

pandas and numpy notebook(1)

author:xnchall

Blog:<http://www.cnblogs.com/xnchl/>

pandas和numpy学习系列，主要目的记录学习中的亮点，以及熟悉对jupyter的使用！

```
In [3]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1. Series是pandas最基本的1维数据形式，存储内容的数据类型任意。pd.Series(data, index=[])data可为字典、np中相关对象。index为数据索引，可以是列表。当有个性化索引时，会按照输入索引与data匹配数据，如果data没有匹配到就是设置为 缺省值NaN

```
In [4]: data = {"a":12, "b":55, "c":50}
pd.Series(data)
```

```
Out[4]: a    12
b    55
c    50
dtype: int64
```

```
In [5]: data1 = {'a': 10, 'b': 20, 'c': 30}
pd.Series(data1, index=['b', 'c', 'a', 'd'])
```

```
Out[5]: b    20.0
c    30.0
a    10.0
d     NaN
dtype: float64
```

2. ndarray是numpy中定义的多维数组[矩阵模块]。ndarray对象可以作为pandas中Series的data数据源进行转换。

ndarray 的一个特点是同构：即其中所有元素的类型必须相同。

NumPy数组的维数称为秩（rank），一维数组的秩为1，二维数组的秩为2，以此类推。在NumPy中，每一个线性的数组称为是一个轴（axes），秩其实是描述轴的数量。比如说，二维数组相当于是一个一维数组，而这个一维数组中每个元素又是一个一维数组。所以这个一维数组就是NumPy中的轴（axes），而轴的数量——秩，就是数组的维数。

```
In [6]: data2 = np.random.randn(5) # 一维随机数列表  
index = ['a', 'b', 'c', 'd', 'e'] # 指定索引  
  
pd.Series(data2, index)
```

```
Out[6]: a    0.483059  
       b    0.104168  
       c    0.413886  
       d   -1.115421  
       e    1.306849  
       dtype: float64
```

2.1 介绍一下Numpy的ndarray对象

```
In [99]: temp = [[1, 9, 6],[2, 8, 5],[3, 7, 4]]
x = np.array(temp)
print(x.T) #获得x的转置矩阵
print(x.size) #获得数组中元素的个数
print(x.ndim) #获得数组的维数
print(x.shape) #获得数组的 ( 行数 , 列数 )
x.flags #返回数组内部的信息
x.flat #将多维数组转化为1维数组 [1,9,6,2,8,5,3,7,4]
# for i in x.flat:
#     print(i)
# x.flat = 2;x#将值赋给1维数组, 再转化成有原有数组的大小形式
y=x.reshape(1,9)#数组转化为n行m列
print(y)
y.base #获得该数组基于另外一个对象数组而来, 如下, y是根据x而来
np.array(range(15)).reshape(3,5)
```

```
[[1 2 3]
 [9 8 7]
 [6 5 4]]
9
2
(3, 3)
[[1 9 6 2 8 5 3 7 4]]
```

```
Out[99]: array([[ 0,  1,  2,  3,  4],
               [ 5,  6,  7,  8,  9],
               [10, 11, 12, 13, 14]])
```

2.2 利用ndarray创建矩阵或者多维数组

```
In [93]: np.ndarray(shape=(2,3), dtype=int, buffer=np.array([1,2,3,4,5,6,7]), offset=0, order="C")
#创建多维数组形状shape,
#数据类型dtype,
#初始化数组buffer
#offset含义是 buffer中用于初始化数组的首个数据的偏移
#order含义是 'C':行优先; 'F':列优先
#order参数的C和F是numpy中数组元素存储区域的两种排列格式, 即C语言格式和Fortran语言格式。
```

```
Out[93]: array([[1, 2, 3],
               [4, 5, 6]])
```

2.3 ones、zeros、empty系列函数

某些时候，我们在创建数组之前已经确定了数组的维度以及各维度的长度。这时我们就可以使用numpy内建的一些函数来创建ndarray。例如：函数ones创建一个全1的数组、函数zeros创建一个全0的数组、函数empty创建一个内容随机的数组，在默认情况下，用这些函数创建的数组的类型都是float64，若需要指定数据类型，只需要闲置dtype参数即可

