第3章 numpy计算

NumPy（Numerical Python的简称）是高性能科学计算和数据分析的基础包。部分功能如下：

- ndarray，一个具有矢量算术运算和复杂广播能力的快速且节省空间的多维数组。
- 用于对整组数据进行快速运算的标准数学函数（无需编写循环）。
- 用于读写磁盘数据的工具以及用于操作内存映射文件的工具。
- 线性代数、随机数生成以及傅里叶变换功能。
- 用于集成由C、C++、Fortran等语言编写的代码的工具。

最后一点也是从生态系统角度来看最重要的一点。由于NumPy提供了一个简单易用的C API，因此很容易将数据传递给由低级语言编写的外部库，外部库也能以NumPy数组的形式将数据返回给Python。这个功能使Python成为一种包装C/C++/Fortran历史代码库的选择，并使被包装库拥有一个动态的、易用的接口。

本章主要分为以下几个部分进行介绍：

一、NumPy的ndarray

- 创建ndarray
- ndarray的数据类型
- ndarray 对象常用方法
- 基本的切片与索引
- 高级索引
- 整数索引
- 布尔索引
- 花式索引
- 按条件返回索引 二、通用函数：快速的元素级数组函数
- 一元通用函数
- 二元通用函数
- 执行特定矢量化运算的特殊ufunc方法 三、统计方法 四、排序 五、数据的唯一化和集合运算 六、随机数生成 七、NumPy 矩阵库(Matrix)及其计算
- 创建矩阵
- 矩阵运算 八、高级数组操作
- 反转、转置数组
- 数组重塑
- 数组的合并和拆分
- 数组元素的添加与删除 九、广播

[1]
```
import numpy.matlib
import numpy as np
import pandas as pd
```

一、NumPy的ndarray

一种多维数组对象NumPy最重要的一个特点就是其N维数组对象（即ndarray），该对象是一个快速而灵活的大数据集容器。本部分主要介绍ndarray以及基本的切片与索引。

ndarray是一系列同类型数据的集合，以 0 下标为开始进行集合中元素的索引。
ndarray 对象是用于存放同类型元素的多维数组。ndarray 中的每个元素在内存中都有相同存储大小的区域。

创建ndarray

创建数组最简单的办法就是使用array函数。它接受一切序列型的对象（包括其他数组），然后产生一个新的含有传入数据的NumPy数组。
numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)
参数说明：
object：数组或嵌套的数列
dtype：数组元素的数据类型，可选
copy：对象是否需要复制，可选
order：创建数组的样式，C为行方向，F为列方向，A为任意方向（默认）
subok：默认返回一个与基类类型一致的数组
ndmin：指定生成数组的最小维度
一般情况下前两个参数是必须有的，其余参数较少使用。

[5]
```
data1=[6,7.5,8,0,1]
arr1=np.array(data1)
arr1
```

```
array([6. , 7.5, 8. , 0. , 1. ])
```

除此之外还有一些函数也可以创建ndarray
从已有数组创建： numpy.asarray(a, dtype = None, order = None)
未初始化数组：numpy.empty(shape, dtype = float, order = 'C')
全0数组：numpy.zeros(shape, dtype = float, order = 'C')
全1数组：numpy.ones(shape, dtype = None, order = 'C')
数值范围数组：numpy.arange(start, stop, step, dtype)
对角数组：numpy.eye(N, M=None, k=0, dtype=, order='C')

其中：
shape: 数组形状
dtype: 数据类型
order: C和F,分别代表行优先和列优先。
start: 起始值，默认为0
stop: 终止值（不包含）
step: 步长，默认为1
dtype: 返回ndarray的数据类型，默认使用输入数据的类型。
num: 要生成的等步长的样本数量，默认为50

[7]
```
print ("np.zeros((2,2)):\n", np.zeros((2,2)))
print ("np.ones((2,2)):\n", np.ones((2,2)))
print ("np.eye((2)):\n", np.eye((2)))
print ("np.random.random((2,2)):\n", np.random.random((2,2)))
```

```
np.zeros((2,2)):
 [[0. 0.]
 [0. 0.]]
np.ones((2,2)):
 [[1. 1.]
 [1. 1.]]
np.eye((2)):
 [[1. 0.]
 [0. 1.]]
np.random.random((2,2)):
 [[0.57223409 0.08745789]
 [0.54097632 0.87183112]]
```

ndarray的数据类型

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

[8]
```
arr1=np. array([1,2,3], dtype=np. float64)
arr2=np. array([1,2,3], dtype=np. int32)
print(arr1.dtype,arr2.dtype)
```

```
float64 int32
```

[11]
```
#你可以通过ndarray的astype方法显式地转换其dtype
arr=np.array([1,2,3,4,5])
print(arr.dtype)
float_arr=arr.astype(np.float64)
print(float_arr.dtype)
```

```
int32
float64
```

ndarray 对象常用方法

```
ndarray.ndim: 秩，即轴的数量或维度的数量
ndarray.shape: 数组的维度，对于矩阵，n 行 m 列
ndarray.size: 数组元素的总个数，相当于 .shape 中 n*m 的值
ndarray.dtype: ndarray 对象的元素类型
ndarray.tolist: narray对象转化为list
ndarray.newaxis:给原数组增加一个维度
```

[13]
```
x=np.array([1,2,3,4,5])
# 秩
print ("x ndim: ", x.ndim)
# 维度
print ("x shape:", x.shape)
# 数组元素的总个数
print ("x size: ", x.size)
# ndarray 对象的元素类型
print ("x dtype: ", x.dtype)
```

```
x ndim:  1
x shape: (5,)
x size:  5
x dtype:  int32
```

