

数据分析之Numpy

Numpy 是Python 的一种开源的数值计算拓展，这种工具可用来存储和处理大型矩阵。

Numpy的一个重要特性是数组计算。

```
import numpy  
import numpy as np  
from numpy import *      import warnings  
                        warnings.filterwarnings('ignore')
```

1. Numpy的数组对象及其索引

数组上的数学操作

列表方法——每个字符+1

```
In [2]: a=[1, 2, 3, 4]  
        a+1
```



```
In [3]: [x+1 for x in a]
```



```
Out[3]: [2, 3, 4, 5]
```



```
In [5]: b=[2, 3, 4, 5]  
        a+b
```



```
Out[5]: [1, 2, 3, 4, 2, 3, 4, 5]
```

```
In [6]: [x+y for (x,y) in zip(a,b)]
```



```
Out[6]: [3, 5, 7, 9] ↴
```

使用Numpy, 更高效

```
In [7]: a=np.array([1, 2, 3, 4])  
a
```

```
Out[7]: array([1, 2, 3, 4])
```

```
In [8]: a+1
```

```
Out[8]: array([2, 3, 4, 5])
```

```
In [9]: a*2b
```

```
Out[9]: array([2, 4, 6, 8])
```

```
In [10]: b=np.array([2, 3, 4, 5])  
a+b
```

```
Out[10]: array([3, 5, 7, 9])
```

产生数组

①从列表产生数组

```
In [13]: 1 l=[0, 1, 2, 3]  
          2 a=np.array(l)  
          3 a
```

```
Out[13]: array([0, 1, 2, 3])
```

②从列表传入

```
In [14]: a=np.array([1, 2, 3, 4])  
a
```

```
Out[14]: array([1, 2, 3, 4])
```

③生成全0数组

```
In [15]: np.zeros(5)
```

```
Out[15]: array([ 0.,  0.,  0.,  0.,  0.])
```

④生成全1的数组

```
In [17]: np.ones(5, dtype='int')
```

```
Out[17]: array([1, 1, 1, 1, 1])
```

```
In [18]: np.ones(5, dtype='bool')
```

```
Out[18]: array([ True,  True,  True,  True,  True], dtype=bool)
```

⑤使用 fill 方法将数组设为指定值

```
In [19]: a=np.array([1, 2, 3, 4])  
a
```

```
Out[19]: array([1, 2, 3, 4])
```

```
In [21]: a.fill(5)  
a
```

```
Out[21]: array([5, 5, 5, 5])
```

与列表不同，数组中要求所有元素的dtype是一样的
可以按照已有类型进行转换类型

```
In [22]: a.fill(2.5)
a
```

```
Out[22]: array([2, 2, 2, 2])
```

```
In [23]: a=a.astype('float')
a.fill(2.5)
a
```

```
Out[23]: array([ 2.5,  2.5,  2.5,  2.5])
```

使用特定的方法生成特殊的数组

⑥生成整数序列

```
In [24]: a=np.arange(1,10)
a
```

```
Out[24]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [25]: a=np.arange(1,10,2)
a
```

```
Out[25]: array([1, 3, 5, 7, 9])
```

⑦生成等差数列（左闭右闭）

```
In [27]: a=np.linspace(1,10,21)
a
```

```
Out[27]: array([ 1. , 1.45, 1.9 , 2.35, 2.8 , 3.25, 3.7 , 4.15,
 4.6 , 5.05, 5.5 , 5.95, 6.4 , 6.85, 7.3 , 7.75,
 8.2 , 8.65, 9.1 , 9.55, 10. ])
```

⑧生成随机数

```
nd1 = np.random.randint(1)#0
nd2 = np.random.randint(1,5)#随机生成一个元素，值>=low ;<high
nd3 = np.random.randint(1,5,size=3)#随机生成一个三个元素的一维数组
nd4 = np.random.randint(1,5,size=(2,3))#随机生成一个二维数组，二行三列
nd5 = np.random.randint(1,5,size=(2,3,2))
生成的nd5 = array([[[3, 2],
[4, 1],
[3, 1]],
[[4, 3],
[2, 3],
[2, 2]]])
```

```
In [28]: np.random.rand(10)
Out[28]: array([ 0.59924725,  0.59663379,  0.52722938,  0.81755596,  0.73019937,
   0.44955513,  0.78959751,  0.8632088 ,  0.18679637,  0.80241495])
```

```
In [29]: np.random.randn(10)
Out[29]: array([-0.82864934, -0.09415961, -0.56594416, -0.10259398, -1.0670236 ,
   0.91548794, -1.03763122, -0.33534751, -1.43346526, -0.02358178])
```

```
In [31]: np.random.randint(1, 10, 10)
Out[31]: array([8, 5, 7, 2, 8, 6, 1, 7, 4, 1])
```