```
In [131]: #可以通过元组指定数组形状
a = np.ones(shape=(2,3))
print(a.dtype)
#也可以通过列表来指定数组形状，同时这里指定了数组类型
b = np.zeros(shape=[3,2], dtype=np.int64)
b.dtype
#函数empty创建一个内容随机的数组
c = np.empty((4,2))
c.dtype

#其他用法
f=[[1,2,3], [3,4,5]]
np.zeros_like(f)
np.ones_like(f)
np.empty_like(f)
```

float64

```
Out[131]: array([[1, 2, 3],
                 [3, 4, 5]])
```

2.4 arange、linspace与logspace

1> arange函数类似python中的range函数，通过指定初始值、终值以及步长（默认步长为1）来创建数组

2> linspace函数通过指定初始值、阈值以及元素个数来创建一维数组

3> logspace函数与linspace类似，只不过它创建的是一个等比数列，同样的也是一个一维数组

```
In [133]: np.arange(0, 10, 2)
```

```
Out[133]: array([0, 2, 4, 6, 8])
```

```
In [145]: np.linspace(0,12, 4)
```

```
Out[145]: array([ 0.,  4.,  8., 12.])
```

```
In [139]: np.logspace(0,2,3)
```

```
Out[139]: array([ 1., 10., 100.])
```

2.5 fromstring、fromfunction系列函数

1> fromstring函数从字符串中读取数据并创建数组

2> fromfunction函数由第一个参数作为计算每个数组元素的函数（函数对象或者lambda表达式均可），第二个参数为数组的形状

```
In [149]: str1 = "1,2,3,4,5"  
np.fromstring(str1, dtype=np.int64, sep=",")
```

```
Out[149]: array([1, 2, 3, 4, 5], dtype=int64)
```

```
In [152]: str2 = "1.01 2.23 3.53 4.76"  
np.fromstring(str2, dtype=np.float64, sep=" ")
```

```
Out[152]: array([1.01, 2.23, 3.53, 4.76])
```

```
In [153]: def func(i, j):  
           return (i+1)*(j+1)
```

```
In [155]: np.fromfunction(func, (2,3)) #dtype=np.float64
```

```
Out[155]: array([[1.,  2.,  3.],  
                [2.,  4.,  6.]])
```

```
In [156]: np.fromfunction(lambda i,j: i+j, (3,3), dtype=int)
```

```
Out[156]: array([[0, 1, 2],  
                [1, 2, 3],  
                [2, 3, 4]])
```

2.6 ndarray创建特殊的二维数组

np中matrix是继承于采用了ndarray建立的二维数组进行了封装的子类。特别注意，关于二维数组的创建，依旧是一个ndarray对象，而不是matrix对象。

```
In [178]: arr = np.arange(9).reshape((3,3))  
print(arr)
```

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

diag函数返回一个矩阵的对角线元素、或者创建一个对角阵，对角线由参数k控制 [这里的k就是列的索引]

```
In [179]: np.diag(arr) #array([0, 4, 8])  
np.diag(arr, k=1) #array([1, 5])  
np.diag(arr, k=-1) #array([3, 7])  
np.diag(np.diag(arr)) #array([[0, 0, 0], [0, 4, 0], [0, 0, 8]])  
np.diag(np.diag(arr), k=1)
```

```
Out[179]: array([[0, 0, 0, 0],  
                [0, 0, 4, 0],  
                [0, 0, 0, 8],  
                [0, 0, 0, 0]])
```

diagflat函数以输入作为对角线元素，创建一个矩阵，对角线有参数k控制

```
In [182]: np.diagflat([1,2,3], k=1)
```

```
Out[182]: array([[0, 1, 0, 0],  
                [0, 0, 2, 0],  
                [0, 0, 0, 3],  
                [0, 0, 0, 0]])
```