[14]
```
# 1-D
x = np.array([1.3 , 2.2 , 1.7])
print ("x: ", x)
print ("x ndim: ", x.ndim)
print ("x shape:", x.shape)
print ("x size: ", x.size)
print ("x dtype: ", x.dtype) # 注意 float datatype
```

```
x:  [1.3 2.2 1.7]
x ndim:  1
x shape: (3,)
x size:  3
x dtype:  float64
```

[15]
```
# 3-D  (矩阵)
```

```
x = np.array([[[1,2,3], [4,5,6], [7,8,9]]])
print ("x:\n", x)
print ("x ndim: ", x.ndim)
print ("x shape:", x.shape)
print ("x size: ", x.size)
print ("x dtype: ", x.dtype)
```

```
x:
 [[[1 2 3]
  [4 5 6]
  [7 8 9]]]
x ndim:  3
x shape: (1, 3, 3)
x size:  9
x dtype:  int32
```

[16]
```
#array转化为list
a = np.array([1,2])
b = a.tolist()
a,b
```

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

[17]
```
#增加一个维度 []→[[]]
x = np.random.randint(1, 8, size=5)
x1 = x[np.newaxis, :]
x,x1
```

```
(array([2, 1, 2, 6, 4]), array([[2, 1, 2, 6, 4]]))
```

基本的切片与索引

ndarray对象的内容可以通过索引或切片来访问和修改。
与 Python 中 list 的切片操作一样，ndarray 数组可以基于 0 - n 的下标进行索引，切片对象可以通过内置的 slice 函数从原数组中切割出一个新数组。
slice(start,stop , step)

[18]
```
# 索引
x = np.array([1, 2, 3])
print ("x[0]: ", x[0])
```

```
x[0] = 0
print ("x: ", x)
```

```
x[0]:  1
x:  [0 2 3]
```

[19]
```
# slice函数切片
a = np.arange(10)
print(a)
s = slice(2,7,2)   # 从索引2开始到索引7停止 ， 间隔为2
print (a[s])
```

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

[20]
```
# 切片
#冒号 :的解释
#如果只放置一个参数，如 [1]，将返回与该索引相对应的单个元素。
#如果为 [1:]，表示从该索引[1]开始以后的所有项都将被提取。
#如果使用了两个参数，如 [1:3]，那么则提取两个索引(不包括停止索引)之间的项，即[1][2]
x = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print (x)
print ("x column 1: ", x[:, 1])
print ("x row 0: ", x[0, :])
print ("x rows 0,1,2 & cols 1,2: \n", x[:3, 1:3])
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
x column 1:  [ 2  6 10]
x row 0:  [1 2 3 4]
x rows 0,1,2 & cols 1,2:
 [[ 2  3]
 [ 6  7]
 [10 11]]
```

高级索引

NumPy 比一般的 Python 序列提供更多的索引方式。除了用整数和切片的索引外，数组可以由整数数组索引、布尔索引等。

布尔索引通过布尔运算（如：比较运算符）来获取符合指定条件的元素的数组。

花式索引根据索引数组的值作为目标数组的某个轴的下标来取值。对于使用一维整型数组作为索引，如果目标是一维数组，那么索引的结果就是对应位置的元素；如果目标是二维数组，那么就是对应下标的行。花式索引跟切片不一样，它总是将数据复制到新数组中。

整数索引

[21]
```
print (x)
rows_to_get = np.arange(len(x))
print ("rows_to_get: ", rows_to_get)
cols_to_get = np.array([0, 2, 1])
print ("cols_to_get: ", cols_to_get)
print ("indexed values: ", x[rows_to_get, cols_to_get])
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
rows_to_get:  [0 1 2]
cols_to_get:  [0 2 1]
indexed values:  [ 1  7 10]
```

布尔索引

[22]
```
# 布尔值索引
x = np.array([[1,2], [3, 4], [5, 6]])
print ("x:\n", x)
print ("x > 2:\n", x > 2)
print ("x[x > 2]:\n", x[x > 2])
```

```
x:
 [[1 2]
 [3 4]
 [5 6]]
x > 2:
 [[False False]
 [ True  True]
 [ True  True]]
x[x > 2]:
 [3 4 5 6]
```

花式索引

[23]
```
# 花式索引
x=np.arange(32).reshape((8,4))
print("x:\n",x)
print ("传入顺序索引数组:\n",x[[4,2,1,7]])
print ("传入倒序索引数组:\n",x[[-4,-2,-1,-7]])
print ("传入多个索引数组:\n",x[np.ix_([1,5,7,2],[0,3,1,2])])
```

```
x:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]]
传入顺序索引数组:
 [[16 17 18 19]
 [ 8  9 10 11]
 [ 4  5  6  7]
 [28 29 30 31]]
传入倒序索引数组:
 [[16 17 18 19]
 [24 25 26 27]
 [28 29 30 31]
 [ 4  5  6  7]]
传入多个索引数组:
 [[ 4  7  5  6]
 [20 23 21 22]
 [28 31 29 30]
 [ 8 11  9 10]]
```

按照条件返回索引

numpy.argmax() 和 numpy.argmin()函数分别沿给定轴返回最大和最小元素的索引。
numpy.nonzero() 函数返回输入数组中非零元素的索引。
numpy.where() 函数返回输入数组中满足给定条件的元素的索引。

