第3章

NumPy数值计算基础

NumPy是在1995年诞生的Python库Numeric的基础上建立起来的，但真正促使NumPy发行的是Python的SciPy库。SciPy是2001年发行的一个类似于MATLAB、Maple和Mathematica等数学计算软件的Python库，它可以实现数值计算中的大多数功能，但SciPy中并没有合适的类似于Numeric中的对于基础数据对象处理的功能。于是，SciPy的开发者将SciPy中的一部分和Numeric的设计思想结合，在2005年发行了NumPy。

NumPy是Python的一种开源的数值计算扩展库。它包含很多功能，如创建n维数组(矩阵)、对数组进行函数运算、数值积分、线性代数运算、傅里叶变换和随机数产生等。

标准的Python用List(列表)保存值，可以当作数组使用，但因为列表中的元素可以是任何对象，所以浪费了CPU运算的时间和内存。NumPy的诞生弥补了这些缺陷，它提供了两种基本的对象。

- ndarray(n-dimensional array object)：是存储单一数据类型的多维数组；
- ufunc(universal function object)：是一种能够对数组进行处理的函数。

NumPy常用的导入格式：

```
import numpy as np
```

导入NumPy后，可通过np.＋Tab键查看可使用的函数。如果对其中一些函数的使用不是很清楚，可以在对应的函数名后＋？(如np.abs?)运行，可以方便地查看相应函数的帮助信息。


视频讲解

3.1 NumPy 多维数组

3.1.1 创建数组对象

通过 NumPy 库的 array 函数可以创建 ndarray 数组。通常来说，ndarray 是一个通用的同构数据容器，即其中的所有元素都需要相同的类型。NumPy 库能将数据（列表、元组、数组或其他序列类型）转换为 ndarray 数组。

1. 使用 array 函数创建数组对象

array 函数的格式：

```
np.array(object, dtype,ndmin)
```

array 函数的主要参数及其使用说明见表 3-1。

表 3-1 array 函数的主要参数及其说明

参 数	说 明
object	接收 array，表示想要创建的数组
dtype	接收 data-type，表示数组所需的数据类型，未给定则选择保存对象所需的最小类型，默认为 None
ndmin	接收 int，指定生成数组应该具有的最小维数，默认为 None

【例 3-1】 创建 ndarray 数组。

```
In[1]: import numpy as np
       data1 = [1,3,5,7]                    #列表
       w1 = np.array(data1)
       print('w1:',w1)
       data2 = (2,4,6,8)                    #元组
       w2 = np.array(data2)
       print('w2:',w2)
       data3 = [[1,2,3,4],[5,6,7,8]]        #多维数组
       w3 = np.array(data3)
       print('w3:',w3)
Out[1]: w1: [1 3 5 7]
        w2: [2 4 6 8]
        w3: [[1 2 3 4]
             [5 6 7 8]]
```

在创建数组时，NumPy 会为新建的数组推断出一个合适的数据类型，并保存在 dtype 中，当序列中有整数和浮点数时，NumPy 会把数组的 dtype 定义为浮点数据类型。

【例 3-2】 在 array 函数中指定 dtype。

```
In[2]:   w3 = np.array([1,2,3,4],dtype = 'float64')
         print(w3.dtype)
```

```
Out[2]:  float64
```

2. 专门创建数组的函数

通过 array 函数使用已有的 Python 序列创建数组效率不高，因此，NumPy 提供了很多专门创建数组的函数。

1) arange 函数

arange 函数类似于 Python 的内置函数 range，但是 arange 主要用来创建数组。

【例 3-3】 使用 arange 创建数组。

```
In[3]:   warray = np.arange(10)
         print(warray)
Out[3]:  [0 1 2 3 4 5 6 7 8 9]
```

arange 函数可以通过指定起始值、终值和步长创建一维数组，创建的数组不包含终值。

【例 3-4】 指定起始值、终值及步长参数的 arange。

```
In[4]:   warray = np.arange(0,1,0.2)
         print(warray)
Out[4]:  [0. 0.2 0.4 0.6 0.8]
```

2) linspace 函数

当 arange 的参数是浮点型时，由于浮点的精度有限，通常不太可能去预测获得元素的数量。出于这个原因，通常选择更好的函数 linspace，它接收元素数量作为参数。linspace 函数通过指定起始值、终值和元素个数创建一维数组，默认包括终值。

【例 3-5】 使用 linspace 函数创建数组。

```
In[5]:   warray = np.linspace(0,1,5)
         print(warray)
Out[5]:  [0. 0.25 0.5 0.75 1. ]
```