数组属性

①查看类型

```
In [32]: a
Out[32]: array([ 1. ,  1.45,  1.9 ,  2.35,  2.8 ,  3.25,  3.7 ,  4.15,
   4.6 ,  5.05,  5.5 ,  5.95,  6.4 ,  6.85,  7.3 ,  7.75,
   8.2 ,  8.65,  9.1 ,  9.55, 10. ])
In [33]: type(a)
Out[33]: numpy.ndarray
```

②查看数组中的数据类型

```
In [34]: a.dtype
Out[34]: dtype('float64')
```

③查看形状，会返回一个元祖，每个元素代表这一维的元素数目

```
In [35]: a.shape
Out[35]: (21, )
```

④查看数组里面元素的数目

```
In [36]: a.size
```

```
Out[36]: 21
```

⑤查看数组的维度

```
In [37]: a.ndim
```

```
Out[37]: 1
```

索引与切片

①索引第一个元素

```
In [38]: a=np.array([0, 1, 2, 3])  
a[0]
```

```
Out[38]: 0
```

②修改第一个元素的值

```
In [39]: a[0]=10  
a
```

```
Out[39]: array([10, 1, 2, 3])
```

③切片，支持负索引

```
In [40]: a=np.array([11, 12, 13, 14, 15])  
a[1:3]
```

```
Out[40]: array([12, 13])
```

```
In [41]: a[1:-2]
```

```
Out[41]: array([12, 13])
```

```
In [42]: a[-4:3]
```

```
Out[42]: array([12, 13])
```

④省略参数

```
In [43]: a[-2:]
```

```
Out[43]: array([14, 15])
```

```
In [44]: a[::-2]
```

```
Out[44]: array([11, 13, 15])
```

实例——假设我们记录一部电影的累积票房

```
In [45]: ob=np.array([21000, 21800, 22240, 23450, 25000])  
ob
```

```
Out[45]: array([21000, 21800, 22240, 23450, 25000])
```

后一天减去前一天的票房

```
In [46]: ob2=ob[1:]-ob[:-1]  
ob2
```

```
Out[46]: array([-800, -440, 1210, 1550])
```

多维数组及其属性

①Array可以用来生成多维数组

```
In [48]: a=np.array([[0, 1, 2, 3], [10, 11, 12, 13]])  
a
```

```
Out[48]: array([[ 0,  1,  2,  3],  
                  [10, 11, 12, 13]])
```

②查看形状

```
In [49]: a.shape
```

```
Out[49]: (2, 4)
```

③查看总的元素个数

```
In [50]: a.size
```

```
Out[50]: 8
```

查看维度

```
In [51]: a.ndim
```

```
Out[51]: 2
```

多维数组索引

1是行索引， 3是列索引：

```
In [52]: a
```

```
Out[52]: array([[ 0,  1,  2,  3],
                 [10, 11, 12, 13]])
```

```
In [53]: a[1, 3] I
```

```
Out[53]: 13
```

通过索引给它赋值，修改变量：

```
In [54]: a[1, 3]=-1
a
```

```
Out[54]: array([[ 0,  1,  2,  3],
                 [10, 11, 12, -1]])
```

利用单索引来索引一整行内容：

```
In [55]: a[1]
```

```
Out[55]: array([10, 11, 12, -1])
```

逗号前面为行索引，逗号后面为列索引，通过索引返回对应的内容：

```
In [56]: a[:, 1]
```

```
Out[56]: array([ 1, 11])
```

多维数组切片

```
In [57]: a=np.array([[0, 1, 2, 3, 4, 5], [10, 11, 12, 13, 14, 15], [20, 21, 22, 23, 24, 25], [30, 31, 32, 33, 34, 35], [40, 41, 42, 43, 44, 45], [50, 51, 52, 53, 54, 55]])
a
```

```
Out[57]: array([[ 0,  1,  2,  3,  4,  5],
                 [10, 11, 12, 13, 14, 15],
                 [20, 21, 22, 23, 24, 25],
                 [30, 31, 32, 33, 34, 35],
                 [40, 41, 42, 43, 44, 45],
                 [50, 51, 52, 53, 54, 55]])
```