```
In [187]: np.diagflat([[1,2],[3,4]])
```

```
Out[187]: array([[1, 0, 0, 0],
                [0, 2, 0, 0],
                [0, 0, 3, 0],
                [0, 0, 0, 4]])
```

tri函数生成一个矩阵，在某对角线以下元素全为1，其余全为0，对角线由参数k控制

```
In [198]: np.tri(3,4, k=1, dtype=int)
```

```
Out[198]: array([[1, 1, 0, 0],
                [1, 1, 1, 0],
                [1, 1, 1, 1]])
```

tril函数输入一个矩阵，返回该矩阵的下三角矩阵，下三角的边界对角线由参数k控制

```
In [209]: print(arr)
print("-----")
print(np.tril(arr, k=0))
print("-----")
print(np.tril(arr, k=1))
```

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

```
-----
[[0 0 0]
 [3 4 0]
 [6 7 8]]
```

```
-----
[[0 1 0]
 [3 4 5]
 [6 7 8]]
```

triu函数与tril类似，返回的是矩阵的上三角矩阵

```
In [210]: np.triu(arr, k=1)
```

```
Out[210]: array([[0, 1, 2],  
                [0, 0, 5],  
                [0, 0, 0]])
```

vander函数输入一个一维数组，返回一个范德蒙矩阵

范德蒙行列式

$$\begin{vmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & & \vdots \\ x_1^{n-1} & x_2^{n-1} & \dots & x_n^{n-1} \end{vmatrix}$$

```
In [215]: np.vander([2,3,4,5])
```

```
Out[215]: array([[ 8,  4,  2,  1],  
                [27,  9,  3,  1],  
                [64, 16,  4,  1],  
                [125, 25,  5,  1]])
```

```
In [217]: np.vander([2,3,4,5],N=2)
```

```
Out[217]: array([[2, 1],  
                [3, 1],  
                [4, 1],  
                [5, 1]])
```

2.7 ndarray元素访问

2.7.1 一维数组

一维的ndarray数组可以使用python内置list操作方式

array[beg:end:step]

beg: 开始索引

end: 结束索引（不包含这个元素）

step: 间隔

需要注意的是：

beg可以为空，表示从索引0开始；
end也可以为空，表示达到索引结束（包含最后一个元素）；
step为空，表示间隔为1；
负值索引：倒数第一个元素的索引为-1，向前以此减1
负值step：从后往前获取元素

```
In [218]: x1 = np.arange(16)*4#list每个元素*4，list规模不变
```

```
In [224]: x1[0:10:3]
```

```
Out[224]: array([ 0, 12, 24, 36])
```

```
In [225]: x1[::-1]#倒置列表---pythonic
```

```
Out[225]: array([60, 56, 52, 48, 44, 40, 36, 32, 28, 24, 20, 16, 12, 8, 4, 0])
```

特别注意的是，ndarray中的切片返回的数组中的元素是原数组元素的索引，对返回数组元素进行修改会影响原数组的值

```
In [226]: x1[:-1]
```

```
Out[226]: array([ 0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56])
```

```
In [229]: y = x1[::-1]  
y
```

```
Out[229]: array([60, 56, 52, 48, 44, 40, 36, 32, 28, 24, 20, 16, 12, 8, 4, 0])
```

```
In [230]: y[0]=9999  
y
```

```
Out[230]: array([9999, 56, 52, 48, 44, 40, 36, 32, 28, 24, 20,  
                16, 12, 8, 4, 0])
```

In [232]:

```
x1
```

Out[232]: array([0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40,
 44, 48, 52, 56, 9999])

仔细观察上边y和x1操作变化！！

除了上述与list相似的访问元素的方式，ndarray还可以通过索引列表来访问，例如：

In [234]:

```
x1[[0,2,4,5]] ##指定获取索引为0,2,4,5的元素
```

Out[234]: array([0, 8, 16, 20])

2.7.2 多维数组

多维ndarray中，每一维都叫一个轴(axis)。

在ndarray中轴axis是非常重要的，有很多对于ndarray对象的运算都是基于axis进行，比如sum、mean等都会有一个axis参数（针对对这个轴axis进行某些运算操作）

对于多维数组，因为每一个轴都有一个索引，所以这些索引由逗号进行分割

In [240]:

