数据科学应用入门

老牌三剑客：Numpy、 Pandas和Scipy

Numpy 简介

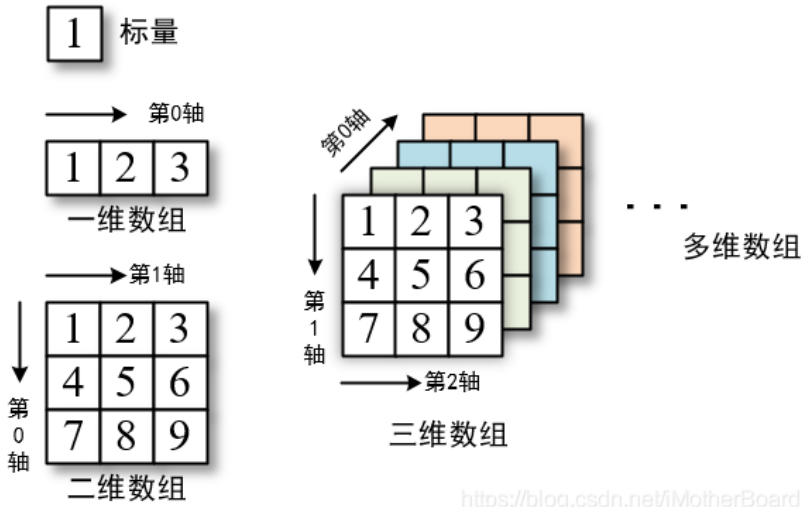
- **Numpy**是Python的一个很重要的第三方库，很多其他科学计算的第三方库都是以Numpy为基础建立的。
- **Numpy**的一个重要特性是它的数组（矩阵\线性代数）计算。
- **Numpy** 不需要使用 for 或者 while 循环就可以完成整个数组的运算

导入 NumPy包

- 命令行安装包：pip install numpy
- import numpy
- import numpy as np
- from numpy import *
- from numpy import array, sin

多维数组对象 ndarray

- NumPy 中利用一种 N 维数组对象 ndarray 来存储和处理数据。该对象可以定义维度，适合做代数运算。虽然看上去和 Python 内置的列表相似，但 ndarray 是标量元素之间的运算。
  - 对应数学上的行列式/向量计算（在线性代数、统计中大量应用）



多维数组对象 ndarray

- 已知人民币兑 100 外币的外汇牌价汇率为：美元 680.6、欧元 800.05、英镑 882.6 和港币 87.79。现有 10 万元人民币，计算可以兑换的外币数量。

```
exchange_rate = [680.6, 800.05, 882.6, 87.79]
for rate in exchange_rate:
    print(round((100000 * 100) / rate), end=' ')
```

