([2, 2, 2, 3, 4, 5, 6, 6, 6])
>>>
>>> a = np.arange(9).reshape((3,3))
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> np.clip(a,2,6)
array([[2, 2, 2],
       [3, 4, 5],
       [6, 6, 6]])
>>>
```

4. numpy.unique()函数

numpy.unique(ar, return_index = False, return_inverse = False, return_counts = False, axis = None)函数用于查找数组中的唯一元素，返回数组中已排序的唯一元素。

ar 表示待查找的数组，如果没有指定在哪个行或列上操作，数组将被展平进行查找。例如：

```
>>> x = np.random.randint(0,5,(3,4))
>>> x
array([[1, 1, 4, 4],
```

```
        [0, 4, 2, 0],
        [2, 4, 0, 1]])
>>> np.unique(x)
array([0, 1, 2, 4])
>>>
```

return_index 为布尔值，如果为 True，将同时返回唯一值在 ar 中首次出现的索引所构成的数组。这时，np.unique()返回一个元组。此元组第一个元素为所有唯一元素构成的有序数组；另外有一个元素为各唯一值在原数组中首次出现的位置（索引值）所构成的数组。例如：

```
>>> np.unique(x, return_index = True)
(array([0, 1, 2, 4]), array([4, 0, 6, 2], dtype = int64))
>>>
```

return_inverse 为布尔值，如果为 True，将同时返回 ar 数组中相应元素在唯一值数组中的位置。这时，np.unique()返回一个元组。该元组的第一个元素是原数组中唯一值构成的有序数组，另外有一个元素是由原数组中各元素在唯一值数组中的位置值所构成的数组。例如：

```
>>> np.unique(x, return_inverse = True)
(array([0, 1, 2, 4]), array([1, 1, 3, 3, 0, 3, 2, 0, 2, 3, 0, 1], dtype = int64))
>>> np.unique(x,return_index = True,return_inverse = True)
(array([0, 1, 2, 4]), array([4, 0, 6, 2], dtype = int64), array([1, 1, 3, 3, 0, 3, 2, 0, 2, 3,
0, 1], dtype = int64))
>>>
```

可以利用 return_inverse＝True 返回的索引数组重构展平后的数组。

```
>>> unique_data, index_data = np.unique(x,return_inverse = True)
>>> unique_data
array([0, 1, 2, 4])
>>> index_data
array([1, 1, 3, 3, 0, 3, 2, 0, 2, 3, 0, 1], dtype = int64)
>>> unique_data[index_data]                    #得到展平后的一维数组
array([1, 1, 4, 4, 0, 4, 2, 0, 2, 4, 0, 1])
>>>
```

return_counts 为布尔值，如果为 True，将返回每项唯一值在原数组 ar 中出现的次数。这时，np.unique()返回一个元组，该元组的第一个元素是由原数组中唯一值构成的有序数组，另外有一个元素是由每个唯一值在原数组中出现的次数所构成的数组。例如：

```
>>> np.unique(x,return_counts = True)
(array([0, 1, 2, 4]), array([3, 3, 2, 4], dtype = int64))
>>>
```

可以对数组中的某一行或某一列计算唯一值构成的有序数组。例如：

```
>>> y = np.random.randint(0,9,(5,6))
>>> y
array([[4, 7, 3, 7, 6, 5],
```

```
       [6, 6, 3, 2, 3, 4],
       [0, 2, 3, 0, 0, 5],
       [7, 7, 0, 8, 1, 2],
       [5, 6, 2, 7, 2, 0]])
>>> y[1]
array([6, 6, 3, 2, 3, 4])
>>> np.unique(y[1])
array([2, 3, 4, 6])
>>> y[:,1]
array([7, 6, 2, 7, 6])
>>> np.unique(y[:,1])
array([2, 6, 7])
>>>
```

在多维数组中，指定轴 axis 可以返回该轴向上的唯一值。例如：