```
x2 = np.arange(0,100,5).reshape(4,5)  
x2
```

Out[240]: array([[0, 5, 10, 15, 20],
 [25, 30, 35, 40, 45],
 [50, 55, 60, 65, 70],
 [75, 80, 85, 90, 95]])

In [241]:

```
x2[1,2] #第1行，第2列
```

Out[241]: 35

In [243]:

```
x2[1:4, 3] #第1行到第4行中所有第3列的元素，实际是第1、2、3行
```

Out[243]: array([40, 65, 90])

In [245]: `x2[:, 4]` *#所有行中的所有第4列的元素*

Out[245]: `array([20, 45, 70, 95])`

In [246]: `x2[0:3, :]` *#第0行到第三行中所有列的元素，实际是第0、1、2行*

Out[246]: `array([[0, 5, 10, 15, 20],
 [25, 30, 35, 40, 45],
 [50, 55, 60, 65, 70]])`

需要注意的是：

1> 当提供的索引比轴数少时，缺失的索引表示整个切片（只能缺失后边的轴）

In [247]: `x2[1:3]`
#确实第二个轴，就是只有x轴，没有y轴。

Out[247]: `array([[25, 30, 35, 40, 45],
 [50, 55, 60, 65, 70]])`

2> 当提供的索引为:时，也表示整个切片

In [250]: `x2[:, 0:4]` *#标识两个轴全部元素，列轴（y轴）范围是0-4*

Out[250]: `array([[0, 5, 10, 15],
 [25, 30, 35, 40],
 [50, 55, 60, 65],
 [75, 80, 85, 90]])`

3> 可以使用...代替几个连续的索引

In [253]: `x2[..., 0:4]`

Out[253]: `array([[0, 5, 10, 15],
 [25, 30, 35, 40],
 [50, 55, 60, 65],
 [75, 80, 85, 90]])`

一维和多维ndarray数组都可以支持迭代！只不过高维迭代需要使用flat做一个降维的操作！

```
In [255]: for itr in x2:  
          print(itr)
```

```
[ 0  5 10 15 20]  
[25 30 35 40 45]  
[50 55 60 65 70]  
[75 80 85 90 95]
```

```
In [256]: for itr in x2.flat: #将多维转化为一维（降维）  
          print(itr)
```

...

3. ndarray对象方法

上边有提到ndarray数组是同构的，那么下边我们来看看ndarray对象有哪些方法。

```
In [4]: print(dir(np.ndarray))
```

...

忽略类属性（__*__），剩下的就是ndarray的类方法。即就是：'all', 'any', 'argmax', 'argmin', 'argpartition', 'argsort', 'astype', 'base', 'byteswap', 'choose', 'clip', 'compress', 'conj', 'conjugate', 'copy', 'ctypes', 'cumprod', 'cumsum', 'data', 'diagonal', 'dot', 'dtype', 'dump', 'dumps', 'fill', 'flags', 'flat', 'flatten', 'getfield', 'imag', 'item', 'itemset', 'itemsizes', 'max', 'mean', 'min', 'nbytes', 'ndim', 'newbyteorder', 'nonzero', 'partition', 'prod', 'ptp', 'put', 'ravel', 'real', 'repeat', 'reshape', 'resize', 'round', 'searchsorted', 'setfield', 'setflags', 'shape', 'size', 'sort', 'squeeze', 'std', 'strides', 'sum', 'swapaxes', 'take', 'tobytes', 'tofile', 'tolist', 'tostring', 'trace', 'transpose', 'var', 'view'

3.1 数据基本操作与转换 Array conversion

常用方法如下：

ndarray.item(*args)	Copy an element of an array to a standard Python scalar and return it 复制数组中的一个元素，并返回
ndarray.tolist()	将数组转换成python内置list类型
ndarray.tostring([order])	创建ndarray数据类型为string
ndarray.tobytes([order])	创建ndarray数据类型是byte
ndarray.itemset(*args)	修改数组中某个元素的值
ndarray.copy([order])	复制数组 order: { 'C' , 'F' } 还有两个参数比较少见 同上 'C':行优先；'F':列优先
ndarray.fill(value)	Fill the array with a scalar value.使用值value填充数组