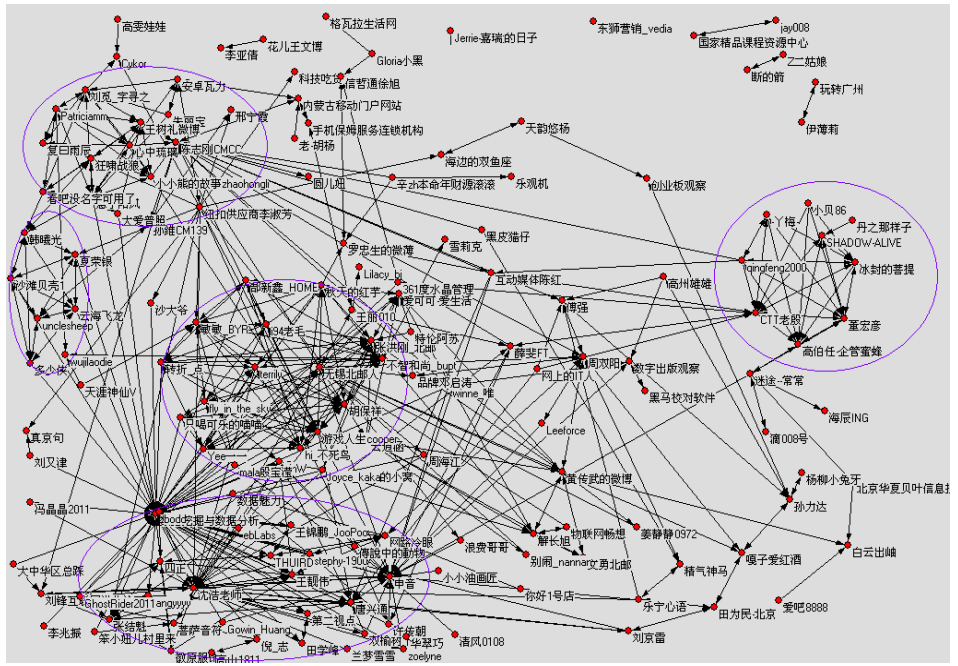
14693 12499 11330 113908

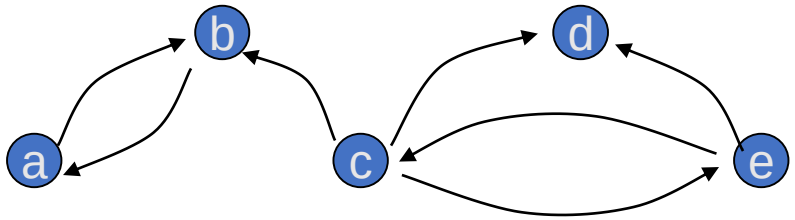
```
import numpy as np
rate_array = np.array([680.6, 800.05, 882.6, 87.79])
np.round((100000 * 100) / rate_array)
```

[14693., 12499., 11330., 113908.]


