前两篇讲了 Python 的基础，今天开始进入 Python 数据分析工具的教程。

Python 数据分析绝对绕不过的四个包是 **numpy、scipy、pandas 还有 matplotlib**。

numPy 是 Python 数值计算最重要的基础包，大多数提供科学计算的包都是用 numPy 的数组作为构建基础。专门用来处理矩阵，它的运算效率比列表更高效。

scipy 是基于 numpy 的科学计算包，包括统计、线性代数等工具。

pandas 是基于 numpy 的数据分析工具，能够快速的处理结构化数据的大量数据结构和函数。

matplotlib 是最流行的用于绘制数据图表的 Python 库。

本文先分享 NumPy 包。

NumPy 的 ndarray：多维数组对象

numpy 的数据结构是 n 维的数组对象，叫做 ndarray。可以用这种数组对整块数据执行一些数学运算，其语法跟标量元素之间的运算一样。

创建并操作多维数组：

```
In [20]: import numpy as np

In [21]: data = np.array([[0.9526, -0.246, 00.08856],[0.5639, 0.2397, 0.9104]])

In [22]: data
Out[22]: array([[ 0.9526 , -0.246  ,  0.08856],
                [ 0.5639 ,  0.2397 ,  0.9104 ]])

In [23]: data * 10
Out[23]: array([[ 9.526 , -2.46  ,  0.8856],
                [ 5.639 ,  2.397 ,  9.104 ]])

In [24]: data + data
Out[24]: array([[ 1.9052 , -0.492  ,  0.17712],
                [ 1.1278 ,  0.4794 ,  1.8208 ]])
```

ndarray 对象中所有元素必须是相同类型的，每个数组都有一个 shape 和 dtype。

- shape ：表示各维度大小的元组
- dtype ：说明数组数据类型的对象

```
In [25]: data.shape
Out[25]: (2, 3)

In [26]: data.dtype
Out[26]: dtype('float64')
```

创建 ndarray：一种多维数组对象

创建数组最简单的办法就是使用 array 函数，它接受一切序列型对象（包括其它数组），然后产生一个新的 NumPy 数组（含有原来的数据）。

np.array 会尝试为新建的这个数组推断出一个较为合适的数据类型，这个数据类型保存在一个特殊的 dtype 对象中。

zeros 和 ones 也分别可以创建指定大小的全 0 或全 1 数组，empty 可以创建一个没有任何具体值的数组（它返回的都是一些未初始化的垃圾值）：

```
In [43]: np.zeros(10)
Out[43]: array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])

In [44]: np.zeros((3,6))
Out[44]: array([[ 0.,  0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.,  0.]])

In [45]: np.empty((2,3,2))
Out[45]: array([[[ -1.49166815e-154,  -1.49166815e-154],
                 [  2.96439388e-323,   0.00000000e+000],
                 [  2.12199579e-314,   1.58817677e-052]],

                [[  5.89713881e-091,   4.50235755e+174],
                 [  1.26037658e-076,   8.00874983e+165],
                 [  3.99910963e+252,   8.34404841e-309]]])
```

arange 是 Python 内置函数 range 的数组版，np.arange 返回间隔均匀的一些值。

ndarray 的数据类型

```
In [54]: arr1 = np.array([1, 2, 3], dtype=np.float64)

In [55]: arr1.dtype
Out[55]: dtype('float64')
```

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

需要知道你所处理的数据的大致类型是浮点数、复数、整数、布尔值、字符串，还是普通的 python 对象。当需要控制数据在内存和磁盘中的存储方式时，就得了解如何控制存储类型。

可通过 ndarray 的 astype 方法显示地转换其 dtype：

```
In [56]: arr = np.array([1,2,3])

In [57]: arr.dtype
Out[57]: dtype('int64')

In [58]: float_arr = arr.astype(np.float64)

In [59]: float_arr.dtype
Out[59]: dtype('float64')
```

若将浮点数转换成整数，则小数部分将会被截断。

若某字符串数组表示的全是数字，可用 astype 将其转换为数值形式：

```
In [60]: numeric_strings = np.array(['1.25', '-9.6', '42'], dtype=np.string_)

In [61]: numeric_strings.astype(float)
Out[61]: array([  1.25,  -9.6 ,  42.  ])
```