```
In [6]: cl = np.random.randint(12, size=(3,4))
cl
```

```
Out[6]: array([[ 7,  1,  0,  3],
               [ 6,  6,  5,  9],
               [ 1, 10, 10,  2]])
```

ndarray.item(*args) 复制数组中的一个元素，并返回。和直接取值是有区别的！！

```
In [11]: cl.item(8)
```

```
Out[11]: 1
```

```
In [12]: cl.item((2,3))
#根据元组（坐标）索引获取元素
```

```
Out[12]: 2
```

ndarray.tolist() 将数组转换成python内置list类型

```
In [13]: cl.tolist()
```

```
Out[13]: [[7, 1, 0, 3], [6, 6, 5, 9], [1, 10, 10, 2]]
```

ndarray.itemset(*args) 修改数组中某个元素的值

```
In [15]: cl.itemset(7, 9999) #修改第7个元素
cl
```

```
Out[15]: array([[ 7,  1,  0,  3],
               [ 6,  6,  5, 9999],
               [ 1, 10, 10,  2]])
```

```
In [16]: cl.itemset((1,1), 222) #修改索引为 ( 1 , 1 ) 对应的数据值
cl
```

```
Out[16]: array([[ 7,  1,  0,  3],
               [ 6, 222,  5, 9999],
               [ 1, 10, 10,  2]])
```

```
ndarray.copy([order])
```

```
In [19]: cl_t = cl.copy()
#这里注意一下：copy是深拷贝。
#什么是深拷贝、浅拷贝？
#简单讲就是：a,b是两个变量做复制操作，要是a与b共享相同物理空间，即就是同一个内存地址。叫做浅拷贝。
#反之，复制之后a,b是两个无关的变量，那么存储地址不相同，称之为深拷贝
#怎么验证a与b是否指向同一物理地址呢？
#修改a的值看b是否也变化了即可~
```

```
In [18]: cl_t
```

```
Out[18]: array([[ 7,  1,  0,  3],
               [ 6, 222,  5, 9999],
               [ 1, 10, 10,  2]])
```

其余参考这里 <https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.ndarray.html> (<https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.ndarray.html>)

3.2 矩阵数列形状变换 Shape manipulation

```
1> ndarray.reshape(shape, order='C')
#Returns an array containing the same data with a new shape. 数据不变，改变形状
2> ndarray.resize(new_shape, refcheck=True)
#Change shape and size of array in-place. 修改数组的形状（需要保持元素个数前后相同）
3> ndarray.transpose(*axes)
#Returns a view of the array with axes transposed. 返回数组针对某一轴进行转置的视图
4> ndarray.swapaxes(axis1, axis2) 返回数组axis1轴与axis2轴互换的视图
5> ndarray.flatten([order]) 返回将原数组压缩成一维数组的拷贝（全新的数组）
6> ndarray.ravel([order]) 返回将原数组压缩成一维数组的视图
7> ndarray.squeeze([axis]) 返回将原数组中的shape中axis==1的轴移除之后的视图
```

```
In [37]: sm = np.arange(0,12)
```

```
In [38]: sm
```

```
Out[38]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

1> ndarray.reshape(shape[,order]) 数据不变，改变形状

```
In [32]: x_sm = sm.reshape((3,4)) #在保证数据域不变情况下，直接修改原来数组结构
sm.reshape((3,4)) #变形
print(sm) #这里就可以看出来reshape直接修改原来数组结构
x_sm[0,0] = 888 #修改值
x_sm
```

```
[[888  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

```
Out[32]: array([[888,  1,  2,  3],
               [ 4,  5,  6,  7],
               [ 8,  9, 10, 11]])
```

2> ndarray.resize(new_shape, refcheck=True) 原地修改数组的形状（需要保持元素个数前后相同）