STORYMAN
老番茄

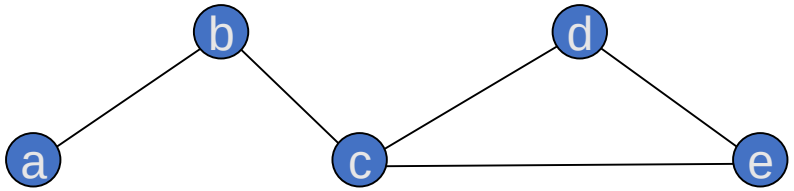

人立方
beijing2008
郭晶晶
吴敏霞
王楠
王励勤
cnBeta.com





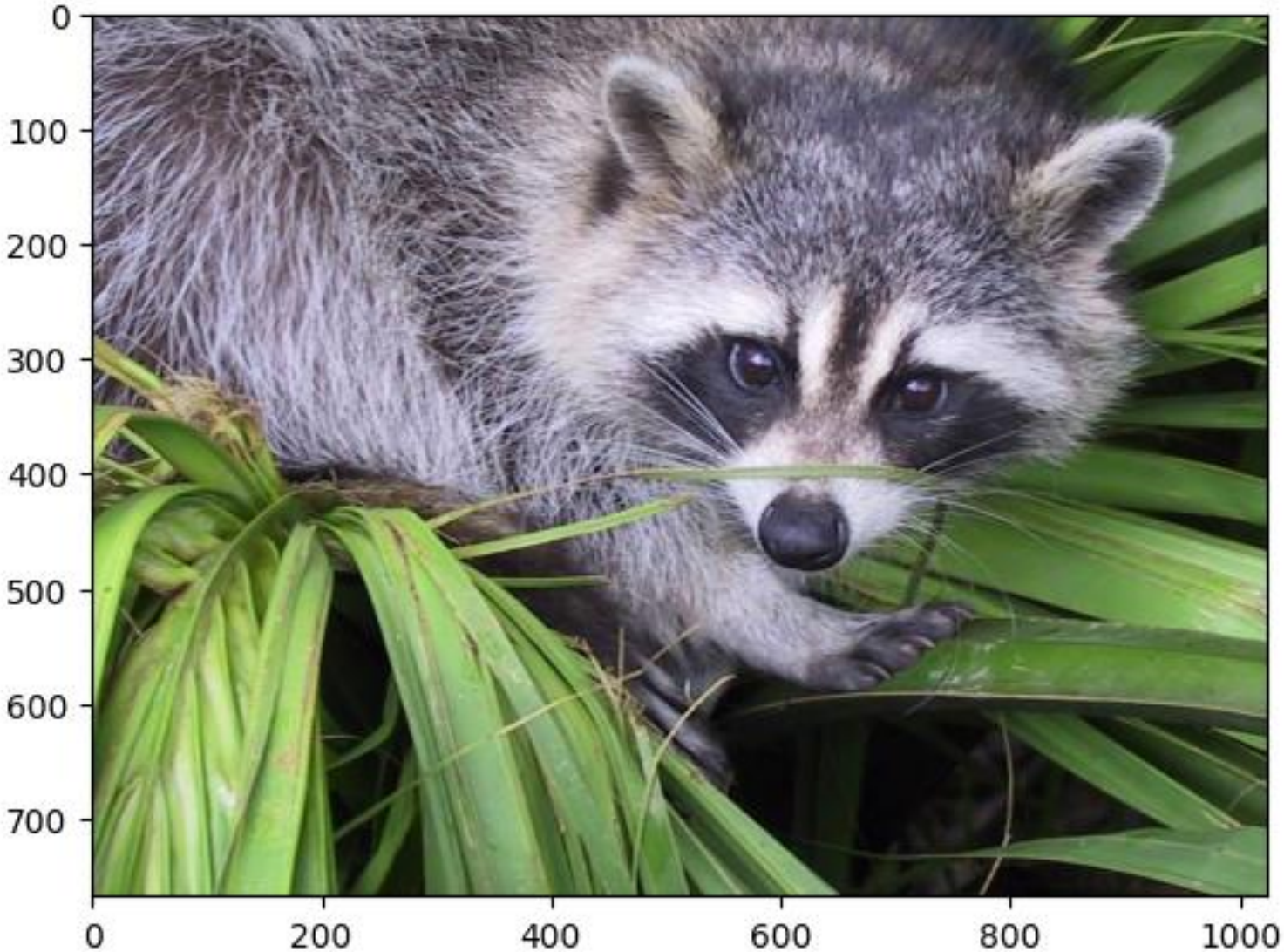
Directed, binary

	a	b	c	d	e
a		1			
b	1				
c		1		1	1
d					
e			1	1	

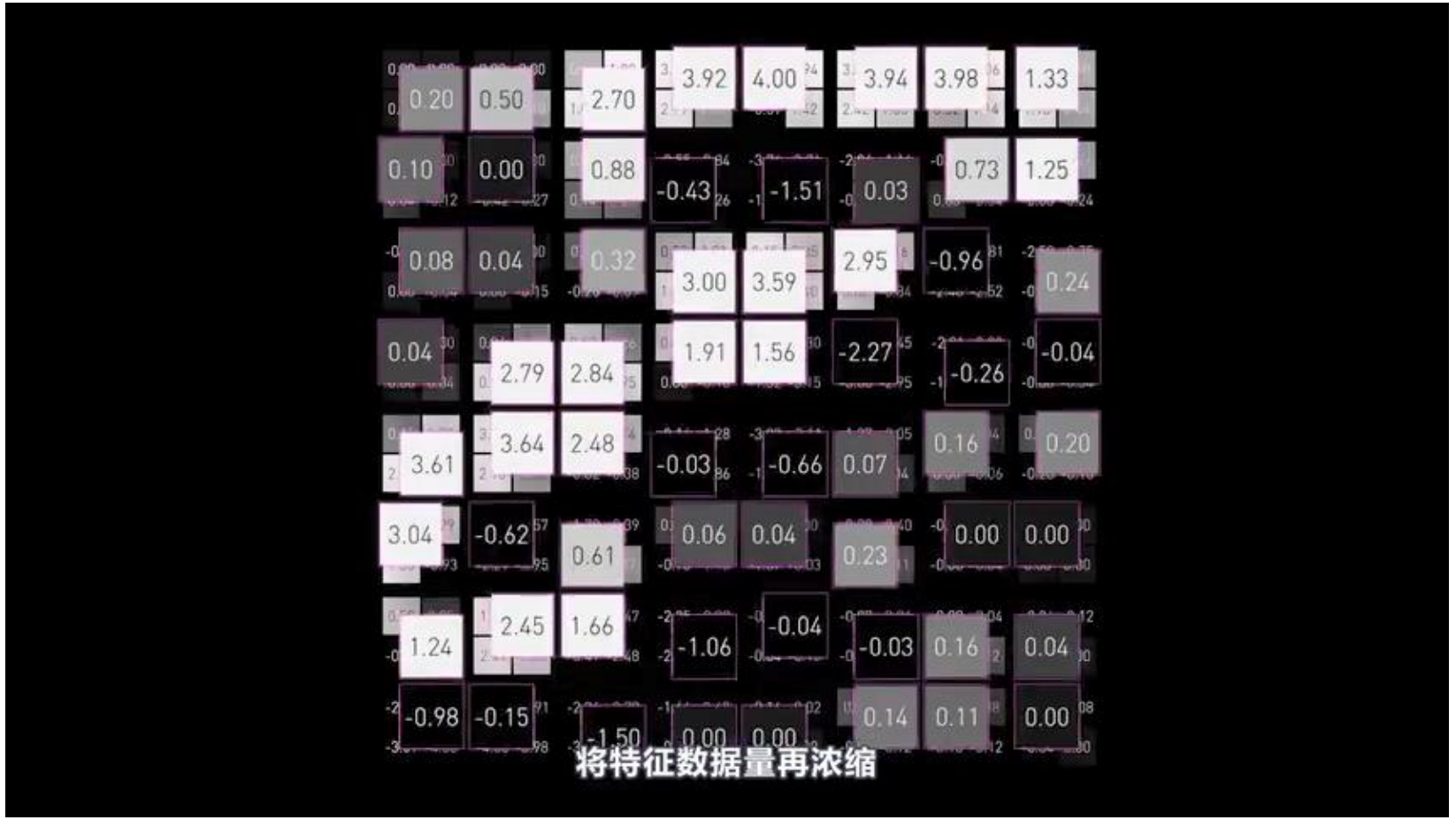


Undirected, binary

	a	b	c	d	e
a		1			
b	1		1		
c		1		1	1
d			1		1
e			1	1	


0
100
200
300
400
500
600
700
0
200
400
600
800
1000

看段视频吧


0.20 0.50 2.70 3.92 4.00 3.94 3.98 1.33
0.10 0.00 0.88 -0.43 -1.51 0.03 0.73 1.25
0.08 0.04 0.32 3.00 3.59 2.95 -0.96 0.24
0.04 2.79 2.84 1.91 1.56 -2.27 -0.26 -0.04
3.61 3.64 2.48 -0.03 -0.66 0.07 0.16 0.20
3.04 -0.62 0.61 0.06 0.04 0.23 0.00 0.00
1.24 2.45 1.66 -1.06 -0.04 -0.03 0.16 0.04
-0.98 -0.15 -1.50 0.00 0.00 0.14 0.11 0.00
将特征数据量再浓缩

- pandas（Python Data Analysis Library）是基于numpy的数据分析模块，提供了大量标准数据模型和高效操作大型数据集所需要的工具

- matplotlib模块依赖于numpy，绘制多种形式的图形，包括线图、直方图、饼状图、散点图、误差线图等等，图形质量可满足出版要求，是数据可视化的重要工具。

- 再往后就是sklearn，tensorflow等机器学习包…