这里没写 np.float64 只写了 float，但是 NumPy 会将 Python 类型映射到等价的 dtype 上。

数组的 dtype 的另一个用法：

```
In [62]: int_array = np.arange(10)

In [63]: int_array
Out[63]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [64]: calibers = np.array([.22, .270, .357, .380, .44, .50], dtype=np.float64)

In [65]: int_array.astype(calibers.dtype)
Out[65]: array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
```

int_array 变成了和 calibers 一样的浮点型数组

用简洁类型的代码表示 dtype：

```
In [69]: empty_uint32 = np.empty(8, dtype='u4')

In [70]: empty_uint32
Out[70]: array([         0, 1075314688,          0, 1075707904,          0,
                1075838976, 2576980378, 1069128089], dtype=uint32)
```

u4(unit32)：无符号的 32 位（4 个字节）整型。

调用 astype 无论如何都会创建出一个新的数组（原始数据的一份拷贝）。

浮点数只能表示近似的分数值，在复杂计算中可能会积累一些浮点错误，因此比较操作只在一定小数位以内有效。

数组和标量之间的运算

数组：可对数据执行批量运算（不用编写循环即可）。这通常叫做矢量化(vectorization)。

- 大小相等的数组之间，它们之间任何的算术运算都会应用到元素级（每个元素都做这个运算了），数组与标量的算术运算也是。
- 不同大小的数组之间的运算叫做**广播(broadcasting)**。

基本的索引和切片

```
In [71]: arr = np.arange(10)

In [72]: arr
Out[72]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [73]: arr[5]
Out[73]: 5

In [74]: arr[5:8]
Out[74]: array([5, 6, 7])

In [75]: arr[5:8]=12

In [76]: arr
Out[76]: array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
```

数据不会被复制，任何修改都直接改了原数组。

如果仅是要一份副本，则用 .copy()。

```
In [77]: arr[5:8].copy()
Out[77]: array([12, 12, 12])
```

对二维数组单个元素的索引：

```
In [79]: arr2d = np.array([[1,2,3],[4,5,6],[7,8,9]])

In [80]: arr2d[0][2]
Out[80]: 3

In [81]: arr2d[0,2]
Out[81]: 3
```

这两种方式等价。

若 arr2d[2]，则输出的是一维数组[7,8,9]。

2*2*3 的数组（2 组 2 行 3 列）：

```
In [83]: arr3d = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])

In [84]: arr3d
Out[84]: array([[[ 1,  2,  3],
                 [ 4,  5,  6]],

                [[ 7,  8,  9],
                 [10, 11, 12]]])

In [85]: arr3d[0]
Out[85]: array([[1, 2, 3],
                [4, 5, 6]])

In [86]: arr3d[0] = 42

In [87]: arr3d
Out[87]: array([[[42, 42, 42],
                 [42, 42, 42]],

                [[ 7,  8,  9],
                 [10, 11, 12]]])
```

切片索引

```
In [88]: arr[1:6]
Out[88]: array([ 1,  2,  3,  4, 12])

In [89]: arr2d
Out[89]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [90]: arr2d[:2]
Out[90]: array([[1, 2, 3],
                [4, 5, 6]])

In [91]: arr2d[:2, :1]
Out[91]: array([[1],
                [4]])

In [92]: arr2d[:2, 1:]
Out[92]: array([[2, 3],
                [5, 6]])

In [93]: arr2d[1, :2]
Out[93]: array([4, 5])

In [94]: arr2d[2, :1]
Out[94]: array([7])

In [95]: arr2d[:, :1]
Out[95]: array([[1],
                [4],
                [7]])
```

布尔型索引

```
In [4]: data = randn(7,4)
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-7c99198be18a> in <module>()
----> 1 data = randn(7,4)

NameError: name 'randn' is not defined
```