```
>>> z = np.array([[1,2,3],[5,7,6],[1,2,3]])
>>> z
array([[1, 2, 3],
       [5, 7, 6],
       [1, 2, 3]])
>>> np.unique(z,axis = 0)                    # 去除了重复行
array([[1, 2, 3],
       [5, 7, 6]])
>>> np.unique(z,axis = 0,return_index = True)
(array([[1, 2, 3],
       [5, 7, 6]]), array([0, 1], dtype = int64))
>>> np.unique(z,axis = 0,return_index = True,return_inverse = True)
(array([[1, 2, 3],
       [5, 7, 6]]), array([0, 1], dtype = int64), array([0, 1, 0], dtype = int64))
>>> np.unique(z,axis = 0,return_index = True,return_counts = True)
(array([[1, 2, 3],
       [5, 7, 6]]), array([0, 1], dtype = int64), array([2, 1], dtype = int64))
>>>
```

用于一维数组的情况类似，例如：

```
>>> x = np.random.randint(0,5,10)          # 创建 10 个整数构成的一维数组
>>> x
array([3, 1, 4, 0, 2, 2, 1, 0, 3, 0])
>>> np.unique(x)
array([0, 1, 2, 3, 4])
>>>
```

读者可以自行在一维数组中对每种参数的使用情况进行试验。

5. numpy.ogrid 对象

numpy.ogrid 是 numpy.lib.index_tricks.OGridClass 类的一个内置对象，可以直接引用。返回一个多维的数组或数组列表。每一维通过切片方式获得数组。如果是一维切片，则获得一维数组，如果是 n(n≥2)维切片，则获得 n 个 n 维数组构成的列表，其中每个

数组只有一个维度元素个数大于1。可以通过广播这些数组构成网格矩阵。

如果步长不是复数，则切片不包含结束值。例如：

```
>>> import numpy as np
>>> np.ogrid[1:5:0.5]                    #步长不是复数,结果不包含结束值
array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
>>>
```

如果步长为j表示的复数，j前面的整数表示以等距离分割返回的元素个数，包含结束值。例如：

```
>>> np.ogrid[1:5:10j]                    #步长为复数,结果包含结束值
array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

以下代码得到两个二维数组，第一个数组只有第一个维度的元素个数大于1，第二个数组只有第二个维度的元素个数大于1。每一维上的元素值都通过切片得到。

```
>>> np.ogrid[1:5:0.5, 0:3:0.6]
[array([[1. ],
       [1.5],
       [2. ],
       [2.5],
       [3. ],
       [3.5],
       [4. ],
       [4.5]]), array([[0. , 0.6, 1.2, 1.8, 2.4]])]
>>> np.ogrid[1:5:0.5, 0:3:5j]
[array([[1. ],
       [1.5],
       [2. ],
       [2.5],
       [3. ],
       [3.5],
       [4. ],
       [4.5]]), array([[0.  , 0.75, 1.5 , 2.25, 3.  ]])]
>>> np.ogrid[1:5:0.5, 0:3:5j, 0:2:5j]
[array([[[1. ]],
       [[1.5]],
       [[2. ]],
       [[2.5]],
       [[3. ]],
       [[3.5]],
       [[4. ]],
       [[4.5]]]), array([[[0.  ],
        [0.75],
        [1.5 ],
        [2.25],
        [3.  ]]]), array([[[0. , 0.5, 1. , 1.5, 2. ]]])]
>>>
```