3) logspace 函数

logspace 函数和 linspace 函数类似，不同点是它所创建的是等比数列。

【例 3-6】 使用 logspace 函数创建数组。

```
In[6]:   warray = np.logspace(0,1,5)
         #生成 1～10 的具有 5 个元素的等比数列
         print(warray)
Out[6]:  [1. 1.77827941 3.16227766 5.62341325?10.]
```

logspace 的参数中，起始位和终止位代表的是 10 的幂(默认基数为 10)，第三个参数表示元素个数。

4) zeros 函数

zeros 函数可以创建指定长度或形状的全 0 数组。

【例 3-7】 使用 zeros 函数创建全零矩阵。

```
In[7]:   print(np.zeros(4))
```

```
          print(np.zeros([3,3]))
Out[7]:   [0. 0. 0. 0.]
          [[0. 0. 0.]
          [0. 0. 0.]
          [0. 0. 0.]]
```

5）ones 函数

ones 函数可以创建指定长度或形状的全 1 数组。

【例 3-8】 使用 ones 函数创建全 1 数组。

```
In[8]:    print(np.ones(5))
          print(np.ones([2,3]))
Out[8]:   [1. 1. 1. 1. 1.]
          [[1. 1. 1.]
          [1. 1. 1.]]
```

6）diag 函数

diag 函数可以创建对角矩阵，即对角线元素为 0 或指定值，其他元素为 0。

【例 3-9】 使用 diag 函数创建对角矩阵。

```
In[9]:    print(np.diag([1,2,3,4]))
Out[9]:   [[1 0 0 0]
          [0 2 0 0]
          [0 0 3 0]
          [0 0 0 4]]
```

此外，使用 eye 函数可创建一条对角线位置为 1、其他位置全为 0 的矩阵。

3.1.2 ndarray 对象属性和数据转换

NumPy 创建的 ndarray 对象属性主要有 shape、size 等属性，具体见表 3-2。

表 3-2 ndarray 对象属性及其说明

属 性	说 明
ndim	秩，即数据轴的个数
shape	数组的维度
size	数组元素个数
dtype	数据类型
itemsize	数组中每个元素的字节大小

【例 3-10】 查看数组的属性。

```
In[10]:   warray = np.array([[1,2,3],[4,5,6]])
          print('秩为:',warray.ndim)
          print('形状为:',warray.shape)
          print('元素个数为:',warray.size)
Out[10]:  秩为: 2
          形状为: (2, 3)
          元素个数为: 6
```

数组的 shape 可以重新设置。

【例 3-11】 设置数组的 shape 属性。

```
In[11]:   warray.shape = 3,2
          print(warray)
Out[11]:  [[1 2]
          [3 4]
          [5 6]]
```

对于创建好的数组 ndarray,可以通过 astype 方法进行数据类型的转换。

【例 3-12】 数组的类型转换。

```
In[12]:   arr1 = np.arange(6)
          print(arr1.dtype)
          arr2 = arr1.astype(np.float64)
          print(arr2.dtype)
Out[12]:  int32
          float64
```

3.1.3 生成随机数

在 NumPy.random 模块中,提供了多种随机数的生成函数,例如 randint 可以生成指定范围的随机整数。

用法:

```
np.random.randint(low,high = None,size = None)
```

【例 3-13】 生成随机整数。

```
In[13]:   arr = np.random.randint(100,200,size = (2,4))
          print(arr)
Out[13]:  [[197 129 112 153]
          [138 195 114 141]]
```

【例 3-14】 生成[0,1]的随机数组。

```
In[14]:   arr1 = np.random.rand(5)
          print(arr1)
          arr2 = np.random.rand(4,2)
          print(arr2)
Out[14]:  [0.13654637 0.09218044 0.44985683 0.24374376 0.60841164]
          [[0.07250518 0.50867613]
          [0.21831215 0.23476073]
          [0.81293096 0.92887008]
          [0.28339637 0.82806109]]
```

注意:因为是随机数,每次运行代码生成的随机数组都不一样。为了每次生成同一份数据,可以指定一个随机种子,然后使用 shuffle 函数打乱生成的随机数。表 3-3 列出了 random 模块常用的随机数生成方法。

表 3-3 random 模块常用的随机数生成函数

函 数	说 明
seed	确定随机数生成器的种子
permutation	返回一个序列的随机排列或返回一个随机排列的范围
shuffle	对一个序列进行随机排序
binomial	产生二项分布的随机数
normal	产生正态(高斯)分布的随机数
beta	产生 beta 分布的随机数
chisquare	产生卡方分布的随机数
gamma	产生 gamma 分布的随机数
uniform	产生在[0,1)中均匀分布的随机数

3.1.4 数组变换

1. 数组重塑

对于定义好的数组,可以通过 reshape 方法改变其数组维度,传入的参数为新维度的元组。

【例 3-15】 改变数组维度。

```
In[15]:  arr1 = np.arange(8)
         print(arr1)
         arr2 = arr1.reshape(4,2)
         print(arr1)
Out[15]: [0 1 2 3 4 5 6 7]
         [[0 1]
          [2 3]
          [4 5]
          [6 7]]
```

reshape 中的一个参数可以设置为-1,表示数组的维度可以通过数据本身来推断。

【例 3-16】 reshape 的一个维度设置为-1。

```
In[16]:  arr1 = np.arange(12)
         print('arr1:',arr1)
         arr2 = arr1.reshape(2, -1)
         print('arr2:',arr2)
Out[16]: arr1: [ 0 1 2 3 4 5 6 7 8 9 10 11]
         arr2: [[ 0 1 2 3 4 5]
                [ 6 7 8 9 10 11]]
```