得到第一行的第4和第5两个元素：

```
In [58]: a[0, 3:5]
```

```
Out[58]: array([3, 4])
```

得到最后两行的最后两列：

```
In [59]: a[4:, 4:]
```

```
Out[59]: array([[44, 45],  
                 [54, 55]])
```

得到第三列：

```
In [60]: a[:, 2]
```

```
Out[60]: array([ 2, 12, 22, 32, 42, 52])
```

△每一维都支持切片的规则，包括负索引、省略：

[lower:upper:step]

如，取出3、5行的奇数列：

```
In [61]: a[2::2, ::2]
```

```
Out[61]: array([[20, 22, 24],  
                  [40, 42, 44]])
```

切片是引用

切片在内存中使用的是引用机制

```
In [87]: a=np.array([0, 1, 2, 3, 4])
b=a[2:4]
print(b)
```

```
[2 3]
```

```
In [88]: b[0]=10
a
```

```
Out[88]: array([ 0,  1, 10,  3,  4])
```

改变b会改变a的值，但该现象在列表中不存在：

```
In [89]: a=[1, 2, 3, 4, 5]
b=a[2:4]
b[0]=10
print(a)
```

```
[1, 2, 3, 4, 5]
```

解决这种限制，可以使用[copy\(\)](#)方法，这个复制会申请新的内存：

```
In [90]: a=np.array([0, 1, 2, 3, 4])
b=a[2:4].copy()
b[0]=10
a
```

```
Out[90]: array([0, 1, 2, 3, 4])
```

花式索引

切片只能支持连续或者等间隔的切片操作，要实现任意位置的操作，需要使用花式索引 fancy slicing

一维花式索引

用arange函数来产生等差数组：

```
In [91]: a=np.arange(0, 100, 10)
a
```

```
Out[91]: array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

花式索引需要指定索引位置：

```
In [92]: index=[1, 2, -3]
y=a[index]
print(y)
```



```
[10 20 70]
```

使用布尔数组来花式索引：

```
In [93]: mask=np.array([0, 0, 0, 0, 1, 0, 0, 1, 0], dtype=bool)
mask
```

```
Out[93]: array([False, True, True, False, False, True, False, True, False], dtype=bool)
```

Mask必须是布尔数组，长度必须和数组长度一致

```
In [94]: a[mask]
```

```
Out[94]: array([10, 20, 50, 80])
```

二维花式索引

需要给定行和列的值：

```
In [95]: a=np.array([[0, 1, 2, 3, 4, 5], [10, 11, 12, 13, 14, 15], [20, 21, 22, 23, 24, 25], [30, 31, 32, 33, 34, 35], [40, 41, 42, 43, 44, 45], [50, 51, 52, 53, 54, 55]])
a
```



```
Out[95]: array([[ 0,  1,  2,  3,  4,  5],
 [10, 11, 12, 13, 14, 15],
 [20, 21, 22, 23, 24, 25],
 [30, 31, 32, 33, 34, 35],
 [40, 41, 42, 43, 44, 45],
 [50, 51, 52, 53, 54, 55]])
```

返回一条次对角线上的5个值：

```
In [96]: a[(0, 1, 2, 3, 4), (1, 2, 3, 4, 5)]
```



```
Out[96]: array([ 1, 12, 23, 34, 45])
```

返回最后三行的第1, 3, 5列：

```
In [97]: a[3:, [0, 2, 4]]
```

```
Out[97]: array([[30, 32, 34],  
                 [40, 42, 44],  
                 [50, 52, 54]])
```

可以使用mask进行索引：

```
In [98]: mask=np.array([1, 0, 1, 0, 0, 1], dtype=bool)  
a[mask, 2]
```

```
Out[98]: array([ 2, 22, 52])
```

△与切片不同，花式索引返回的是原对象的一个复制而不是引用。

“不完全”索引

只给定索引的时候，返回整行：

```
In [148]: y=a[:3]  
y
```

```
Out[148]: array([[ 0,  1,  2,  3,  4,  5],  
                  [10, 11, 12, 13, 14, 15],  
                  [20, 21, 22, 23, 24, 25]])
```

使用花式索引取出第2, 3, 5行：

```
In [149]: con=np.array([0, 1, 1, 0, 1, 0], dtype=bool)  
a[con]
```

```
Out[149]: array([[10, 11, 12, 13, 14, 15],  
                  [20, 21, 22, 23, 24, 25],  
                  [40, 41, 42, 43, 44, 45]])
```

where 语句

where(array)

where 函数会返回所有非零元素的索引

一维数组

判断数组中的元素是否大于10:

```
In [150]: a=np.array([0, 12, 5, 20])
```

```
In [151]: a>10
```

```
Out[151]: array([False, True, False, True], dtype=bool)
```

数组中所有大于10的元素的索引位置:

```
In [152]: np.where(a>10)
```

```
Out[152]: (array([1, 3], dtype=int64), )
```

△where的返回值是一个元祖，返回的是索引位置

可以用数组操作:

```
In [153]: a[a>10]
```

```
Out[153]: array([12, 20])
```

```
In [154]: a[np.where(a>10)]
```

```
Out[154]: array([12, 20])
```

2. 数组类型

基本类型	可用的Numpy类型	备注
布尔型	bool	占1个字节
整型	int8, int16, int32, int64, int128, int	int 跟C语言中的 long 一样大
无符号整型	uint8, uint16, uint32, uint64, uint128, uint	uint 跟C语言中的 unsigned long 一样大
浮点数	float16, float32, float64, float, longfloat	默认为双精度 float64 , longfloat 精度大小与系统有关
复数	complex64, complex128, complex, longcomplex	默认为 complex128 , 即实部虚部都为双精度
字符串	string, unicode	可以使用 dtype=S4 表示一个4字节字符串的数组
对象	object	数组中可以使用任意值
时间	datetime64, timedelta64	

类型转换

```
In [155]: a=np.array([1.5,-3], dtype=float)
a
```

Out[155]: array([1.5, -3.])

asarray函数：

```
In [157]: a=np.array([1,2,3])
np.asarray(a, dtype=float)
```

Out[157]: array([1., 2., 3.])

astype函数，返回一个新数组：

```
In [161]: a=np.array([1, 2, 3])
b=a.astype(float)
```

```
In [163]: b[0]=0.5
b
```

```
Out[163]: array([ 0.5,  2. ,  3. ])
```

```
In [164]: a
```

```
Out[164]: array([1, 2, 3])
```

3. 数组操作

以豆瓣10部高分电影为例

```
In [63]: ##电影名称
mv_name=['肖申克的救赎','控方证人','美丽人生','阿甘正传','霸王别姬','泰坦尼克号','辛德勒的名单','这个杀手不太冷','',
          'In [64]: ##评分人数
mv_num=np.array([692795, 42995, 327855, 580897, 478523, 157074, 306904, 662552, 284652, 159302])
```

```
In [65]: ##评分
mv_score=np.array([9.6, 9.5, 9.5, 9.4, 9.4, 9.4, 9.4, 9.3, 9.3, 9.3])
```

```
In [66]: ##电影时长(分钟)
mv_length=np.array([142, 116, 116, 142, 171, 194, 195, 133, 109, 92])
```

数组排序

①sort函数

```
In [165]: np.sort(mv_num)
```

```
Out[165]: array([ 42995, 157074, 159302, 284652, 306904, 327855, 478523, 580897,
                  662552, 692795])
```

```
In [166]: mv_num
```

```
Out[166]: array([692795, 42995, 327855, 580897, 478523, 157074, 306904, 662552,
                  284652, 159302])
```

②argsort函数，返回从小到大的排列在数组中的索引位置：

```
In [167]: order=np.argsort(mv_num)
order
```

```
Out[167]: array([1, 5, 9, 8, 6, 2, 4, 3, 7, 0], dtype=int64)
```

```
In [167]: order=np.argsort(mv_num)
order
```

```
Out[167]: array([1, 5, 9, 8, 6, 2, 4, 3, 7, 0], dtype=int64)
```

```
In [169]: mv_name[order[-1]]
```

```
Out[169]: '肖申克的救赎'
```

③求和

```
In [170]: np.sum(mv_num)
```

```
Out[170]: 3693549
```

```
In [171]: mv_num.sum()
```

```
Out[171]: 3693549
```

④最大值

```
In [172]: np.max(mv_length)
```

```
Out[172]: 195
```

```
In [173]: mv_length.max()
```

```
Out[173]: 195
```

⑤最小值

```
In [174]: np.min(mv_score)
```

```
Out[174]: 9.3000000000000007
```

```
In [175]: mv_score.min()
```

```
Out[175]: 9.3000000000000007
```

⑥均值

```
In [176]: np.mean(mv_length)
```

```
Out[176]: 141.0
```

```
In [177]: mv_length.mean()
```

```
Out[177]: 141.0
```

⑦标准差

```
In [178]: np.std(mv_length)
```

```
Out[178]: 33.713498780162226
```

```
In [179]: mv_length.std()
```

```
Out[179]: 33.713498780162226
```

⑧相关系数矩阵

```
In [180]: np.cov(mv_score, mv_length)
```

```
Out[180]: array([[ 9.88888889e-03,  4.55555556e-01],
                  [ 4.55555556e-01,  1.26288889e+03]])
```

多维数组操作

数组形状