[24]
```
a = np.array([[30,40,70],[80,20,10],[50,90,60]])
print ("数组展开最大元素的索引：",np.argmax(a))
print ("数组按列最大元素的索引：",np.argmax(a, axis =  0))
print ("数组按行最大元素的索引：",np.argmax(a, axis =  1))
print ("数组展开最大元素的索引：",np.argmin(a))
```

```
print ("数组按列最大元素的索引：",np.argmin(a, axis =  0))
print ("数组按行最大元素的索引：",np.argmin(a, axis =  1))
print ("数组中非零元素的索引：",np.nonzero (a))
print ("数组中大于50的索引：",np.where(a > 50))
print ('使用这些索引来获取满足条件的元素：',a[np.where(a > 50)])
```

```
数组展开最大元素的索引：  7
数组按列最大元素的索引：  [1 2 0]
数组按行最大元素的索引：  [2 0 1]
数组展开最大元素的索引：  5
数组按列最大元素的索引：  [0 1 1]
数组按行最大元素的索引：  [0 2 0]
数组中非零元素的索引：  (array([0, 0, 0, 1, 1, 1, 2, 2, 2], dtype=int64),
array([0, 1, 2, 0, 1, 2, 0, 1, 2], dtype=int64))
数组中大于50的索引：  (array([0, 1, 2, 2], dtype=int64), array([2, 0, 1, 2],
dtype=int64))
使用这些索引来获取满足条件的元素：  [70 80 90 60]
```

二、通用函数：快速的元素级数组函数

通用函数（即ufunc）是一种对ndarray中的数据执行元素级运算的函数。你可以将其看做简单函数（接受一个或多个标量值，并产生一个或多个标量值）的矢量化包装器。本部分主要介绍一元、二元通用函数以及执行特定矢量化运算的特殊方法。

一元通用函数

```
abs、fabs                    计算整数、浮点数或复数的绝对值。对于非复数值，可以使用更
快的fabs
sqrt                      计算各元素的平方根。相当于arr**0.5
square                      计算各元素的平方。相当于arr**2
exp                     计算各元素的指数e
log、log10、log2、loglp        分别为自然对数（底数为e）、底数为10的log、底数为2的
log、log（1+x）
sign                      计算各元素的正负号：1（正数）、0（零）、-1（负数）
ceil                      计算各元素的ceiling值，即大于等于该值的最小整数
floor                      计算各元素的floor值，即小于等于该值的最大整数
rint                      将各元素值四舍五入到最接近的整数，保留dtype
modf                      将数组的小数和整数部分以两个独立数组的形式返回
isnan                      返回一个表示“哪些值是NaN（这不是一个数字）”的布尔型数组
isfinite、isinf                分别返回一个表示“哪些元素是有穷的（非inf，非
NaN）”或“哪些元素是无穷的”的布尔型数组
cos、cosh、sin、sinh、tan、tanh  普通型和双曲型三角函数
arccos、arccosh、arcsin、        反三角函数
```

arcsinh、arctan、arctanh logical_not 计算各元素notx的真值。相当于-arr

[31]
```
a = np.array([1.0,5.55,123,-0.567,25.532])
print  ('原数组：',a)
print('绝对值：',np.abs(a))
print('平方根：',np.sqrt(a))
print('平方：',np.square(a))
print ('保留一位小数：',np.around(a, decimals =  1))
print ('向下取整：',np.floor(a))
print ('向上取整：',np.ceil(a))
```

```
原数组： [  1.      5.55  123.     -0.567  25.532]
绝对值： [  1.      5.55  123.      0.567  25.532]
平方根： [ 1.          2.3558438  11.09053651         nan  5.05291995]
平方： [1.00000000e+00 3.08025000e+01 1.51290000e+04 3.21489000e-01
 6.51883024e+02]
保留一位小数： [  1.    5.6 123.   -0.6  25.5]
向下取整： [  1.   5. 123.  -1.  25.]
向上取整： [  1.   6. 123.  -0.  26.]
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: RuntimeWarning:
invalid value encountered in sqrt
  after removing the cwd from sys.path.
```

[32]
```
#指数运算
x = [1, 2, 3]
print("x     =", x)
print("e^x   =", np.exp(x))
print("2^x   =", np.exp2(x))
print("3^x   =", np.power(3, x))
print("ln(x)    =", np.log(x))
print("log2(x)  =", np.log2(x))
print("log10(x) =", np.log10(x))
```

```
x     = [1, 2, 3]
e^x   = [ 2.71828183  7.3890561  20.08553692]
2^x   = [2. 4. 8.]
3^x   = [ 3  9 27]
ln(x)    = [0.         0.69314718 1.09861229]
log2(x)  = [0.        1.        1.5849625]
log10(x) = [0.         0.30103    0.47712125]
```

[35]
```
theta = np.linspace(0, np.pi, 3)
print("theta      = ", theta)
print("sin(theta) = ", np.sin(theta))
print("cos(theta) = ", np.cos(theta))
print("tan(theta) = ", np.tan(theta))
```

```
theta      =  [0.         1.57079633 3.14159265]
sin(theta) =  [0.0000000e+00 1.0000000e+00 1.2246468e-16]
cos(theta) =  [ 1.000000e+00  6.123234e-17 -1.000000e+00]
tan(theta) =  [ 0.00000000e+00  1.63312394e+16 -1.22464680e-16]
```

[36]
```
x = [-1, 0, 1]
print("x         = ", x)
print("arcsin(x) = ", np.arcsin(x))
print("arccos(x) = ", np.arccos(x))
print("arctan(x) = ", np.arctan(x))
```

```
x         =  [-1, 0, 1]
arcsin(x) =  [-1.57079633  0.          1.57079633]
arccos(x) =  [3.14159265 1.57079633 0.        ]
arctan(x) =  [-0.78539816  0.          0.78539816]
```

二元通用函数

```
+   np.add          加法(例如，1 + 1 = 2)
-   np.subtract       减法(例如，3 - 2 = 1)
-   np.negative       一元否定(例如，-2)
*   np.multiply        乘法(例如，2 * 3 = 6)
/   np.divide          除法(例如，3 / 2 = 1.5)
//  np.floor_divide     取整除法(例如，3 // 2 = 1)
**  np.power          指数(例如，2 ** 3 = 8)
%   np.mod          模数/余数(例如，9 % 4 = 1)
```