```
In [42]: sm.resize((4,3)) #resize没有返回值，会直接修改数组的shape，如下所示
sm
```

```
Out[42]: array([[ 0,  1,  2],
               [ 3,  4,  5],
               [ 6,  7,  8],
               [ 9, 10, 11]])
```

3> ndarray.transpose(*axes) 返回数组针对某一轴进行转置的视图

```
In [43]: sm.transpose()
```

```
Out[43]: array([[ 0,  3,  6,  9],  
               [ 1,  4,  7, 10],  
               [ 2,  5,  8, 11]])
```

```
In [64]: sm.resize(1,2,6)  
sm
```

```
Out[64]: array([[[ 0,  1,  2,  3,  4,  5],  
                [ 6,  7,  8,  9, 10, 11]])]
```

```
In [101]: ar = np.arange(12)  
ss = ar.reshape((4,3))
```

4> ndarray.swapaxes(axis1, axis2) 返回数组axis1轴与axis2轴互换的视图

```
In [93]: #这个点亲看后边3.2关于axis专门做的解释说明!  
sm.swapaxes(1,2)
```

```
Out[93]: array([[[ 0,  6],  
                [ 1,  7],  
                [ 2,  8],  
                [ 3,  9],  
                [ 4, 10],  
                [ 5, 11]])]
```

5> ndarray.flatten([order]) 返回将原数组压缩成一维数组的拷贝（返回全新的数组）---降维


```
In [110]: print(ss)
y=ss.flatten()
y
```

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

```
Out[110]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
In [104]: ss #flatten不会修改原来的数组
```

```
Out[104]: array([[ 0,  1,  2],
 [ 3,  4,  5],
 [ 6,  7,  8],
 [ 9, 10, 11]])
```

6> ndarray.ravel([order]) 返回将原数组压缩成一维数组的视图---返回原数组

```
In [107]: ss.ravel()
```

```
Out[107]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

7> ndarray.squeeze([axis]) 返回将原数组中的shape中axis==1的轴移除之后的视图

```
In [118]: #这稍复杂，请看官网。
ss.squeeze().shape #没有1
```

```
Out[118]: (4, 3)
```

3.3 计算

这里我们先来理解一下ndarray总的“axis”
请看举例：

```
In [145]: np.random.seed(123)
x = np.random.randint(0,5,[3,2,2])
print(x)
x.max(axis=0)
```

```
[[[2 4]
  [2 1]]
```

```
[[3 2]
 [3 1]]
```

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

```
Out[145]: array([[3, 4],
                [3, 1]])
```

分析：

首先数列模型是`shape(3,2,2)`，由输出看出这个“3”就是x的三个部分，分别有换行区分明显区分出来。而axis取值0对应的就是3，那么`max(axis=0)`代表在这三个部分求取最大值。按照位置——分式比较就是`max(2, 3, 1)=3`,`max(4,2,0)=4`,`max(2,3,1)=3`,`max(1,1,1)=1`进而得到如下结果

采用集合思想，`shape(3,2,2)`相当于是一个多维空间，axis=0意味着三个面的重叠；axis=1是3个面每个面中的列组成的集合；axis=2是3个面每个面行组成的集合

那么请看详细数据说明：

```
In [142]: x.max(axis=1)
```

```
Out[142]: array([[2, 4],
                [3, 2],
                [1, 1]])
```

```
In [143]: x.max(axis=2)
```

```
Out[143]: array([[4, 2],
                [3, 3],
                [1, 1]])
```