创建 ndarray 数组

- 常用创建 ndarray 数组的方法有：array()、arange()、ones()、zeros()、empty() 和 full() 等。

- 通过 NumPy 的 array() 方法， 可以将输入的列表、元组、数组或其他序列类型转换为 ndarray 数组。
  - Numpy.array(object, dtype=None, copy=True, ndmin=0)
  - 参数：
    - object: 任何具有数组接口方法的对象(列表，元组等）
    - dtype：数据类型
    - copy：布尔型, 可选参数，默认为true，则object对象被复制;
    - ndmin: 指定生成数组的最小维数

- 生成数组

```
>>> np.array([1, 2, 3, 4, 5])          # 把列表转换为数组
array([1, 2, 3, 4, 5])
>>> np.array((1, 2, 3, 4, 5))          # 把元组转换成数组
array([1, 2, 3, 4, 5])
>>> np.array(range(5))                 # 把range对象转换成数组
array([0, 1, 2, 3, 4])
>>> np.array([[1, 2, 3], [4, 5, 6]]) # 二维数组
array([[1, 2, 3],
       [4, 5, 6]])
```

array() 方法

- dtype 属性

类型	类型代码	说明
int8、uint8	i1、u1	有符号、无符号 8 位整型
int16、uint16	i2、u2	有符号、无符号 16 位整型
int32、uint32	i4、u4	有符号、无符号 32 位整型
int64、uint64	i8、u8	有符号、无符号 64 位整型
float32	f4	标准单精度浮点数
float64	f8	标准双精度浮点数
bool		布尔类型
object		Python 对象类型
unicode_	U	固定长度 unicode 类型，如 U4