[37]
```
x = np.arange(4)
print("x     =", x)
print("       算术操作结果              对应Ufunc函数结果")
print("x + 5 =", x + 5,"              ",np.add(x,5))
print("x - 5 =", x - 5,"            ",np.subtract(x,5))
print("x * 2 =", x * 2,"              ",np.multiply(x,2))
print("x / 2 =", x / 2,"        ",np.divide(x,2))
print("x // 2 =", x // 2,"               ",np.floor_divide(x,2))
print("-x     = ", -x,"          ",np.negative(x))
print("x ** 2 = ", x ** 2,"              ",np.power(x,2))
print("x % 2  = ", x % 2,"              ",np.mod(x,2))
```

```
x     = [0 1 2 3]
       算术操作结果              对应Ufunc函数结果
x + 5 = [5 6 7 8]                    [5 6 7 8]
x - 5 = [-5 -4 -3 -2]                [-5 -4 -3 -2]
x * 2 = [0 2 4 6]                    [0 2 4 6]
```

```
x / 2 = [0.  0.5 1.  1.5]          [0.  0.5 1.  1.5]
x // 2 = [0 0 1 1]                 [0 0 1 1]
-x     =  [ 0 -1 -2 -3]            [ 0 -1 -2 -3]
x ** 2 =  [0 1 4 9]                [0 1 4 9]
x % 2  =  [0 1 0 1]                [0 1 0 1]
```

执行特定矢量化运算的特殊ufunc方法

NumPy的各个二元ufunc都有一些用于执行特定矢量化运算的特殊方法。
reduce（x）reduce接受一个数组参数，并通过一系列的二元运算对其值进行聚合（可指明轴向）
accumulate（x）聚合值，保留所有局部聚合结果
outer（x，y）对x和y中的每对元素应用原始运算。结果数组的形状为x.shape+y.shape
reduceat（x，bins）“局部”约简（也就是groupby）。约简数据的各个切片以产生聚合型数组

[38]
```
#我们可以用np.add.reduce对数组中各个元素进行求和。
#起始值取决于ufunc（对于add的情况，就是0）。如果设置了轴号，约简运算就会沿该轴向执行。
arr=np. arange(10)
np.add.reduce(arr),arr.sum()
```

```
(45, 45)
```

[42]
```
#用np.logical_and检查数组各行中的值是否是有序的：
arr=np.random.randn(5,5)
arr[::2].sort(1)#对部分行进行排序
arr[:,:-1]<arr[:,1:],np.logical_and.reduce(arr[:,:-1]
<arr[:,1:],axis=1)#logical_and.reduce跟all方法是等价的。
```

```
(array([[ True,  True,  True,  True],
        [False,  True,  True, False],
        [ True,  True,  True,  True],
        [ True, False,  True, False],
        [ True,  True,  True,  True]]),
 array([ True, False,  True, False,  True]))
```

[43]
```
#accumulate跟reduce的关系就像cumsum跟sum的关系那样。它产生一个跟原数组大小相同的中间“累计”值数组：
arr=np. arange(15). reshape((3,5))
```

```
np.add.accumulate(arr, axis=1)
```

```
array([[ 0,  1,  3,  6, 10],
       [ 5, 11, 18, 26, 35],
       [10, 21, 33, 46, 60]], dtype=int32)
```

[44]
```
#outer 计算两个数组的叉积
arr=np.arange(3).repeat([1,2,2])
arr,np.multiply.outer(arr, np. arange(5))
```

```
(array([0, 1, 1, 2, 2]), array([[0, 0, 0, 0, 0],
        [0, 1, 2, 3, 4],
        [0, 1, 2, 3, 4],
        [0, 2, 4, 6, 8],
        [0, 2, 4, 6, 8]]))
```

[55]
```
#outer输出结果的维度是两个输入数据的维度之和:
result = np.subtract.outer(np.random.randn(3, 4),
np.random.randn(5))
result.shape
```

```
(3, 4, 5)
```

[53]
```
#reduceat用于对数据各切片进行聚合的groupby运算。
arr = np.arange(10)
np.add.reduceat(arr, [0, 5, 8])
```

```
array([10, 18, 17], dtype=int32)
```

三、统计方法

```
NumPy 提供了很多统计函数，用于从数组中查找最小元素，最大元素，百分位标准差和方差等。
numpy.min()          求最小值
numpy.amin()       用于计算数组中的元素沿指定轴的最小值。
numpy.max()          求最大值
numpy.amax()       用于计算数组中的元素沿指定轴的最大值。
numpy.ptp()        函数计算数组中元素最大值与最小值的差（最大值 - 最小值）。
```

numpy.percentile(a, q, axis)百分位数是统计中使用的度量，表示小于这个值的观察值的百分比。

numpy.sum()	计算元素和
numpy.prod()	计算元素乘积
numpy.std()	计算标准差
numpy.var()	计算方差
numpy.any()	评估任何元素是否为真
numpy.all()	评估所有元素是否为真
numpy.median()	函数用于计算数组 a 中元素的中位数（中值）
numpy.mean()	函数返回数组中元素的算术平均值。
numpy.average()	函数根据在另一个数组中给出的各自的权重计算数组中元素的加权平均值。
numpy.cumsum()	所有元素的累计和
numpy.cumprod()	所有元素的累计积