5.4 使用 NumPy 进行简单统计分析

NumPy 数组的基本统计分析函数及其说明见表 5.2。函数中参数的用法请参考帮助文档。

表 5.2 NumPy 数组的基本统计分析函数及其说明

函 数	说 明
sum()	对数组中全部或某轴向的元素求和，如果元素中出现 nan 值，结果为 nan
nansum()	跳过 nan 值，对数组中的全部或某轴向的其余元素求和
mean()	计算数组中全部元素或某轴向元素的算术平均数，如果元素中出现 nan 值，结果为 nan
nanmean()	跳过 nan 值，计算数组中全部元素或某轴向其余元素的算术平均数，其中 nan 值的位置不算元素个数
std()，var()	计算数组中全部元素或某轴向元素的标准差或方差，如果元素中出现 nan 值，结果为 nan
nanstd()/nanvar()	跳过 nan 值，计算数组中全部元素或某轴向其余元素的标准差或方差
min()，max()	计算数组中全部元素或某轴向元素的最小值、最大值，如果元素中出现 nan 值，结果为 nan
nanmin()/nanmax()	跳过 nan 值，计算数组中全部元素或某轴向其余元素的最小值、最大值
argmin()，argmax()	计算数组中的全部元素或某轴向元素的最小值、最大值对应的索引（下标）。使用 argmin 时，如果数组中出现 nan 值，则 nan 值作为最小值处理；使用 argmax 时，如果数组中出现 nan 值，则该 nan 值作为最大值处理
nanargmin()/nanargmax()	跳过 nan 值，计算数组中全部或某轴向其余元素的最小值、最大值在数组中对应的索引（下标）
cumsum()	计算数组中全部元素或某轴向元素各位置上的累加和，如果出现了 nan，则后续结果均为 nan
nancumsum()	将 nan 值作为 0，计算数组中全部元素或某轴向元素各位置上的累加和
cumprod()	计算数组中全部元素或某轴向元素的累乘积，如果出现了 nan，则后续结果均为 nan
nancumprod()	将 nan 值作为 1，计算数组中全部元素或某轴向元素的累乘积
argsort()	数组中元素按照某种规则从小到大排序后的索引值所构成的数组
cov()	数组中全部元素或某轴向元素的协方差
corrcoef()	数组中全部元素或某轴向元素的相关系数
bincount()	统计每个值在非负整数数组中出现的次数
median()	计算数组中的中位数
percentile()	找出数组中指定分位的数值

【例 5.3】 stock.csv 中存储某股票一段时间的交易信息。B、C、D 和 E 列数据分别是开盘价、最高价、最低价、收盘价；G 列是交易量。读取收盘价和交易量数据，统计收盘价的算术平均数、加权平均值（权值为交易量）、方差、中位数、最小值和最大值。

程序源代码如下：

```
#example5_3.py
import numpy as np

close_price,change_volume = np.loadtxt('stock.csv',delimiter = ',',
                                       usecols = (4,6),unpack = True,
                                       skiprows = 1)
meanS1 = np.mean(close_price)
print('收盘价的算术平均值:',meanS1)
wavgS1 = np.average(close_price,weights = change_volume)
print('收盘价的加权平均值:',wavgS1)
varS1 = np.var(close_price)
print('收盘价的方差:',varS1)
medianS1 = np.median(close_price)
print('收盘价的中位数:',medianS1)
minS1 = np.min(close_price)
print('收盘价的最小值:',minS1)
maxS1 = np.max(close_price)
print('收盘价的最大值:',maxS1)
```

程序 example5_3.py 的运行结果如下：

```
收盘价的算术平均值: 11.116
收盘价的加权平均值: 11.09117588266202
收盘价的方差: 1.603284
收盘价的中位数: 10.95
收盘价的最小值: 9.53
收盘价的最大值: 13.54
```

【例 5.4】 multi_stock.csv 文件中的 A、B、C 和 D 列分别保存日期和三家企业从 1981 年第一个交易日到 2018 年 6 月 1 日各天的收盘价。读取这三家企业的股票收盘价，并计算这些收盘价的协方差矩阵和相关系数矩阵。

程序源代码如下：

```
#example5_4.py
import numpy as np

c,d,m = np.loadtxt('multi_stock.csv',delimiter = ',',
                   usecols = (1,2,3),unpack = True,skiprows = 1)
covCDM = np.cov([c,d,m])
relCDM = np.corrcoef([c,d,m])
print('C,D,M 三家公司股票收盘价的协方差为:')
print(covCDM)
print('C,D,M 三家公司股票收盘价的相关系数为:')
print(relCDM)
```