与 reshape 相反的方法是数据散开(ravel)或数据扁平化(flatten)。

【例 3-17】 数据散开。

```
In[17]:  arr1 = np.arange(12).reshape(3,4)
         print('arr1:',arr1)
         arr2 = arr1.ravel()
```

```
          print('arr2:',arr2)
Out[17]:  arr1: [[ 0 1 2 3]
          [ 4 5 6 7]
          [ 8 9 10 11]]
          arr2: [ 0 1 2 3 4 5 6 7 8 9 10 11]
```

需要注意的是,数据重塑不会改变原来的数组。

2. 数组合并

数组合并用于多个数组间的操作,NumPy 使用 hstack、vstack 和 concatenate 函数完成数组的合并。

横向合并是将 ndarray 对象构成的元组作为参数,传给 hstack 函数。

【例 3-18】 两个数组的横向合并。

```
In[18]:   arr1 = np.arange(6).reshape(3,2)
          arr2 = arr1 * 2
          arr3 = np.hstack((arr1,arr2))
          print(arr3)
Out[18]:  [[ 0 1 0 2]
          [ 2 3 4 6]
          [ 4 5 8 10]]
```

纵向合并是使用 vstack 将数组纵向合并。

【例 3-19】 数组纵向合并。

```
In[19]:   arr1 = np.arange(6).reshape(3,2)
          arr2 = arr1 * 2
          arr3 = np.vstack((arr1,arr2))
          print(arr3)
Out[19]:  [[ 0 1]
          [ 2 3]
          [ 4 5]
          [ 0 2]
          [ 4 6]
          [ 8 10]]
```

concatenate 函数可以实现数组的横向合并或纵向合并,其中的参数 axis=1 时进行横向合并,axis=0 时进行纵向合并。

【例 3-20】 使用 concatenate 函数合并数组。

```
In[20]:   arr1 = np.arange(6).reshape(3,2)
          arr2 = arr1 * 2
          print('横向合并为:',np.concatenate((arr1,arr2),axis = 1))
          print('纵向合并为:',np.concatenate((arr1,arr2),axis = 0))
Out[20]:  横向合并为: [[ 0 1 0 2]
                      [ 2 3 4 6]
                      [ 4 5 8 10]]
          纵向合并为: [[ 0 1]
```

```
       [ 2 3]
       [ 4 5]
       [ 0 2]
       [ 4 6]
       [ 8 10]]
```

3. 数组分割

与数组合并相反，NumPy 提供了 hsplit、vsplit 和 split 分别实现数组的横向、纵向和指定方向的分割。

【例 3-21】 数组的分割。

```
In[21]:  arr = np.arange(16).reshape(4,4)
         print('横向分割为:\n',np.hsplit(arr,2))
         print('纵向组合为:\n',np.vsplit(arr,2))
Out[21]: 横向分割为:
         [array([[ 0, 1],
                [ 4, 5],
                [ 8, 9],
                [12, 13]]),
         array([[ 2, 3],
                [ 6, 7],
                [10, 11],
                [14, 15]])]
         纵向组合为:
         [array([[0, 1, 2, 3],
                [4, 5, 6, 7]]),
         array([[ 8, 9, 10, 11],
                [12, 13, 14, 15]])]
```

同样，split 在参数 axis＝1 时实现数组的横向分割，axis＝0 时则进行纵向分割。

4. 数组转置和轴对换

数组转置是数组重塑的一种特殊形式，可以通过 transpose 方法进行转置。transpose 方法需要传入轴编号组成的元组。

【例 3-22】 数组使用 transpose 转置。

```
In[22]:  arr = np.arange(6).reshape(3,2)
         print('矩阵:',arr)
         print('转置矩阵:',arr.transpose((1,0)))
Out[22]: 矩阵: [[0 1]
               [2 3]
               [4 5]]
         转置矩阵: [[0 2 4]
                   [1 3 5]]
```

除了使用 transpose 外，可以直接使用数组的 T 属性进行数组转置。

【例 3-23】 数组的 T 属性转置。

```
In[23]:  print(arr.T)
Out[23]: [[0 2 4]
          [1 3 5]]
```

ndarray 的 swapaxes 方法实现轴对换。

【例 3-24】 数组的轴对换。

```
In[24]:  arr = np.arange(6).reshape(3,2)
         print(arr)
         print(arr.swapaxes(0,1))
Out[24]: [[0 1]
          [2 3]
          [4 5]]
          [[0 2 4]
          [1 3 5]]
```