[56]
```
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print(a)
print ("数组中的最小值：",np.amin(a),np.min(a))
print ("每行的最小值：",np.amin(a,1))
print ("每列的最小值：",np.amin(a,0))
print ("数组中的最大值：",np.amax(a),np.max(a))
print ("每行的最大值：",np.amax(a, axis =  1))
print ("每列的最大值：",np.amax(a, axis =  0))
print ("每行最大值与最小值的差：",np.ptp(a,axis =  1))
print ("在横行上求50% 的分位数：",np.percentile(a, 50,axis =  1))
print ("在横行上求中位数（中值）：",np.median(a, axis =  1))
print ("在横行上求算术平均值：",np.mean(a, axis =  1))
print ("所有元素的累计和：",np.cumsum(a))
print ("所有元素的累计积：",np.cumprod(a))
```

```
[[3 7 5]
 [8 4 3]
 [2 4 9]]
数组中的最小值： 2 2
每行的最小值： [3 3 2]
每列的最小值： [2 4 3]
数组中的最大值： 9 9
每行的最大值： [7 8 9]
每列的最大值： [8 7 9]
每行最大值与最小值的差： [4 5 7]
在横行上求50% 的分位数： [5. 4. 4.]
在横行上求中位数（中值）： [5. 4. 4.]
在横行上求算术平均值： [5. 5. 5.]
所有元素的累计和： [ 3 10 15 23 27 30 32 36 45]
所有元素的累计积： [     3     21    105    840   3360  10080  20160  80640
725760]
```

[57]
```
a = np.array([1,2,3,4])
print ('调用 average() 函数：')
```

```
print (np.average(a))
# 不指定权重时相当于 mean 函数
print ('加上权重后再次调用 average() 函数：')
print (np.average(a,weights = [4,3,2,1]))
# 如果 returned 参数设为 true，则返回权重的和
print ('权重的和：')
print (np.average([1,2,3,  4],weights =  [4,3,2,1], returned =
 True))
```

```
调用 average() 函数：
2.5
加上权重后再次调用 average() 函数：
2.0
权重的和：
(2.0, 10.0)
```

四、排序

跟Python内置的列表类型一样，NumPy数组也可以通过sort方法就地排序

[61]
```
arr=np.random.randn(8)
print('排序前：\n',arr)
arr.sort()
print('排序后：\n',arr)
```

```
排序前：
 [-1.09423727  0.16327629 -0.39470357 -0.27369042 -1.20361238 -0.4834135
 -0.93118398  0.14813413]
排序后：
 [-1.20361238 -1.09423727 -0.93118398 -0.4834135  -0.39470357
-0.27369042
  0.14813413  0.16327629]
```

[63]
```
#多维数组可以在任何一个轴向上进行排序，只需将轴编号传给sort即可
arr=np.random.randn(5,3)
print('排序前：\n',arr)
arr.sort(1)
print('排序后：\n',arr)
```

```
排序前：
 [[-0.15975062 -0.32468362 -0.1220071 ]
 [-0.71144555  1.22757902  0.86319888]
 [ 0.25438513 -0.30601358  1.29707533]
 [-0.71755231 -0.29995742  0.5938006 ]
 [-0.56019644  1.04587863  0.19253527]]
```

排序后：
```
[[-0.32468362 -0.15975062 -0.1220071 ]
 [-0.71144555  0.86319888  1.22757902]
 [-0.30601358  0.25438513  1.29707533]
 [-0.71755231 -0.29995742  0.5938006 ]
 [-0.56019644  0.19253527  1.04587863]]
```

[66]
```
#np.sort返回的是数组的已排序副本，而就地排序则会修改数组本身。计算数1分位数最简单的办法是对其进行排序，然后选取特定位置的值：
large_arr=np.random.randn(1000)
large_arr.sort()
large_arr[int(0.05*len(large_arr))]#5%分位数
```

```
-1.7198952770284313
```

五、数据的唯一化和集合运算

NumPy提供了一些针对一维ndarray的基本集合运算。最常用的可能要数np.unique了，它用于找出数组中的唯一值并返回已排序的结果。

[67]
```
a = np.array([1,2,3,4,5,6])
b = np.array([[3,7,5,5,5,3]])
print("a中的唯一元素:",np.unique(b))
```

```
a中的唯一元素: [3 5 7]
```

集合运算函数
intersect1d（x，y）计算x和y中的公共元素，并返回有序结果
union1d（x，y）计算x和y的并集，并返回有序结果
in1d（x，y）得到一个表示“x的元素是否包含于y”的布尔型数组
setdiff1d（x，y）集合的差，即元素在x中且不在y中
setxor1d（x，y）集合的对称差，即存在于一个数组中但不同时存在于两个数组中的元素

[68]
```
print("a和b中的公共元素：",np.intersect1d(a,b))
print("a的元素是否包含于b",np.in1d(a,b))
```

```
a和b中的公共元素：  [3 5]
a的元素是否包含于b [False False  True False  True False]
```

六、随机数生成

numpy.random模块对Python内置的random进行了补充，增加了一些用于高效生成多种概率分布的样本值的函数。

部分numpy.random函数

numpy.seed() 随机数生成种子
numpy.permutation() 返回一个序列的随机排列或返回一个随机排列的范围
numpy.shuffle() 对一个序列就地随机排列
numpy.rand() 产生均匀分布的样本值
numpy.randint() 从给定的上下限范围内随机选取整数
numpy.randn() 产生生态分布，平均值为0，标准差为1的样本值
numpy.binomial() 产生二项分布的样本值
numpy.normal() 产生正态分布样本值
numpy.beta 产生beta分布的样本值
numpy.chisqure() 产生卡方分布的样本值
numpy.gamma() 产生gamma分布的样本值
numpy.uniform() 产生[0,1)均匀分布的样本值