程序 example5_3.py 的运行结果如下：

```
C,D,M 三家公司股票收盘价的协方差为:
[[ 212.53933623   31.95173298  645.19695107]
 [  31.95173298   50.28885268  -38.44185581]
```

```
 [ 645.19695107   -38.44185581 3332.0614872 ]]
C,D,M 三家公司股票收盘价的相关系数为:
[[ 1.         0.30905722  0.76668355]
 [ 0.30905722  1.         -0.09391003]
 [ 0.76668355 -0.09391003  1.        ]]
```

可以利用 argsort()函数或数组对象的 argsort()方法返回数组中元素按照某种规则从小到大排序后的索引值所构成的新数组。语法格式如下：

```
numpy.argsort(a, axis = -1, kind = 'quicksort', order = None)
```

其中，参数 a 表示要排序的数组；参数 axis 表示排序的轴向，默认为－1 表示按照数组的最后一个轴来排序(通常用于多维数组)，axis＝0 表示按列排序，axis＝1 表示按行排序，axis＝None 表示将数组扁平化后排序；参数 kind 表示排序算法，有四种算法(quicksort、mergesort、heapsort、stable)可选，默认为 quicksort；参数 order 给出排序的字段参考顺序，当数组 a 是一个定义了字段名的数组时，该参数用一个序列指定首先比较哪个字段，然后比较哪个字段。其中每个字段名可以用字符串表示。并不是所有字段都需要指定。即使某些字段没有出现在 ord 序列中，排序时仍将可能使用未指定的字段，按照它们在 dtype 中出现的顺序作为排序依据，以打破排序时的并列值。例如：

```
>>> import numpy as np
>>> x = np.array([5,3,1,2])                    # 创建数组
```

这时，数组 x 中，元素 5、3、1、2 的索引分别为 0、1、2、3。

```
>>> np.argsort(x)                              # 对数组按照升序排列
array([2, 3, 1, 0], dtype = int64)
```

排序后的结果为索引为 2 的元素 1 放在第一个位置，索引为 3 的元素 2 放在第二个位置，索引为 1 的元素 3 放在第三个位置，索引为 0 的元素 5 放在第四个位置。因此得到结果为[2,3,1,0]构成的数组，表示依次放置原数组中的第 2、3、1、0 位置上的元素将构成一个升序排列的数组。

argsort()函数执行完后，元数组 x 保持不变。例如：

```
>>> x
array([5, 3, 1, 2])
```

也可以利用数组对象 x 的 argsort()方法来完成相同的功能。例如：

```
>>> x.argsort()
array([2, 3, 1, 0], dtype = int64)
```

同样地，执行完数组对象的 argsort()方法后，原数组保持不变。例如：

```
>>> x
array([5, 3, 1, 2])
>>>
```

argsort()函数或方法也可以用于多维数组中。例如：

```
>>> x = np.array([[0, 3, 5], [6, 2, 1], [2, 1, 3]])
>>> x
array([[0, 3, 5],
       [6, 2, 1],
       [2, 1, 3]])
>>> np.argsort(x,axis = 0)                    #按列排序
array([[0, 2, 1],
       [2, 1, 2],
       [1, 0, 0]], dtype = int64)
```

以上述结果的最后一列为例，表示依次放置原数组中最后一列的第 1、2、0 行元素将构成升序排列的数组。

当参数 axis＝1 时，对同一行上的元素进行排序。例如：

```
>>> np.argsort(x,axis = 1)                    #按行排序
array([[0, 1, 2],
       [2, 1, 0],
       [1, 0, 2]], dtype = int64)
```

当参数 axis 不赋值，默认为－1，按照最后一个轴(这里为行)排序。例如：

```
>>> np.argsort(x)
array([[0, 1, 2],
       [2, 1, 0],
       [1, 0, 2]], dtype = int64)
>>> np.argsort(x,axis = -1)                   #axis 为 -1,按照最后一个轴(这里为行)排序
array([[0, 1, 2],
       [2, 1, 0],
       [1, 0, 2]], dtype = int64)
>>>
```