ndarray 数组的属性

1. dtype 属性

   在 NumPy 数组中， 要求所有元素的数据类型是一致的。 常用的 NumPy 数据类型如表所示。

2. ndim 属性

   NumPy 数组可以是多维的。ndim 属性用来查看数组的维数。

3. shape 属性

   通过 shape 属性得到的是数组的形状。 例如，上面代码中的 n_array 数组， 可以通过 shape 属性查看其形状。

```
>> rate_array = np.array([680.6, 800, 882.6, 87.79])
>> print(rate_array.dtype)
float64

>> rate_array = np.array([680.6, 800, 882.6, 87.79], int32)
>> print(rate_array)

[680.6  800.   882.6   87.79]
```

array() 方法

- 通过 NumPy 的 array() 方法， 可以将输入的列表、元组、数组或其他序列类型转换为 ndarray 数组。

```
>> int_array = np.array([1, 2, 3, 4, 5])
>> print(int_array)
Output:  [1, 2, 3, 4, 5]

>> int_array = np.array([1, 2, 3, 4, 5], dtype=np.float64)
>> int_array
Output:  [1., 2., 3., 4., 5.]
```

array() 方法

- **ndim 属性**
- **shape 属性**

```
>> nd = [[1,2,3],[3,2,1]]
>> n_array = np.array(nd, ndmin=3)
>> print(n_array)
>> print(n_array.shape)
```

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

数组的形状, 第一维一组数据，第二维两组数据，第三维三组数据

array() 方法

- 复制数组

```
>> n1 = np.array([1,2,3])
>> n2 = np.array(n1, copy=True)
>>print(n1,n2)
```

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

arange() 方法

- 在 Python 内置函数中，可以利用 range() 函数得到产生整数的一个可迭代对象。与该函数类似，在 NumPy 中，通过 arange() 方法可以得到一个 NumPy 数组。

arange([start,] stop[, step,], dtype=None)

- 其中，start 是可选参数，代表起始值，默认值为0。stop 是截止值，和 range() 函数一样，结果中并不包含该值。step 也是可选参数，设置的是步长。dtype 设置数据类型，默认为 None 值，NumPy 将推测数组最终的数据类型。

arange() 方法

```
>>> np.arange(5)
>>> [0, 1, 2, 3, 4]

>>> np.arange(1, 10, 2)
array([1, 3, 5, 7, 9])

>>> np.arange(5.0)
[0., 1., 2., 3., 4.]

>>> np.arange(2, 5.0)
[2., 3., 4.]
```

其他数组生成方法

- 这些方法语法结构相似， 均可以接受一个元组作为数组的形状且 dtype 指定数据类型。

- ones() 方法：生成全 1 数组。
- zeros() 方法：生成全 0 数组。
- empty() 方法：生成新数组，只分配空间不填充任何值（值不确定）
- full() 方法：利用指定值填充生成的数组。

ones()、zeros()、empty() 和 full() 方法

```
>> np.ones((2, 3))
>> [ [1., 1., 1.]
      [1., 1., 1.] ]
```

```
>> np.empty((2, 2))
>> [[？ ？ ]
      [？ ？ ]]
```

```
>> np.zeros((2, 3), dtype=np.int32)
>> [ [0, 0, 0]
      [0, 0, 0] ]
```

```
>>np.full((2, 2), 5)
>> [[5 5]
      [5 5]]
```

随机生成数组

- 随机生成数组主要使用Numpy中的random模块

- random.rand(): 用于生成（0，1）之间的随机数组
  - numpy. random.rand(d0,d1,d2…dn)
  - d0 – dn为整数，表示维度，可以为空

- random.randint(): 用于生成一定范围内的随机整数数组
  - numpy.random.randint(low, high,size)
  - low: 低值（起始值），整数。
  - high: 高值（终止值），整数，可以为空。
  - size: 数组维数，整数或者元组，整数表示一维数组，元组表示多维数组。默认为空，返回一个整数。
  - 生成数组的范围从低（包括）到高（不包括），即[low, high)。
  - 如果没有写参数high的值，则返回[0,low)的值。

rand()方法

```
>>n1=np.random.rand(5)
>>print(n1)
[0.13909826 0.24922103 0.96310374 0.37019871 0.41997613]

>>n2=np.random.rand(2,5)
>>print(n2)
[[0.58521035 0.95233726 0.39045262 0.17489286 0.85297126]
 [0.98256282 0.50466043 0.6803248  0.37868105 0.00274871]]
```

randint()方法

```
n0=np.random.randint(1)
print(n0)
```

0

```
n1=np.random.randint(1,None,5)
print(n1)
```

[0 0 0 0 0]

```
n2=np.random.randint(1,5)
print(n2)
```

2

```
n3=np.random.randint(2,5,3)
print(n3)
```

[3 4 4]

```
n4=np.random.randint(2,5,(3,2))
print(n4)
```

[[3 2]
[2 4]
[4 4]]