[69]
```
print("正态分布 (loc均值，scale标准差，size数量):\n",np.random.normal(loc = 0,scale=1,size =16))
print("正态分布4*4:\n",np.random.normal(size = (4,4)))
```

```
正态分布 (loc均值，scale标准差，size数量):
 [ 0.46384438 -2.31677463  0.84148649  1.39812905 -0.2515894
0.61764442
  0.44076124 -1.07475377 -2.83993261  0.55597439 -0.38872308 -0.70239379
 -0.5789592  -2.41054828  0.92540134  1.42467094]
正态分布4*4:
 [[-0.07244142 -1.0878258   0.31374739  0.24233281]
 [ 0.35617531  0.7546339   1.14400791 -0.51559365]
 [ 1.41558495 -1.1836136  -0.70112879  1.00018804]
 [-0.36122484 -0.1997344  -1.31597231 -0.93843363]]
```

七、NumPy 矩阵库(Matrix)及其计算

本部分主要介绍矩阵及其相关的线性代数的计算

NumPy 中包含了一个矩阵库numpy.matlib，该模块中的函数返回的是一个矩阵，而不是 ndarray 对象。

一个m*n的矩阵是一个由行m（row）列n（column）元素排列成的矩形阵列。
矩阵里的元素可以是数字、符号或数学式。

创建矩阵

numpy.matlib.empty(shape, dtype, order)函数返回一个新的矩阵
numpy.matlib.zeros((2,3))创建一个以 0 填充的矩阵
numpy.matlib.ones((2,3))创建一个以 1 填充的矩阵
numpy.matlib.eye(n, M,k, dtype)返回一个矩阵，对角线元素为 1，其他位置为零
numpy.matlib.identity()返回给定大小的单位矩阵
numpy.matlib.rand() 函数创建一个给定大小的矩阵，数据是随机填充的。
参数说明：
shape: 定义新矩阵形状的整数或整数元组
Dtype: 可选，数据类型
order: C（行序优先） 或者 F（列序优先）
n: 返回矩阵的行数
M: 返回矩阵的列数，默认为 n
k: 对角线的索引

[71]
```
print("2*3矩阵：\n",np.matlib.empty((2,3)))
print("2*3矩阵：\n",np.matlib.zeros((2,3)))
print("2*3矩阵：\n",np.matlib.ones((2,3)))
print("3*4矩阵：\n",np.matlib.eye(n =  3, M =  4, k =  0, dtype =
 float))
print("大小为 5，类型位浮点型的矩阵：\n",np.matlib.identity(5, dtype =
 float))
print("3*3随机填充数据矩阵：\n",np.matlib.rand(3,3))
```

```
2*3矩阵：
 [[4.24399158e-314 8.48798317e-314 1.27319747e-313]
 [1.69759663e-313 2.12199579e-313 2.54639495e-313]]
2*3矩阵：
 [[0. 0. 0.]
 [0. 0. 0.]]
2*3矩阵：
 [[1. 1. 1.]
 [1. 1. 1.]]
3*4矩阵：
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
大小为 5，类型位浮点型的矩阵：
 [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
```

```
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
3*3随机填充数据矩阵:
 [[0.68154925 0.8388852  0.2949826 ]
 [0.96304809 0.27917241 0.08094938]
 [0.65357831 0.2416451  0.93637365]]
```

[72]
```
#矩阵总是二维的，而 ndarray 是一个 n 维数组。 两个对象都是可互换的。
i = np.matrix('1,2;3,4')
i
```

```
matrix([[1, 2],
        [3, 4]])
```

[73]
```
j = np.asarray(i)
j
```

```
array([[1, 2],
       [3, 4]])
```

矩阵运算

线性代数（如矩阵乘法、矩阵分解、行列式以及其他方阵数学等）是任何数组库的重要组成部分。

NumPy 提供了线性代数函数库 linalg，该库包含了线性代数所需的所有功能
diag()　以一维数组的形式返回方阵的对角线（或非对角线）元素，或将一维数组转换为方阵（非对角线元素为0）
dot()　两个数组的点积，即元素对应相乘。
inner()　两个数组的内积
matmul()　两个数组的矩阵积
det()　计算输入矩阵的行列式。
inv()　计算矩阵的乘法逆矩阵
trace()　计算对角线元素的和
eig()　计算方阵的本征值和本征向量
pinv()　计算矩阵的Moore-Penrose伪逆qr计算QR分解
solve()　求解线性矩阵方程Ax=b，其中A为一个方阵
svd()　计算奇异值分解（SVD）
lstsq()　计算Ax=b的最小二乘解

[74]
```
a = np.array([[1,2],[3,4]])
```

```
b = np.array([[11,12],[13,14]])
print("a:\n",a)
print("b:\n",b)
print ("点积:\n",np.dot(a,b))
print ("内积:\n",np.inner(a,b))
print ("矩阵积:\n",np.matmul(a,b))
print ("行列式:\n",np.linalg.det(a))
print ('a 的逆：\n',np.linalg.inv(a) )
print ('a 的对角线和：\n',np.linalg.trace(a))
print ('a 的本征值和本征向量：\n',np.linalg.eig(a))
print ('a 的伪逆矩阵：\n',np.linalg.pinv(a))
```

```
a:
 [[1 2]
 [3 4]]
b:
 [[11 12]
 [13 14]]
点积:
 [[37 40]
 [85 92]]
内积:
 [[35 41]
 [81 95]]
矩阵积:
 [[37 40]
 [85 92]]
行列式:
 -2.0000000000000004
a 的逆：
 [[-2.   1. ]
 [ 1.5 -0.5]]
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call
last)
<ipython-input-74-a223fabc1ced> in <module>()
      8 print ("行列式:\n",np.linalg.det(a))
      9 print ('a 的逆：\n',np.linalg.inv(a) )
---> 10 print ('a 的对角线和：\n',np.linalg.trace(a))
     11 print ('a 的本征值和本征向量：\n',np.linalg.eig(a))
     12 print ('a 的伪逆矩阵：\n',np.linalg.pinv(a))

AttributeError: module 'numpy.linalg' has no attribute 'trace'
```