需要先引入：**from** numpy.random **import** randn

或将代码改成：data = np.random.randn(7, 4)

```
In [6]: names
Out[6]: array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'],
              dtype='<U4')

In [7]: data
Out[7]: array([[ 1.67169932, -0.02625055, -0.1215976 ,  1.19701455],
               [ 0.82737515, -0.49482389, -0.22104016, -0.06203454],
               [-0.05667422, -0.08610461, -0.88145237, -1.54452162],
               [-0.11777671,  2.22354631,  1.41530186,  0.20124233],
               [-2.1729956 ,  0.49395586, -0.96438951, -0.98995052],
               [-0.38029473, -0.45216299,  0.01849184, -1.03100717],
               [-0.29125522, -0.14626425,  1.25064415,  0.72695862]])

In [8]: names == 'Bob'
Out[8]: array([ True, False, False,  True, False, False, False], dtype=bool)

In [9]: data[names == 'Bob']
Out[9]: array([[ 1.67169932, -0.02625055, -0.1215976 ,  1.19701455],
               [-0.11777671,  2.22354631,  1.41530186,  0.20124233]])

In [10]: data[names == 'Bob', 2:]
Out[10]: array([[-0.1215976 ,  1.19701455],
                [ 1.41530186,  0.20124233]])

In [11]: data[names == 'Bob', 3]
Out[11]: array([ 1.19701455,  0.20124233])
```

布尔型数组的长度必须跟被索引的轴长度一致。每个名字对应 data 数组一行。

对条件进行否定的两种方式：

```
In [12]: names != 'Bob'
Out[12]: array([False,  True,  True, False,  True,  True,  True], dtype=bool)
```

```
In [14]: data[~(names == 'Bob')]
Out[14]: array([[ 0.82737515, -0.49482389, -0.22104016, -0.06203454],
                [-0.05667422, -0.08610461, -0.88145237, -1.54452162],
                [-2.1729956 ,  0.49395586, -0.96438951, -0.98995052],
                [-0.38029473, -0.45216299,  0.01849184, -1.03100717],
                [-0.29125522, -0.14626425,  1.25064415,  0.72695862]])
```

组合应用多个布尔条件，可使用&、|等布尔算术运算符：

```
In [16]: mask = (names == 'Bob')|(names == 'Will')

In [17]: mask
Out[17]: array([ True, False,  True,  True,  True, False, False], dtype=bool)
```

通过布尔型索引选取数组中的数组，将总是创建数据的副本，即使返回一模一样的数组也是一样。

通过布尔型数组设置值：

```
In [18]: data[data < 0] = 0

In [19]: data

Out[19]: array([[ 1.67169932,  0.        ,  0.        ,  1.19701455],
                [ 0.82737515,  0.        ,  0.        ,  0.        ],
                [ 0.        ,  0.        ,  0.        ,  0.        ],
                [ 0.        ,  2.22354631,  1.41530186,  0.20124233],
                [ 0.        ,  0.49395586,  0.        ,  0.        ],
                [ 0.        ,  0.        ,  0.01849184,  0.        ],
                [ 0.        ,  0.        ,  1.25064415,  0.72695862]])
```

通过一维布尔数组设置整行或列的值：

```
In [23]: data[names != 'Joe']= 7

In [24]: data

Out[24]: array([[ 7.        ,  7.        ,  7.        ,  7.        ],
                [ 0.82737515,  0.        ,  0.        ,  0.        ],
                [ 7.        ,  7.        ,  7.        ,  7.        ],
                [ 7.        ,  7.        ,  7.        ,  7.        ],
                [ 7.        ,  7.        ,  7.        ,  7.        ],
                [ 0.        ,  0.        ,  0.01849184,  0.        ],
                [ 0.        ,  0.        ,  1.25064415,  0.72695862]])
```

花式索引

指利用整数数组进行索引。

np.**empty**((8,4))

Return a **new array** of given shape **and** type, without initializing entries.