```
In [182]: a=np.arange(6)  
a
```

```
Out[182]: array([0, 1, 2, 3, 4, 5])
```

```
In [183]: a.shape=2,3  
a
```

```
Out[183]: array([[0, 1, 2],  
                  [3, 4, 5]])
```

```
In [184]: a.shape
```

```
Out[184]: (2, 3)
```

与之对应的方法是reshape，但不会修改原来数组的值，而是返回一个新的数组：

```
In [185]: a=np.arange(6)  
a
```

```
Out[185]: array([0, 1, 2, 3, 4, 5])
```

```
In [186]: a.reshape(2,3)
```

```
Out[186]: array([[0, 1, 2],  
                  [3, 4, 5]])
```

```
In [187]: a
```

```
Out[187]: array([0, 1, 2, 3, 4, 5])
```

转置

```
In [188]: a=a.reshape(2, 3)  
a
```

```
Out[188]: array([[0, 1, 2],  
                  [3, 4, 5]])
```

```
In [189]: a.T
```

```
Out[189]: array([[0, 3],  
                  [1, 4],  
                  [2, 5]])
```

```
In [190]: a.transpose()
```

```
Out[190]: array([[0, 3],  
                  [1, 4],  
                  [2, 5]])
```

```
In [191]: a
```

```
Out[191]: array([[0, 1, 2],  
                  [3, 4, 5]])
```

数组连接

concatenate(a0,a1,.....,aN),axis=0)

△这些数组要用()括起来到一个元组中去。

除了给定的轴外，这些数组其他轴的长度必须是一样的。

```
In [193]: x=np.array([[0, 1, 2], [10, 11, 12]])  
y=np.array([[50, 51, 52], [60, 61, 62]])  
print(x.shape)  
print(y.shape)
```

```
(2, 3)  
(2, 3)
```

默认沿着第一维进行连接：

```
In [195]: z=np.concatenate((x, y))  
z
```

```
Out[195]: array([[ 0,  1,  2],  
                  [10, 11, 12],  
                  [50, 51, 52],  
                  [60, 61, 62]])
```

想改变形状，可以对轴进行修改，axis =1

```
In [196]: z=np.concatenate((x, y), axis=1)  
z
```

```
Out[196]: array([[ 0,  1,  2, 50, 51, 52],  
                  [10, 11, 12, 60, 61, 62]])
```

这里x和y的形状是一样的，还可以将它们连接成三维的数组，但是concatenate不能提供这样的功能，但可以用下面的方法：

```
In [197]: z=np.array((x, y))  
z
```

```
Out[197]: array([[[ 0,  1,  2],  
                   [10, 11, 12]],  
  
                   [[50, 51, 52],  
                   [60, 61, 62]]])
```

事实上，Numpy提供了分别对应这三种情况的函数：

vstack-----纵向堆叠

```
In [199]: np.vstack((x, y))
```

```
Out[199]: array([[ 0,  1,  2],
                  [10, 11, 12],
                  [50, 51, 52],
                  [60, 61, 62]])
```

hstack-----横向堆叠

```
In [200]: np.hstack((x, y))
```

```
Out[200]: array([[ 0,  1,  2, 50, 51, 52],
                  [10, 11, 12, 60, 61, 62]])
```

dstack-----维度上

```
In [201]: np.dstack((x, y))
```

```
Out[201]: array([[[ 0, 50],
                  [ 1, 51],
                  [ 2, 52]],
                  [[10, 60],
                  [11, 61],
                  [12, 62]]])
```

numpy内置函数

求绝对值：

```
In [202]: a=np.array([-1, 2, 3, -2])
```

```
In [203]: np.abs(a)
```

```
Out[203]: array([1, 2, 3, 2])
```

求指数

```
In [204]: np.exp(a)
```

```
Out[204]: array([ 0.36787944, 7.3890561 , 20.08553692, 0.13533528])
```

求中值

```
In [205]: np.median(a)
```

```
Out[205]: 0.5
```

求累计值

```
In [206]: np.cumsum(a)
```

```
Out[206]: array([-1, 1, 4, 2], dtype=int32)
```