视频讲解

3.2 数组的索引和切片

在数据分析中经常会选取符合条件的数据,NumPy 中通过数组的索引和切片进行数组元素的选取。

3.2.1 一维数组的索引

一维数组的索引类似于 Python 中的列表。

【例 3-25】 一维数组的索引。

```
In[25]:  arr = np.arange(10)
         print(arr)
         print(arr[2])
         print(arr[-1])
         print(arr[1:4])
Out[25]: [0 1 2 3 4 5 6 7 8 9]
         2
         9
         [1 2 3]
```

数组的切片返回的是原始数组的视图,并不会产生新的数据,这就意味着在视图上的操作会使原数组发生改变。如果需要的并非视图而是要复制数据,则可以通过 copy 方法实现。

【例 3-26】 数组元素的复制。

```
In[26]:  arr1 = arr[-4:-1].copy()
         print(arr)
         print(arr1)
```

```
Out[26]:  [0 1 2 3 4 5 6 7 8 9]
          [6 7 8]
```

3.2.2 多维数组的索引

对于多维数组，它的每一个维度都有一个索引，各个维度的索引之间用逗号分隔。

【例 3-27】 多维数组的索引。

```
In[27]:   arr = np.arange(12).reshape(3,4)
          print(arr)
          print(arr[0,1:3])          #索引第 0 行中第 1 列到第 2 列的元素
          print(arr[:,2])            #索引第 2 列元素
          print(arr[:1,:1])          #第 0 行第 0 列元素
Out[27]:  [[ 0 1 2 3]
          [ 4 5 6 7]
          [ 8 9 10 11]]
          [1 2]
          [ 2 6 10]
          [[0]]
```

也可以使用整数函数和布尔值索引访问多维数组。

【例 3-28】 访问多维数组。

```
In[28]:   arr = np.arange(12).reshape(3,4)
          #从两个序列的对应位置取出两个整数来组成下标:arr[0,1],arr[1,3]
          print(arr)
          print('索引结果 1:',arr[(0,1),(1,3)])
          #索引第 1 行中第 0、2、3 列的元素
          print('索引结果 2:',arr[1:2,(0,2,3)])
          mask = np.array([1,0,1],dtype = np.bool)
          #mask 是一个布尔数组,它索引第 0、2 行中第 1 列元素
          print('索引结果 3:',arr[mask,1])
Out[28]:  [[ 0 1 2 3]
          [ 4 5 6 7]
          [ 8 9 10 11]]
          索引结果 1: [1 7]
          索引结果 2: [[4 6 7]]
          索引结果 3: [1 9]
```

3.3 数组的运算

视频讲解

数组的运算支持向量化运算，将本来需要在 Python 级别进行的运算放到 C 语言的运算中，会明显地提高程序的运算速度。

3.3.1 数组和标量间的运算

数组之所以很强大是因为不需要通过循环就可以完成批量计算，例如相同维度的数

组的算术运算直接应用到元素中。

【例 3-29】 数组元素的追加。

```
In[29]:   a = [1,2,3]
          b = []
          for i in a:
              b.append(i * i)
          print('b 数组:',b)
          wy = np.array([1,2,3])
          c = wy * 2
          print('c 数组:',c)
Out[29]:  b 数组: [1, 4, 9]
          c 数组: [2 4 6]
```

3.3.2 ufunc 函数

ufunc 函数全称为通用函数,是一种能够对数组中的所有元素进行操作的函数。ufunc 函数针对数组进行操作,而且以 NumPy 数组作为输出。对一个数组进行重复运算时,使用 ufunc 函数比使用 math 库中的函数效率要高很多。

1. 常用的 ufunc 函数运算

常用的 ufunc 函数运算有四则运算、比较运算和逻辑运算。

(1) 四则运算:加(+)、减(-)、乘(*)、除(/)、幂(**)。数组间的四则运算表示对每个数组中的元素分别进行四则运算,所以形状必须相同。

(2) 比较运算:>、<、==、>=、<=、!=。比较运算返回的结果是一个布尔数组,每个元素为每个数组对应元素的比较结果。

(3) 逻辑运算:np.any 函数表示逻辑"or";np.all 函数表示逻辑"and",运算结果返回布尔值。

【例 3-30】 数组的四则运算。

```
In[30]:   x = np.array([1,2,3])
          y = np.array([4,5,6])
          print('数组相加结果:',x + y)
          print('数组相减结果:',x - y)
          print('数组相乘结果:',x * y)
          print('数组幂运算结果:',x ** y)
Out[30]:  数组相加结果: [5 7 9]
          数组相减结果: [-3 -3 -3]
          数组相乘结果: [ 4 10 18]
          数组幂运算结果: [ 1 32 729]
```

ufunc 也可以进行比较运算,返回的结果是一个布尔数组,其中每个元素都是对应元素的比较结果。

【例 3-31】 数组的比较运算。

```
In[31]:  x = np.array([1,3,6])
         y = np.array([2,3,4])
         print('比较结果(<):',x < y)
         print('比较结果(>):',x > y)
         print('比较结果( == ):',x == y)
         print('比较结果(> = ):',x > = y)
         print('比较结果(!= ):',x!= y)
Out[31]: 比较结果(<): [ True False False]
         比较结果(>): [False False True]
         比较结果( == ): [False True False]
         比较结果(> = ): [False True True]
         比较结果(!= ): [ True False True]
```