for i in range(8):

arr[i] = i

Return an object that produces a sequence of integers from start (inclusive)

to stop (exclusive) by step.

```
In [31]: arr
Out[31]: array([[ 0.,  0.,  0.,  0.],
                [ 1.,  1.,  1.,  1.],
                [ 2.,  2.,  2.,  2.],
                [ 3.,  3.,  3.,  3.],
                [ 4.,  4.,  4.,  4.],
                [ 5.,  5.,  5.,  5.],
                [ 6.,  6.,  6.,  6.],
                [ 7.,  7.,  7.,  7.]])

In [32]: arr[[4,3,0,6]]
Out[32]: array([[ 4.,  4.,  4.,  4.],
                [ 3.,  3.,  3.,  3.],
                [ 0.,  0.,  0.,  0.],
                [ 6.,  6.,  6.,  6.]])

In [33]: arr[[-3,-5,-7]]
Out[33]: array([[ 5.,  5.,  5.,  5.],
                [ 3.,  3.,  3.,  3.],
                [ 1.,  1.,  1.,  1.]])
```

为了以特定顺序选取行的子集，只需传入一个用于指定顺序的整数列表或ndarray，使用负数索引会从末尾开始选取行（最后一行是 -1）。

一次传入多个索引组，返回一个一维数组：

```
In [43]: arr[[1,5,7,2]]

Out[43]: array([[ 4,  5,  6,  7],
                [20, 21, 22, 23],
                [28, 29, 30, 31],
                [ 8,  9, 10, 11]])

In [44]: arr[[0,3,1,2]]

Out[44]: array([[ 0,  1,  2,  3],
                [12, 13, 14, 15],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11]])

In [45]: arr[[1,5,7,2],[0,3,1,2]]

Out[45]: array([ 4, 23, 29, 10])
```

取整列的两种方法，相当于给列排了顺序：

```
In [46]: arr[[1,5,7,2]][:,[0,3,1,2]]

Out[46]: array([[ 4,  7,  5,  6],
                [20, 23, 21, 22],
                [28, 31, 29, 30],
                [ 8, 11,  9, 10]])

In [47]: arr[np.ix_([1,5,7,2],[0,3,1,2])]

Out[47]: array([[ 4,  7,  5,  6],
                [20, 23, 21, 22],
                [28, 31, 29, 30],
                [ 8, 11,  9, 10]])
```

花式索引跟切片不一样，总是将数据复制到新数组中。

数组转置和轴对换

转置返回的是源数据的视图，不进行任何复制操作。数组有 transpose 方法，还有一个 T 属性来完成转置：

```
In [49]: arr
Out[49]: array([[ 0,  1,  2,  3,  4],
                [ 5,  6,  7,  8,  9],
                [10, 11, 12, 13, 14]])

In [50]: arr.T
Out[50]: array([[ 0,  5, 10],
                [ 1,  6, 11],
                [ 2,  7, 12],
                [ 3,  8, 13],
                [ 4,  9, 14]])
```

高维数组

Transpose 要一个轴编号：

```
In [56]: arr
Out[56]: array([[[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7]],

                [[ 8,  9, 10, 11],
                 [12, 13, 14, 15]]])

In [57]: arr.transpose((1,0,2))
Out[57]: array([[[ 0,  1,  2,  3],
                 [ 8,  9, 10, 11]],

                [[ 4,  5,  6,  7],
                 [12, 13, 14, 15]]])
```

arr 是 2 组 2 行 4 列的数组，transpose 的参数表示 shape 的形状，对于这个例子来说，即 2[0]、2[1]、4[2]，transpose(1,0,2)转置后变为 2[1]、2[0]、4[2]，看起来仍是 2 组 2 行 4 列的形状，但数组内的元素经过转换后索引已经改变，也要遵循（1，0，2）的顺序。如转置前的数组 arr[0,1,0]索引值为 4，转置后的数组 arr'[1,0,0]，索引值才为 4。其它同理。

ndarray 的 swapaxes 方法接受一对轴编号且返回源数据的视图：

```
In [58]: arr.T
Out[58]: array([[[ 0,  8],
                 [ 4, 12]],

                [[ 1,  9],
                 [ 5, 13]],

                [[ 2, 10],
                 [ 6, 14]],

                [[ 3, 11],
                 [ 7, 15]]])

In [59]: arr.swapaxes
Out[59]: <function ndarray.swapaxes>

In [60]: arr.swapaxes(1, 2)
Out[60]: array([[[ 0,  4],
                 [ 1,  5],
                 [ 2,  6],
                 [ 3,  7]],

                [[ 8, 12],
                 [ 9, 13],
                 [10, 14],
                 [11, 15]]])
```