对于具有字段名的数组，可以定义排序的字段参考顺序。例如：

```
>>> x = np.array([(3,1,5),(2,6,4),(3,4,5)], \
       dtype = [("x",np.int32),("y",np.int32),("z",np.int32)])
```

上述操作定义了一个数组，数组中共有三列，字段名分别为 x、y、z。

```
>>> x
array([(3, 1, 5), (2, 6, 4), (3, 4, 5)],
      dtype = [('x', '<i4'), ('y', '<i4'), ('z', '<i4')])
>>> np.argsort(x, order = ('x','z','y'))
array([1, 0, 2], dtype = int64)
```

如上代码中，参数 order＝('x','z','y')表示对数组先按照字段'x'的升序进行排列；如果在字段'x'上的值相同，再按照字段'z'的值升序排列；如果字段'z'的值也相同，则按照字段'y'的值升序排列。

```
>>> np.argsort(x, order = ('z','y','x'))
array([1, 0, 2], dtype = int64)
```

如上代码中，参数 order＝('z','y','x')表示对数组先按照字段'z'的升序进行排列；如

果在字段'z'上的值相同，再按照字段'y'的值升序排列；如果字段'y'的值也相同，则按照字段'x'的值升序排列。

可以利用 bincount()函数统计一个非负整数数组中从 0 到元素最大值 n 的 n+1 个元素分别出现的次数。语法格式如下：

```
numpy.bincount(x, weights = None, minlength = 0)
```

其中，x 是待统计的非负整数数组；weights 是一个加权数组，与 x 的形状相同；参数 minlength 表示输出数组的最小长度。例如：

```
>>> import numpy as np
>>> x = np.random.randint(1,6,size = 5)
>>> x
array([3, 4, 5, 5, 2])
>>> np.bincount(x)
array([0, 0, 1, 1, 1, 2], dtype = int64)
>>>
```

如上代码中，数组 x 中元素的最大值为 5，因此产生数组依次表示 0～5 这 6 个整数各自的个数。

测试输出数组的长度：

```
>>> np.bincount(x).size == np.max(x) + 1
True
>>>
```

如果有权值参数 weights 数组，则计算输出时，累加数组 x 对应位置的权重值，而不是累加个数。其中 weights 数组的元素个数与数组 x 的元素个数相同。

```
>>> x
array([3, 4, 5, 5, 2])
>>> w = np.array([0.3,0.6,0.2,0.5, - 0.1])
>>> np.bincount(x,weights = w)
array([ 0. ,  0. , - 0.1,  0.3,  0.6,  0.7])
>>>
```

上述结果中最后一个位置上的 0.7 由两个 5 对应位置上的权重之和构成。

如果参数 minlength 被指定，那么输出数组中元素的数量至少为 minlength 指定的数。如果待计算数组 x 中元素最大值 n 所决定的元素个数 n+1 大于 minlength，则输出 n+1 个元素。例如：

```
>>> x
array([3, 4, 5, 5, 2])
```

以下代码指定了 minlength=8，至少输出 0～7 的元素个数：

```
>>> np.bincount(x, minlength = 8)
array([0, 0, 1, 1, 1, 2, 0, 0], dtype = int64)
```

以下代码虽然指定 minlength=3，但 x 中的元素最大值为 5，至少输出 0～5 这 6 个

元素的个数：

```
>>> np.bincount(x, minlength = 3)
array([0, 0, 1, 1, 1, 2], dtype = int64)
>>>
```

可以利用 numpy.percentile()函数在多维数组中沿指定轴计算数据在指定分位上的值。可以计算 0～100 任意分位对应的值。其语法格式如下：

```
numpy.percentile(a, q, axis = None, out = None, overwrite_input = False, interpolation =
'linear', keepdims = False)
```