[75]
```
#slove
a = np.array([[1,1,1],[0,2,5],[2,5,-1]])
b = np.array([[6],[-4],[27]])
print ('计算：A^(-1)B：')
x = np.linalg.solve(a,b)
print (x)
```

```
计算：A^(-1)B：
[[ 5.]
 [ 3.]
 [-2.]]
```

[76]
```
#svd
S = np.zeros([5,5])
A=np.random.randint(1,25,[5,5])
u,sigma,vt = np.linalg.svd(A)
print(A)
```

```
[[ 6 24 16 22 17]
 [20 14 11 14 15]
 [ 5  9  3 14  9]
 [ 9  2 16  3 16]
 [24 21 20  7  8]]
```

[77]
```
#lstsq，传入A.T和观测值里的变量即可求得f(x)=a+bx里的a和b。a和b记录在lstsq函数的第一个返回值里
#第一个返回值sol共有两个值，sol[0]即是估计出来的f(x)=a+bx里a，sol[1]代表f(x)=a+bx里b。因此f(x)为：y_fit = sol[0] + sol[1] * x
m = 100
x = np.linspace(-1, 1, m)
y_exact = 1 + 2 * x
xi = x + np.random.normal(0, 0.05, 100)
yi = 1 + 2 * xi + np.random.normal(0, 0.05, 100)
A = np.vstack([xi**0, xi**1])
sol, r, rank, s = np.linalg.lstsq(A.T, yi)    #求取各个系数大小
y_fit = sol[0] + sol[1] * x
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:9: FutureWarning:
`rcond` parameter will change to the default of machine precision times
``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  if __name__ == '__main__':
```

八、高级数组操作

除花式索引、切片、布尔条件取子集等操作之外，数组的操作方式还有很多。虽然pandas中的高级函数可以处理数据分析工作中的许多重型任务，但有时你还是需要编写一些在现有库中找不到的数据算法。本部分主要介绍转置数组、数组重塑、合并与拆分。

翻转、转置数组

numpy.transpose(arr, axes)对换数组的维度，如形状 (2,3,4) transpose 后就变成(4,3,2)。
ndarray.T 和transpose() 相同

[78]
```
# transpose,T
a = np.array([[1,2],[3,4]])
np.transpose(a),a.T
```

```
(array([[1, 3],
        [2, 4]]), array([[1, 3],
        [2, 4]]))
```

数组重塑

鉴于我们已经学过的有关NumPy数组的知识，当你知道“无需复制任何数据，数组就能从一个形状转换为另一个形状”时应该会感到有一点吃惊。只需向数组的实例方法reshape传入一个表示新形状的元组即可实现该目的。假设有一个一维数组，我们希望将其重新排列为一个矩阵：

[79]
```
#numpy.reshape(arr, newshape, order='C') 不改变数据的条件下修改形状
a = np.arange(8)
b = a.reshape(4,2)
a,b
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7]), array([[0, 1],
        [2, 3],
        [4, 5],
        [6, 7]]))
```

[80]
```
#与reshape将一维数组转换为多维数组的运算过程相反的运算通常称为扁平化（flattening）或散开（raveling）：
arr=np. arange(15). reshape((5,3))
arr,arr.ravel()
```

```
(array([[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11],
        [12, 13, 14]]),
 array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14]))
```

[82]
```
#如果没有必要，ravel不会产生源数据的副本（下面将详细介绍）。flatten方法的行为类似于ravel，只不过它总是返回数据的副本：
arr.flatten()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

数组的合并和拆分

numpy.concatenate((a1, a2, ...), axis)沿指定轴axis连接相同形状的两个或多个数组
numpy.stack(arrays, axis) 沿新轴axis连接数组序列
便捷化函数：vstack、hstack、dstack以（沿轴0）（沿轴1）（沿轴2）的方式对数组进行堆叠
numpy.split(ary, indices_or_sections, axis)将一个数组ary分割为indices_or_sections个子数组，axis是沿着哪个维度进行切向，默认为0，横向切分。为1时，纵向切分
便捷化函数：hsplit、vsplit、dsplit split分别沿轴0、轴1、轴2进行拆分

[83]
```
#concatenate
a = np.array([[1,2],[3,4]])
print ('第一个数组：')
print (a)

b = np.array([[5,6],[7,8]])
print ('第二个数组：')
print (b)

# 两个数组的维度相同
print ('沿轴 0 连接两个数组：')
print (np.concatenate((a,b)))
print ('沿轴 1 连接两个数组：')
print (np.concatenate((a,b),axis = 1))
```

```
第一个数组：
[[1 2]
 [3 4]]
第二个数组：
[[5 6]
 [7 8]]
沿轴 0 连接两个数组：
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
沿轴 1 连接两个数组：
[[1 2 5 6]
 [3 4 7 8]]
```

[84]

```
#stack
a = np.array([[1,2],[3,4]])

print ('第一个数组：')
print (a)

b = np.array([[5,6],[7,8]])
print ('第二个数组：')
print (b)

print ('沿轴 0 堆叠两个数组：')
print (np.stack((a,b),0))
print ('沿轴 1 堆叠两个数组：')
print (np.stack((a,b),1))
```

```
第一个数组：
[[1 2]
 [3 4]]
第二个数组：
[[5 6]
 [7 8]]
沿轴 0 堆叠两个数组：
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
沿轴 1 堆叠两个数组：
[[[1 2]
  [5 6]]