2. ufunc 函数的广播机制

广播(broadcasting)是指不同形状的数组之间执行算术运算的方式。广播机制需要遵循 4 个原则：

(1) 让所有输入数组都向其中 shape 最长的数组看齐，shape 中不足的部分都通过在前面加 1 补齐。

(2) 输出数组的 shape 是输入数组 shape 的各个轴上的最大值。

(3) 如果输入数组的某个轴和输出数组的对应轴的长度相同或者其长度为 1 时，这个数组能够用来计算，否则出错。

(4) 当输入数组的某个轴的长度为 1 时，沿着此轴运算时都用此轴上的第一组值。

【例 3-32】 ufunc 函数的广播。

```
In[32]:  arr1 = np.array([[0,0,0],[1,1,1],[2,2,2]])
         arr2 = np.array([1,2,3])
         print('arr1:\n',arr1)
         print('arr2:\n',arr2)
         print('arr1 + arr2:\n',arr1 + arr2)
Out[32]: arr1:
         [[0 0 0]
         [1 1 1]
         [2 2 2]]
         arr2:
         [1 2 3]
         arr1 + arr2:
         [[1 2 3]
         [2 3 4]
         [3 4 5]]
```

3.3.3 条件逻辑运算

在 NumPy 中可以使用基本的逻辑运算实现数组的条件运算。

【例 3-33】 数组的逻辑运算。

```
In[33]:   arr1 = np.array([1,3,5,7])
          arr2 = np.array([2,4,6,8])
          cond = np.array([True,False,True,False])
          result = [(x if c else y)for x,y,c in zip(arr1,arr2,cond)]
          result
Out[33]:  [1, 4, 5, 8]
```

但这种方法对大规模数组处理效率不高,也无法用于多维数组。NumPy 提供的 where 方法可以克服这些问题。

where 的用法:

```
np.where(condition, x,y)
```

如果满足条件(condition)输出 x;不满足则输出 y。

【例 3-34】 where 的基本用法。

```
In[34]:   np.where([[True,False], [True,True]],[[1,2], [3,4]],[[9,8], [7,6]])
Out[34]:  array([[1, 8],[3, 4]])
```

在这个例子中条件为[[True,False],[True,True]],分别对应最后输出结果的 4 个值,运算时第一个值从[1,9]中选,因为条件为 True,所以选 1;第二个值从[2,8]中选,因为条件为 False,所以选 8,后面以此类推。

【例 3-35】 where 中只有 condition 参数。

```
In[35]:   w = np.array([2,5,6,3,10])
          np.where(w > 4)
Out[35]:  (array([1, 2, 4], dtype = int64),)
```

where 中若只有条件 (condition),没有 x 和 y,则输出满足条件元素的坐标。这里的坐标以 tuple 的形式给出,通常原数组有多少维,输出的 tuple 中就包含几个数组,分别对应符合条件元素的各维坐标。

3.4 数组读/写

3.4.1 读/写二进制文件

NumPy 提供了多种文件操作函数存取数组内容。文件存取的格式分为两类:二进制和文本。二进制格式的文件又分为 NumPy 专用的格式化二进制类型和无格式类型。NumPy 中读/写二进制文件的方法有以下两种。

(1) NumPy.load("文件名.npy"):从二进制的文件中读取数据。

(2) NumPy.save("文件名[.npy]",arr):以二进制的格式保存数据。

它们会自动处理元素类型和 shape 等信息,使用它们读/写数组就非常方便。但是 np.save 输出的文件很难用其他语言编写的程序读入。

【例 3-36】 数组的读/写。

```
In[36]:  a = np.arange(1,13).reshape(3,4)
         print(a)
         np.save('arr.npy', a)        # np.save("arr", a)
         c = np.load( 'arr.npy' )
         print(c)
Out[36]: [[ 1 2 3 4]
         [ 5 6 7 8]
         [ 9 10 11 12]]
         [[ 1 2 3 4]
         [ 5 6 7 8]
         [ 9 10 11 12]]
```

【例 3-37】 多个数组保存。

```
In[37]:  a = np.array([[1,2,3],[4,5,6]])
         b = np.arange(0, 1.0, 0.1)
         c = np.sin(b)                  #长度为 10
         print(c)
         np.savez('result.npz', a, b, sin_array = c)
         r = np.load('result.npz')
         r['arr_0']                     #数组 a
Out[37]: [0.0.09983342 0.19866933 0.29552021
         0.38941834 0.47942554 0.56464247 0.64421769
         0.71735609 0.78332691]
         array([[1, 2, 3],
                [4, 5, 6]])
```

3.4.2 读/写文本文件

NumPy 中读/写文本文件的主要方法有以下几种。

(1) NumPy.loadtxt("../tmp/arr.txt",delimiter=","):把文件加载到一个二维数组中。

(2) NumPy.savetxt("../tmp/arr.txt",arr,fmt="%d",delimiter=","):将数组写到某种分隔符隔开的文本文件中。