其中，参数 a 表示数组；参数 q 表示要计算的百分位数或百分位数序列，其值必须介于 0～100；参数 axis 表示计算的轴向，0 表示各列分别计算，1 表示各行分别计算，默认值 None 表示将数组展平后计算对应的分位数；参数 out 表示用于放置结果的数组，必须具有与预期输出相同的形状和缓冲区长度；参数 interpolation 表示当所需的分位值位于两个数据点之间时使用的插值方法，可以从集合{'linear','lower','higher','midpoint','nearest'}中取值，具体可以查看帮助文档；参数 keepdims 为布尔值，如果为 True，结果将保持与元素矩阵 a 相同的维度数；如果为默认值 False，计算结果将被展开。例如：

```
>>> x = np.array([[0, 3, 5], [6, 2, 1]])
>>> x
array([[0, 3, 5],
       [6, 2, 1]])
>>> np.percentile(x,50)
2.5
>>> np.percentile(x,50,axis = 0)                    #各列分别计算
array([3. , 2.5, 3. ])
>>> np.percentile(x,50,axis = 1)                    #各行分别计算
array([3., 2.])
>>> np.percentile(x,50,axis = 1,keepdims = True)    #保持维度
array([[3.],
       [2.]])
>>>
```

5.5 数组在其他文件中的存取

为了方便将来使用或数据的传递，可以将数组存储在文件中，使用时再取出。5.2.2 节介绍了 NumPy 数组在文本文件中的存取方法，本节介绍 NumPy 数组在其他文件中的存取方法。

5.5.1 数组在无格式二进制文件中的存取

使用数组的 tofile()方法可以将数组中的数据以二进制的格式写进文件。tofile()方法写入文件的数据不保存数组形状和元素类型等信息。写入的文件扩展名没有特定要求。例如：

```
>>> x = np.array([[1,2],[3,4]],dtype = np.int32)
>>> x
array([[1, 2],
       [3, 4]])
>>> x.tofile("d:/test/array.bin")
```

由于通过 tofile()方法写入的数组内容在文件中没有形状等信息,因此从该文件通过 np.fromfile()读取的都是一维数组。

```
>>> y = np.fromfile("d:/test/array.bin",dtype = np.int32)
>>> y
array([1, 2, 3, 4])
>>>
```

5.5.2 数组在 npy 文件中的存取

numpy.save()函数可以保存任意维度的 NumPy 数组到一个扩展名为 npy 的二进制文件中。除了保存数据信息外,该方法同时将形状、元素类型等信息保存到文件中。注意,保存的文件扩展名必须为 npy,否则使用 load()函数读取时将出错。例如:

```
>>> x = np.array([[1,2],[3,4]],dtype = np.int32)
>>> x
array([[1, 2],
       [3, 4]])
>>> np.save("d:/test/array.npy",x)
```

用 numpy.load()函数读取.npy 文件保存信息的示例如下。

```
>>> y = np.load("d:/test/array.npy")
>>> y
array([[1, 2],
       [3, 4]])
>>>
```

也可以在一个 npy 文件中写入多个数组。例如:

```
>>> z = np.array(range(24)).reshape((2, 3, 4))
>>> z
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
>>> f = open("d:/test/multi_array.npy","wb")
>>> np.save(f,x)
>>> np.save(f,z)
>>> f.close()
```

可以用以下方法从一个 npy 文件中读取多个数组信息。

```
>>> f = open("d:/test/multi_array.npy","rb")
>>> np.load(f)
array([[1, 2],
       [3, 4]])
>>> np.load(f)
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
>>>
```

5.5.3 数组在 npz 文件中的存取

numpy.savez()函数可以一次性将多个数组保存在同一个扩展名为 npz 的二进制文件中。该文件中除了保存数组元素，还保存数组形状、元素类型等信息。例如：

```
>>> x = np.array([[1,2],[3,4]],dtype = np.int32)
>>> x
array([[1, 2],
       [3, 4]])
>>> y = np.array(range(24)).reshape((2, 3, 4))
>>> y
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
>>> np.savez("d:/test/array.npz",x,y)
```