转置后的数组 arr.T 为 4[2] 组 2[1] 行 2[0] 列数组，swapaxes(1,2)就是将第二个维度（中括号内数字）和第三个维度交换，即转换为 2 组 4 行 2 列。

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，就是一些简单函数。

利用数组进行数据处理

用数组表达式代替循环的做法，通常被称为矢量化。NumPy 数组将多种数据处理任务表述为数组表达式。

```
In [18]: points = np.arange(-5, 5, 0.01)

In [19]: points
                -4.82000000e+00,  -4.81000000e+00,  -4.80000000e+00,
                -4.79000000e+00,  -4.78000000e+00,  -4.77000000e+00,
                -4.76000000e+00,  -4.75000000e+00,  -4.74000000e+00,
                -4.73000000e+00,  -4.72000000e+00,  -4.71000000e+00,
                -4.70000000e+00,  -4.69000000e+00,  -4.68000000e+00,
                -4.67000000e+00,  -4.66000000e+00,  -4.65000000e+00,
                -4.64000000e+00,  -4.63000000e+00,  -4.62000000e+00,
                -4.61000000e+00,  -4.60000000e+00,  -4.59000000e+00,
                -4.58000000e+00,  -4.57000000e+00,  -4.56000000e+00,
                -4.55000000e+00,  -4.54000000e+00,  -4.53000000e+00,
                -4.52000000e+00,  -4.51000000e+00,  -4.50000000e+00,
                -4.49000000e+00,  -4.48000000e+00,  -4.47000000e+00,
                -4.46000000e+00,  -4.45000000e+00,  -4.44000000e+00,
                -4.43000000e+00,  -4.42000000e+00,  -4.41000000e+00,
                -4.40000000e+00,  -4.39000000e+00,  -4.38000000e+00,
                -4.37000000e+00,  -4.36000000e+00,  -4.35000000e+00,
                -4.34000000e+00,  -4.33000000e+00,  -4.32000000e+00,
                -4.31000000e+00,  -4.30000000e+00,  -4.29000000e+00,
                -4.28000000e+00,  -4.27000000e+00,  -4.26000000e+00,
                -4.25000000e+00,  -4.24000000e+00,  -4.23000000e+00,

In [20]: xs, ys = np.meshgrid(points, points)

In [21]: xs
Out[21]: array([[-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
                [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
                [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
                ...,
                [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
                [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
                [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99]])

In [22]: ys
Out[22]: array([[-5.  , -5.  , -5.  , ..., -5.  , -5.  , -5.  ],
                [-4.99, -4.99, -4.99, ..., -4.99, -4.99, -4.99],
                [-4.98, -4.98, -4.98, ..., -4.98, -4.98, -4.98],
                ...,
                [ 4.97,  4.97,  4.97, ...,  4.97,  4.97,  4.97],
                [ 4.98,  4.98,  4.98, ...,  4.98,  4.98,  4.98],
                [ 4.99,  4.99,  4.99, ...,  4.99,  4.99,  4.99]])
```

np.meshgrid 函数接受两个一维数组，并产生两个二维矩阵（对应于两个数组中所有的（x, y）对。

将条件逻辑表述为数组运算

np.wherea 函数是三元表达式 x if condition else y 的矢量化版本。

```
In [28]: xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])

In [29]: yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])

In [30]: cond = np.array([True, False, True, True, False])

In [31]: result = np.where(cond, xarr, yarr)

In [32]: result
Out[32]: array([ 1.1,  2.2,  1.3,  1.4,  2.5])
```