(3) NumPy.genfromtxt("../tmp/arr.txt", delimiter=","):结构化数组和缺失数据。

【例 3-38】 读/写文本文件。

```
In[38]:  a = np.arange(0,12,0.5).reshape(4, -1)
         np.savetxt("a1-out.txt", a)
         #默认按照'%.18e'格式保存数值
         np.loadtxt("a1-out.txt")
         np.savetxt("a2-out.txt", a, fmt = "%d", delimiter = ",")
         #改为保存为整数,以逗号分隔
         np.loadtxt("a2-out.txt",delimiter = ",")
```

```
                # 读入的时候也需要指定逗号分隔
Out[38]:  array([[ 0., 0., 1., 1., 2., 2.],
                 [ 3., 3., 4., 4., 5., 5.],
                 [ 6., 6., 7., 7., 8., 8.],
                 [ 9., 9., 10., 10., 11., 11.]])
```

3.4.3 读取 CSV 文件

读取 CSV 文件格式：

```
loadtxt(fname, dtype = , comments = '#', delimiter = None, converters = None, skiprows = 0,
usecols = None, unpack = False, ndmin = 0, encoding = 'bytes')
```

主要参数及其说明见表 3-4。

表 3-4 主要参数及其说明

参　数	说　明
fname	str,读取的 CSV 文件名
delimiter	str,数据的分隔符
usecols	tuple(元组),执行加载数据文件中的哪些列
unpack	bool,是否将加载的数据拆分为多个组,True 表示拆,False 表示不拆
skiprows	int,跳过多少行,一般用于跳过前几行的描述性文字
encoding	bytes,编码格式


视频讲解

3.5 NumPy 中的数据统计与分析

在 NumPy 中,数组运算更为简捷而快速,通常比等价的 Python 方式快很多,尤其在处理数组统计计算与分析的情况下。

3.5.1 排序

NumPy 的排序方式有直接排序和间接排序。直接排序是对数据直接进行排序,间接排序是指根据一个或多个键值对数据集进行排序。在 NumPy 中,直接排序经常使用 sort 函数,间接排序经常使用 argsort 函数和 lexsort 函数。

sort 函数是最常用的排序方法,函数调用改变原始数组,无返回值。

格式：

```
numpy.sort(a,axis,kind,order)
```

主要参数及其说明见表 3-5。

表 3-5 sort 主要参数及其说明

参 数	说 明
a	要排序的数组
axis	使得 sort 函数可以沿着指定轴对数据集进行排序。axis＝1 为沿横轴排序；axis＝0 为沿纵轴排序；axis＝None，将数组平坦化之后进行排序
kind	排序算法，默认为 quicksort
order	如果数组包含字段，则是要排序的字段

【例 3-39】 使用 sort 函数进行排序。

```
In[39]:  arr = np.array([7,9,5,2,9,4,3,1,4,3])
         print('原数组:',arr)
         arr.sort()
         print('排序后:',arr)
Out[39]: 原数组: [7 9 5 2 9 4 3 1 4 3]
         排序后: [1 2 3 3 4 4 5 7 9 9]
```

【例 3-40】 带轴向参数的 sort 排序。

```
In[40]:  arr = np.array([[4,2,9,5],[6,4,8,3],[1,6,2,4]])
         print('原数组:\n',arr)
         arr.sort(axis = 1)              #沿横向排序
         print('横向排序后:\n',arr)
Out[40]: 原数组:
         [[4 2 9 5]
         [6 4 8 3]
         [1 6 2 4]]
         横向排序后:
         [[2 4 5 9]
         [3 4 6 8]
         [1 2 4 6]]
```

使用 argsort 和 lexsort 函数，可以在给定一个或多个键时，得到一个由整数构成的索引数组，索引值表示数据在新的序列中的位置。

【例 3-41】 使用 argsort 函数进行排序示例。

```
In[41]   arr = np.array([7,9,5,2,9,4,3,1,4,3])
         print('原数组:',arr)
         print('排序后:',arr.argsort())
         #返回值为数组排序后的下标排列
         print('显示较大的 5 个数:',arr[arr.argsort()][-5:])
Out[41]  原数组: [7 9 5 2 9 4 3 1 4 3]
         排序后: [7 3 6 9 5 8 2 0 1 4]
         显示较大的 5 个数: [4 5 7 9 9]
```

【例 3-42】 使用 lexsort 排序示例。

```
In[42]   a = [1,5,7,2,3,-2,4]
         b = [9,5,2,0,6,8,7]
```

```
            ind = np.lexsort((b,a))
            print('ind:',ind)
            tmp = [(a[i],b[i])for i in ind]
            print('tmp:',tmp)
Out[42]     ind: [5 0 3 4 6 1 2]
            tmp: [(-2, 8), (1, 9), (2, 0), (3, 6), (4, 7), (5, 5), (7, 2)]
```