 [[3 4]
  [7 8]]]
```

[85]
```
#split
a = np.arange(9)
print ('第一个数组：')
print (a)

print ('将数组分为三个大小相等的子数组：')
b = np.split(a,3)
print (b)

print ('将数组在一维数组中表明的位置分割：')
b = np.split(a,[4,7])
print (b)
```

```
第一个数组：
[0 1 2 3 4 5 6 7 8]
将数组分为三个大小相等的子数组：
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]
将数组在一维数组中表明的位置分割：
[array([0, 1, 2, 3]), array([4, 5, 6]), array([7, 8])]
```

数组元素的添加与删除

numpy.append(arr, values, axis=None)在数组的末尾添加值。 追加操作会分配整个数组，并把原来的数组复制到新数组中。
numpy.insert(arr, obj, values, axis)在给定索引obj之前，沿给定轴axis在输入数组arr中插入值values
numpy.delete(arr, obj, axis)从输入数组arr中删除指定子数组的新数组

[86]
```
#append
a = np.array([[1,2,3],[4,5,6]])
print ('第一个数组：')
print (a)
print ('向数组添加元素：')
print (np.append(a, [7,8,9]))
```

```
第一个数组：
[[1 2 3]
 [4 5 6]]
向数组添加元素：
[1 2 3 4 5 6 7 8 9]
```

[87]
```
#insert
a = np.array([[1,2],[3,4],[5,6]])
print (a)
```

```
print (np.insert(a,3,[11,12]))
print (np.insert(a,1,[11],axis = 0))
```

```
[[1 2]
 [3 4]
 [5 6]]
[ 1  2  3 11 12  4  5  6]
[[ 1  2]
 [11 11]
 [ 3  4]
 [ 5  6]]
```

[88]
```
#delete
a = np.arange(12).reshape(3,4)
print (a)
print (np.delete(a,2))#删除展开后的第2列
print (np.delete(a,2,axis = 1))#删除第二列
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[ 0  1  3  4  5  6  7  8  9 10 11]
[[ 0  1  3]
 [ 4  5  7]
 [ 8  9 11]]
```

九、广播

广播(Broadcast)是 numpy 很重要的一个特性，它对不同形状(shape)的数组进行数值计算的方式， 通常在数组的算术运算对相应的元素进行。
如果两个数组 a 和 b 形状相同，即满足 a.shape == b.shape，那么 a*b 的结果就是 a 与 b 数组对应位相乘。这要求维数相同，且各维度的长度相同。
NumPy中的广播遵循一组严格的规则来确定两个数组之间的相互作用
规则1：如果这两个数组的维数不同，则维数较少的数组的形状为衬垫在它的前面(左边)。
规则2：如果两个数组的形状在任何维度中不匹配，则在该维度中形状等于1的数组将被拉伸以与另一个形状匹配。
规则3：如果在任何维度中，大小不一致，且两者都不等于1，则会引发错误。

[89]
```
#对于相同大小的数组，在逐个元素的基础上执行：
a = np.array([0, 1, 2])
b = np.array([5, 5, 5])
a + b
```

```
array([5, 6, 7])
```

[90]
```
#广播允许在不同大小的数组上执行这些类型操作
#例如，我们可以很容易地将标量(将其视为零维数组)添加到数组中：
a + 5
```

```
array([5, 6, 7])
```

5进入数组[5, 5, 5]，并加结果到a。
我们也可以将其扩展到更高维度的数组。

[91]
```
# 在将一维数组添加到二维数组时
M = np.ones((3, 3))
M
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

[92]
```
M + a
```

```
array([[1., 2., 3.],
       [1., 2., 3.],
       [1., 2., 3.]])
```

虽然这些例子相对容易理解，但更复杂的例子可能涉及两个数组的广播。

[93]
```
a = np.arange(3)
b = np.arange(3)[:, np.newaxis]
print(a)
print(b)
```

```
[0 1 2]
[[0]
 [1]
 [2]]
```

[94]
```
a + b
```

```
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])
```

广播实例与分析过程

[95]
```
#向一维数组中添加一个二维数组
M = np.ones((2, 3))
a = np.arange(3)
#两个数组都需要广播的例子
c = np.arange(3).reshape((3, 1))
d = np.arange(3)
```

思考：
数组的形状如下

```
M.shape = (2, 3)
a.shape = (3,)
c.shape = (3, 1)
d.shape = (3,)
```

我们通过规则1看到数组a,d维度较少，所以我们把1放在左边：

```
M.shape -> (2, 3)
a.shape -> (1, 3)
c.shape -> (3, 1)
d.shape -> (1, 3)
```

根据规则2，我们现在看到第一个维度不一致，因此我们扩展这个维度来匹配：

```
M.shape -> (2, 3)
a.shape -> (2, 3)
c.shape -> (3, 3)
d.shape -> (3, 3)
```

[96]
```
#[[1. 1. 1.]    +    [0 1 2]
# [1. 1. 1.]]
M + a
```

```
array([[1., 2., 3.],
       [1., 2., 3.]])
```

[97]
```
#[[0]
# [1]   +  [0 1 2]
# [2]]
c+d
```

```
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])
```

[98]
```
#形状不匹配的情况下又是如何呢？
M = np.ones((3, 2))
a = np.arange(3)
M,a
```

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

同样，我们将从写出数组的形状开始：

```
M.shape = (3, 2)
a.shape = (3,)

M.shape -> (3, 2)
a.shape -> (1, 3)

M.shape -> (3, 2)
a.shape -> (3, 3)
```
最终的形状不匹配，所以这两个数组是不兼容的，无法计算出结果

[99]
```
#[[1, 1],
#[1, 1],   +     [0, 1, 2]
#[1, 1]]
M + a
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
last)
<ipython-input-99-e8522c44aa47> in <module>()
      2 #[1, 1],   +     [0, 1, 2]
      3 #[1, 1]]
----> 4 M + a
```

```
ValueError: operands could not be broadcast together with shapes (3,2)
(3,)
```

[100]
```
#解决：还记得newaxis吗？
M + a[:, np.newaxis]
```

```
array([[1., 1.],
       [2., 2.],
       [3., 3.]])
```

[ ]