所以明白了axis的原理，那么计算就很简单了，只简单需要看一下方法就可以。

ndarray.max([axis, out, keepdims]) 返回根据指定的axis计算最大值
ndarray.argmax([axis, out]) 返回根据指定axis计算最大值的索引
ndarray.min([axis, out, keepdims]) 返回根据指定的axis计算最小值
ndarray.argmin([axis, out]) 返回根据指定axis计算最小值的索引
ndarray.ptp([axis, out]) 返回根据指定axis计算最大值与最小值的差
ndarray.clip([min, max, out]) 返回数组元素限制在[min, max]之间的新数组（小于min的转为min，大于max的转为max）
ndarray.round([decimals, out]) 返回指定精度的数组（四舍五入）
ndarray.trace([offset, axis1, axis2, dtype, out]) 返回数组的迹（对角线元素的和）
ndarray.sum([axis, dtype, out, keepdims]) 根据指定axis计算数组的和，默认求所有元素的和
ndarray.cumsum([axis, dtype, out]) 根据指定axis计算数组的累积和
ndarray.mean([axis, dtype, out, keepdims]) 根据指定axis计算数组的平均值
ndarray.var([axis, dtype, out, ddof, keepdims]) 根据指定的axis计算数组的方差
ndarray.std([axis, dtype, out, ddof, keepdims]) 根据指定axis计算数组的标准差
ndarray.prod([axis, dtype, out, keepdims]) 根据指定axis计算数组的积
ndarray.cumprod([axis, dtype, out]) 根据指定axis计算数据的累积积
ndarray.all([axis, dtype, out]) 根据指定axis判断所有元素是否全部为真
ndarray.any([axis, out, keepdims]) 根据指定axis判断是否有元素为真

3.4 选择元素以及操作

ndarray.take(indices[, axis, out, mode]) 从原数组中根据指定的索引获取对应元素，并构成一个新的数组返回
ndarray.put(indices, values[, mode]) 将数组中indices指定的位置设置为values中对应的元素值
ndarray.repeat(repeats[, axis]) 根据指定的axis重复数组中的元素
ndarray.sort([axis, kind, order]) 原地对数组元素进行排序
ndarray.argsort([axis, kind, order]) 返回对数组进行升序排序之后的索引
ndarray.partition(kth[, axis, kind, order]) 将数组重新排列，所有小于kth的值在kth的左侧，所有大于或等于kth的值在kth的右侧
ndarray.argpartition(kth[, axis, kind, order]) 对数组执行partition之后的元素索引
ndarray.searchsorted(v[, side, sorter]) 若将v插入到当前有序的数组中，返回插入的位置索引
ndarray.nonzero() 返回数组中非零元素的索引
ndarray.diagonal([offset, axis1, axis2]) 返回指定的对角线

```
In [150]: aa = np.arange(0,100,5).reshape(4,5)
          aa.flatten()
```

```
Out[150]: array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
                85, 90, 95])
```

```
In [151]: aa.take([0,2,3,4])
```

```
Out[151]: array([ 0, 10, 15, 20])
```

```
In [153]: aa.take([[2,3],[4,7]])
```

```
Out[153]: array([[10, 15],  
                [20, 35]])
```

```
In [164]: aa.put([0,-1],[3,252]) #设置索引为0的值为3，最后一位为252
```

```
In [165]: aa
```

```
Out[165]: array([[ 3,  5, 10, 15, 20],  
                [25, 30, 35, 40, 45],  
                [50, 55, 60, 65, 70],  
                [75, 80, 85, 90, 252]])
```

```
In [170]: bb = aa.flatten()  
bb[10]=9999  
bb.sort()  
bb
```

```
Out[170]: array([ 3,  5, 10, 15, 20, 25, 30, 35, 40, 45, 55,  
                60, 65, 70, 75, 80, 85, 90, 252, 9999])
```

```
In [171]: bb.argsort() #获取排序后的元素索引
```

```
Out[171]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,  
                17, 18, 19], dtype=int64)
```

```
In [172]: bb[bb.argsort()] #获取上述索引对应的值
```

```
Out[172]: array([ 3,  5, 10, 15, 20, 25, 30, 35, 40, 45, 55,  
                60, 65, 70, 75, 80, 85, 90, 252, 9999])
```

```
In [179]: bb[2]=8888  
print(bb[10])  
bb.partition(10)#小于70在左侧, 大于在右侧  
bb
```

70

```
Out[179]: array([ 55,  3,  5, 25, 30, 35, 45, 40, 65, 70, 75,  
                80, 90, 85, 252, 8888, 8888, 8888, 8888, 9999])
```

其余的可以自己尝试~

Copying@xnchall