3.5.2 重复数据与去重

在数据统计分析中,需要提前将重复数据剔除。在 NumPy 中,可以通过 unique 函数找到数组中的唯一值并返回已排序的结果,其中的参数 return_counts 设置为 True 时可以返回每个取值出现的次数。

【例 3-43】 数组内数据去重。

```
In[43]      names = np.array(['红色','蓝色','蓝色','白色','红色','红色'])
            print('原数组:',names)
            print('去重后的数组:',np.unique(names))
            print('数据出现次数:',np.unique(names,return_counts = True))
Out[43]     原数组: ['红色' '蓝色' '蓝色' '白色' '红色' '红色']
            去重后的数组: ['白色' '红色' '蓝色']
            数据出现次数: (array(['白色', '红色', '蓝色'], dtype = '<U2'), array([1, 3, 2],
            dtype = int64))
```

统计分析中有时需要把一个数据重复若干次,在 NumPy 中主要使用 tile 和 repeat 函数实现数据重复。

tile 函数的格式:

```
numpy.tile(A,reps)
```

其中参数 A 表示要重复的数组,reps 表示重复次数。

【例 3-44】 使用 tile 函数实现数据重复。

```
In[44]:     arr = np.arange(5)
            print('原数组:',arr)
            wy = np.tile(arr,3)
            print('重复数据处理:\n',wy)
Out[44]:    原数组: [0 1 2 3 4]
            重复数据处理:
            [0 1 2 3 4 0 1 2 3 4 0 1 2 3 4]
```

repeat 函数的格式:

```
numpy.repeat(a,reps,axis = None)
```

其中,参数 a 是需要重复的数组元素,参数 reps 是重复次数,参数 axis 指定沿着哪个轴进行重复:axis=0 表示按行进行元素重复,axis=1 表示按列进行元素重复。

【例 3-45】 使用 repeat 函数实现数据重复。

```
In[45]:  arr = np.arange(5)
         print('原数组:',arr)
         wy = np.tile(arr,3)
         print('重复数据处理:\n',wy)
         arr2 = np.array([[1,2,3],[4,5,6]])
         print('重复数据处理:\n',arr2.repeat(2,axis = 0))
Out[45]: 原数组: [0 1 2 3 4]
         重复数据处理:
         [0 1 2 3 4 0 1 2 3 4 0 1 2 3 4]
         重复数据处理:
         [[1 2 3]
         [1 2 3]
         [4 5 6]
         [4 5 6]]
```

3.5.3 常用统计函数

NumPy 中提供了很多用于统计分析的函数，常见的有 sum、mean、std、var、min 和 max 等。几乎所有的统计函数在针对二维数组时都需要注意轴的概念。当 axis 参数为 0 时，表示沿着纵轴进行计算；当 axis=1 时表示沿横轴进行计算。

【例 3-46】 NumPy 中常用函数的使用。

```
In[46]:  arr = np.arange(20).reshape(4,5)
         print('创建的数组:\n',arr)
         print('数组的和:',np.sum(arr))
         print('数组纵轴的和:',np.sum(arr,axis = 0))
         print('数组横轴的和:',np.sum(arr,axis = 1))
         print('数组的均值:',np.mean(arr))
         print('数组横轴的均值:',np.mean(arr,axis = 1))
         print('数组的标准差:',np.std(arr))
         print('数组横轴的标准差:',np.std(arr,axis = 1))
Out[46]: 创建的数组:
         [[ 0 1 2 3 4]
         [ 5 6 7 8 9]
         [10 11 12 13 14]
         [15 16 17 18 19]]
         数组的和: 190
         数组纵轴的和:[30 34 38 42 46]
         数组横轴的和:[10 35 60 85]
         数组的均值: 9.5
         数组横轴的均值:[ 2. 7. 12. 17.]
         数组的标准差: 5.766281297335398
         数组横轴的标准差:[1.41421356 1.41421356 1.41421356 1.41421356]
```

3.6 本章小结

本章重点介绍了 NumPy 的基础内容，主要包括数组及其索引、数组运算、数组读/写及常用的统计与分析方法。

3.7 本章习题

一、单项选择题

1. Numpy 提供了两种基本对象，一种是 ndarray，另一种是(　　)。

A. array　　B. func　　C. matrix　　D. Series

2. 创建一个 3×3 的数组，下列代码中错误的是(　　)。

A. np.arange(0,9).reshape(3,3)　　B. np.eyes(3)

C. np.random.random([3,3,3])　　D. np.mat("1,2,3;4,5,6;7,8,9")

3. Numpy 中统计数组元素个数的方法是(　　)。

A. ndim　　B. shape　　C. size　　D. itemsize

二、填空题

1. 有 arr=np.arange(12).reshape(3,4)，则 arr[(0,1),(1,3)]对应的值是_____和_____；arr[1:2,(0, 3)]对应的元素是_____和_____；arr.ndim 的值是_____。