可以利用 numpy.load()函数读取这个文件的信息，然后依次获得所有数组。例如：

```
>>> z = np.load("d:/test/array.npz")
>>> z["arr_0"]
array([[1, 2],
       [3, 4]])
>>> z["arr_1"]
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
>>>
```

5.5.4 数组在 hdf5 文件中的存取

利用 hdf5 格式的文件存储数组，也可以同时存储数组的形状和元素类型。hdf5 文件占用空间相对较小，适合数组很大的情况。例如：

```
>>> import numpy as np
>>> import h5py
>>> x = np.array([[1,2],[3,4]],dtype = np.int32)
>>> x
array([[1, 2],
       [3, 4]])
>>> y = np.array(range(24)).reshape((2, 3, 4))
>>> y
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
>>> f = h5py.File("d:/test/array.h5","w")
>>> f.create_dataset("x",data = x)
<HDF5 dataset "x": shape (2, 2), type "<i4">
>>> f.create_dataset("y",data = y)
<HDF5 dataset "y": shape (2, 3, 4), type "<i4">
>>> f.close()
```

这时，在 d：/test/array.h5 文件中存入了两个数组 x 和 y，分别用关键字"x"和"y"来标记。接着可以通过 h5py 文件对象关键字和切片取得相应数组。例如：

```
>>> f = h5py.File("d:/test/array.h5","r")
>>> type(f)
<class 'h5py._hl.files.File'>
>>> f["x"]
<HDF5 dataset "x": shape (2, 2), type "<i4">
>>> f["x"][:]
array([[1, 2],
       [3, 4]])
>>> f["y"][:]
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
>>>
```

以上给出了利用 hdf5 文件存取 NumPy 数组的简单示例。可以利用切片来进一步

提高大数组的存取效率，这里不展开讨论。

习题 5

1. 下载一只股票某段时间内的每日交易信息数据，用 NumPy 读取该数据文件，统计收盘价的算术平均数、加权平均值（权值为交易量）、方差、中位数、最小值、最大值。

2. 下载一只股票某段时间内的每日交易信息数据，用 NumPy 读取该数据文件中的开盘价、最高价和收盘价数据，并计算这些数据的协方差矩阵和相关系数矩阵。

3. 下载一个开源数据集，用 NumPy 读取该文件，对数据执行排序操作，然后将排序后的数据存储到 npy 文件中。

第6章

Matplotlib数据可视化基础

学习目标

- 掌握利用 Matplotlib 绘制基本图形的流程。
- 掌握绘制多轴图的方法。
- 掌握图形基本属性的设置。
- 掌握三维图形的绘制方法。

Matplotlib 是 Python 中最著名的绘图库，其中，pyplot 模块包含大量与 MATLAB 相似的函数调用接口，非常适合进行绘图以达到数据可视化的目的。

Matplotlib 是非标准库，需要提前安装。Anaconda 中集成了这个库，可以直接使用 Anaconda，也可以不安装 Anaconda。这样需要安装完官方标准的 Python 发行版后在线或下载后安装 Matplotlib。

如果采用在线安装，打开 Windows 系统的命令窗口，输入 pip install matplotlib 命令，完成 Matplotlib 的安装。当网络传输质量不高时，在线安装可能会出现中断，导致安装失败。可以下载安装模块后在本地安装，这时需要先到官方网站下载 whl 文件。然后打开 Windows 系统的命令窗口，进入下载文件所在目录，执行 pip install 文件名.whl 命令。

使用 Matplotlib 创建图表的标准步骤如下：首先，创建 Figure 对象；接着，用 Figure 对象创建一个或者多个 Axes 或者 Subplot 对象；最后，调用 Axies 等对象的方法创建各种简单类型的图表。其中系统会默认创建一个 Figure 对象，因此一般可以省略此步骤。

6.1 绘制基本图形

本节主要介绍折线图、散点图、直方图和饼图等基本图形的绘制方法，并给出图形的一些基本属性设置方法。