np.where 的第二个和第三个参数不必是数组，传递给 where 的数组大小可以不相等，甚至可以是标量值。在数据分析工作中，where 通常用于根据另一个数组而产生一个新的数组。

```
In [34]: arr = np.random.randn(4,4)

In [35]: arr
Out[35]: array([[-0.10082533,  0.04670593,  0.45367222,  0.5431559 ],
                [-0.05446967,  1.71919405,  0.82878696,  0.67773252],
                [ 0.95075067,  1.00100095,  1.65505897,  0.30913714],
                [ 0.43594045, -0.14124494,  1.40191718,  0.066239  ]])

In [36]: np.where(arr > 0, 2, -2)
Out[36]: array([[-2,  2,  2,  2],
                [-2,  2,  2,  2],
                [ 2,  2,  2,  2],
                [ 2, -2,  2,  2]])

In [37]: np.where(arr > 0, 2, arr)
Out[37]: array([[-0.10082533,  2.        ,  2.        ,  2.        ],
                [-0.05446967,  2.        ,  2.        ,  2.        ],
                [ 2.        ,  2.        ,  2.        ,  2.        ],
                [ 2.        , -0.14124494,  2.        ,  2.        ]])
```

用 where 表述出更复杂的逻辑：（where 的嵌套）

```
In [38]: cond1 = np.array([True, False, True, True, False])

In [39]: cond2 = np.array([True, True, True, True, False])

In [40]: np.where(cond1 & cond2, 0,
                  np.where(cond1, 1,
                           np.where(cond2, 2, 3)))
Out[40]: array([0, 2, 0, 0, 3])
```

```
In [46]: result = 1 * (cond1 & ~cond2) + 2 * (cond2 & ~cond1) + 3 * ~(cond1|cond2)

In [47]: result
Out[47]: array([0, 2, 0, 0, 3])
```

用于布尔型数组的方法

有两个方法 any 和 all。

```
In [69]: bools = np.array([False, True, False, True])

In [70]: bools.all()
Out[70]: False
```

排序

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：

```
In [78]: arr = np.random.randn(5, 3)

In [79]: arr
Out[79]: array([[ 1.55882011, -0.14961335,  0.26094308],
                [ 0.87969241, -1.24446791,  0.16426709],
                [-0.24708545, -0.85359541,  2.1032816 ],
                [-1.07034015,  0.0833223 , -1.66588773],
                [ 0.01784438, -0.21925355, -0.75199218]])

In [80]: arr.sort(0)

In [81]: arr
Out[81]: array([[-1.07034015, -1.24446791, -1.66588773],
                [-0.24708545, -0.85359541, -0.75199218],
                [ 0.01784438, -0.21925355,  0.16426709],
                [ 0.87969241, -0.14961335,  0.26094308],
                [ 1.55882011,  0.0833223 ,  2.1032816 ]])
```

顶级方法 np.sort 返回的数组已排序的副本，就地排序则会修改数组。

唯一化以及其他的集合逻辑

```
In [88]: names = np.array(['Bob', 'J', 'W', 'B', 'W', 'J'])

In [89]: np.unique(names)
Out[89]: array(['B', 'Bob', 'J', 'W'],
               dtype='<U3')

In [90]: ints =np.array([3,3,3,4,4,5,5,2,2,1,6])

In [91]: np.unique(ints)
Out[91]: array([1, 2, 3, 4, 5, 6])
```

np.unique 找出数组中的唯一值并返回已排序的结果。

```
In [92]: value = np.array([6,0,0,3,2,5,6])

In [93]: np.inld(value, [2])
Out[93]: array([False, False, False, False,  True, False, False], dtype=bool)
```

np.in1d 用于测试一个数组的值在另一个数组的情况。

随机数生成

numpy.random 模块多了用于高效生产多种概率分布的样本值的函数（用来生成大量样本值）。

到这里，numpy 的基础就讲解的差不多了，明后天将讲解 pandas 和 matplotlib。更深入的应用，后面也会分享实际应用这些包得数据分析，欢迎关注！

往期内容：

学习计划丨带你 10 周入门数据分析

如何炼就数据分析的思维？

数据分析惯用的 5 种思维方法

数据分析必备的 43 个 Excel 函数，史上最全！

实操：如何用 Excel 做一次数据分析

写给新人的数据库入门指南

零基础快速自学 SQL，2 天足矣！

数据分析必掌握的统计学知识