2. 对于上题中的 arr，若定义 mask = np.array([1,0,1],dtype = np.bool)，则 arr[mask,1] 对应的元素是_____和_____。

3. 已知 a=np.arange(8).reshape(2,4)，np.hsplit(a,2)的返回值是_____。

4. 数组转置是数据重塑的一种特殊形式，可以通过_____方法或数组的 T 属性实现。

5. 创建一个范围为(0,1)的长度为 12 的等差数列的语句是_____。

6. Numpy 中数组的方法 sort、argsort 和 lexsort 分别是指_____、_____和_____。

7. 实现创建一个 10×10 的 ndarray 对象，满足矩阵边界全为 1，里面全为 0，对应的代码是_____。

8. 将数组 arr 中所有的奇数置为−1 对应的语句是_____。

3.8 本章实训

本实训读取 iris 数据集中鸢尾花的萼片、花瓣长度数据(已保存为 CSV 格式)，并对其进行排序、去重，并求出和、累积和、均值、标准差、方差、最小值、最大值。

1. 导入模块

```
In[1]:  import numpy as np
        import csv
```

2. 获取数据

```
In[2]:  iris_data = []
        with open("iris.csv") as csvfile:
                # 使用 csv.reader 读取 csvfile 中的文件
                csv_reader = csv.reader(csvfile)
                # 读取第一行各列的标题
                birth_header = next(csv_reader)
                # 将 csv 文件中的数据保存到 birth_data 中
                for row in csv_reader:
                    iris_data.append(row)
Out[2]: [['1', '5.1', '3.5', '1.4', '0.2', 'setosa'],
        ['2', '4.9', '3', '1.4', '0.2', 'setosa'],
        ['3', '4.7', '3.2', '1.3', '0.2', 'setosa'],
        …
        ['148', '6.5', '3', '5.2', '2', 'virginica'],
        ['149', '6.2', '3.4', '5.4', '2.3', 'virginica'],
        ['150', '5.9', '3', '5.1', '1.8', 'virginica']]
```

3. 数据清理：去掉索引号

```
In[3]:  iris_list = []
        for row in iris_data:
            iris_list.append(tuple(row[1:]))
        iris_list
Out[3]: [('5.1', '3.5', '1.4', '0.2', 'setosa'),
        ('4.9', '3', '1.4', '0.2', 'setosa'),
        ('4.7', '3.2', '1.3', '0.2', 'setosa'),
        ('4.6', '3.1', '1.5', '0.2', 'setosa'),
        ('5', '3.6', '1.4', '0.2', 'setosa'),
        …
```

4. 数据统计

(1) 创建数据类型。

```
In[4]:  datatype = np.dtype([("Sepal.Length", np.str_, 40),
                             ("Sepal.Width", np.str_, 40),
                             ("Petal.Length",np.str_, 40),
                             ("Petal.Width", np.str_, 40),
                             ("Species",np.str_, 40)] )
        print(datatype)
Out[4]: [('Sepal.Length', '<U40'),
```

```
        ('Sepal.Width', '<U40'),
        ('Petal.Length', '<U40'),
        ('Petal.Width', '<U40'),
        ('Species', '<U40')]
```

(2) 创建二维数组。

```
In[5]:   iris_data = np.array(iris_list,dtype = datatype)
         iris_data
Out[5]:  array([('5.1', '3.5', '1.4', '0.2', 'setosa'),
                ('4.9', '3', '1.4', '0.2', 'setosa'),
                ('4.7', '3.2', '1.3', '0.2', 'setosa'),
                ('4.6', '3.1', '1.5', '0.2', 'setosa'),
                ('5', '3.6', '1.4', '0.2', 'setosa'),
                …])
```

(3) 将待处理数据的类型转化为 float 类型。

```
In[6]:   PetalLength = iris_data["Petal.Length"].astype(float)
         PetalLength
Out[6]:  array([1. , 1.1, 1.2, 1.2, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.4, 1.4,
                1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.5, 1.5,
                1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.6, 1.6,
                …])
```

(4) 排序。

```
In[7]:   np.sort(PetalLength)
Out[7]:  array([1. , 1.1, 1.2, 1.2, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.4, 1.4,
                1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4, 1.5, 1.5,
                …])
```

(5) 数据去重。

```
In[8]:   np.unique(PetalLength)
Out[8]:  array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.9, 3. , 3.3, 3.5, 3.6,
                3.7, 3.8, 3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9,
                5. , 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.3,
                6.4, 6.6, 6.7, 6.9])
```

(6) 对指定列求和、均值、标准差、方差、最小值及最大值。

```
In[9]:   np.sum(PetalLength)
Out[9]:  563.7
In[10]:  np.mean(PetalLength)
Out[10]: 3.7580000000000005
In[11]:  np.std(PetalLength)
Out[11]: 1.759404065775303
In[12]:  np.var(PetalLength)
Out[12]: 3.0955026666666665
In[13]:  np.min(PetalLength)
Out[13]: 1.0
In[14]:  np.max(PetalLength)
Out[14]: 6.9
```