访问数组元素

- 对该数组中元素的访问，可以使用类似列表的索引和切片等方式。在 NumPy 中，访问方式得到了更大的扩展。
- **1　普通索引**
- 与 Python 内置列表相似，可以通过下标索引（整数索引）的方式获取 NumPy 数组中的元素
- 语法：x[obj]

```
>> close_price = np.array([[3.22, 3.21, 3.20, 3.19, 3.20],
                [3.19, 3.16, 3.17, 3.15, 3.15],
                [4.96, 4.93, 4.92, 4.90, 4.89],
                [6.21, 6.19, 6.13, 6.17, 6.20]])
>> print(close_price[0])
>> print(close_price[0][1])

[3.22 3.21 3.2  3.19 3.2 ]
3.21
```

访问数组元素

- **2　切片**
- 切片用于获取数组中的一小块数据，其表示形式和 Python 内置列表的切片相似：在中括号中用冒号连接给出切片起始位置、截止位置（不包含）和步长。
- 语法：[起始索引：终止索引：步长]

```
>> close_price = np.array([[3.22, 3.21, 3.20, 3.19, 3.20],
                [3.19, 3.16, 3.17, 3.15, 3.15],
                [4.96, 4.93, 4.92, 4.90, 4.89],
                [6.21, 6.19, 6.13, 6.17, 6.20]])
>> print(close_price[0:2])
>> print(close_price[:2])
>> print(close_price[0:3, 1:3])
>> print(close_price[0:2, 0::2])
```

```
[[3.22 3.21 3.2  3.19 3.2 ]
 [3.19 3.16 3.17 3.15 3.15]]

[[3.22 3.21 3.2  3.19 3.2 ]
 [3.19 3.16 3.17 3.15 3.15]]

[[3.21 3.2 ]
 [3.16 3.17]
 [4.93 4.92]]

[[3.22 3.2  3.2 ]
 [3.19 3.17 3.15]]
```

访问数组元素

- **2　切片**
- 在对切片赋值时，可以将一个标量值赋给切片，这样的赋值操作会被扩散到整个切片区域
- 也可以使用数组或列表对切片赋值。数组或列表的形状与切片的形状一致或者可广播（广播指的是不同形状的 NumPy 数组之间运算的执行方式，是 NumPy 数组非常强大的功能）。

```
>> close_price = np.array([[3.22, 3.21, 3.20, 3.19, 3.20],
                           [3.19, 3.16, 3.17, 3.15, 3.15],
                           [4.96, 4.93, 4.92, 4.90, 4.89],
                           [6.21, 6.19, 6.13, 6.17, 6.20]])
>> close_price[:, -2] = 0
>> print(close_price)

>> close_price[:,-2] = [3.19, 3.15, 4.9, 6.17]
>> print(close_price)
```

```
[[3.22, 3.21, 3.2 , 0.  , 3.2 ],
 [3.19, 3.16, 3.17, 0.  , 3.15],
 [4.96, 4.93, 4.92, 0.  , 4.89],
 [6.21, 6.19, 6.13, 0.  , 6.2 ]]

[[3.22, 3.21, 3.2 , 3.19, 3.2 ],
 [3.19, 3.16, 3.17, 3.15, 3.15],
 [4.96, 4.93, 4.92, 4.9 , 4.89],
 [6.21, 6.19, 6.13, 6.17, 6.2 ]]
```

访问数组元素

- NumPy 数组的比较运算是矢量化的，即数组和标量进行比较时，得到的是数组中每个元素与标量比较结果组成的布尔数组

```
>> bank_array = np.array(['中国银行', '农业银行', '工商银行', '建设银行’])
>> print(bank_array == '农业银行’)

[False,  True, False, False]
```

- 可以将这个布尔型的数组用于数组索引。

```
>> close_price = np.array([[3.22, 3.21, 3.20, 3.19, 3.20],
                          [3.19, 3.16, 3.17, 3.15, 3.15],
                          [4.96, 4.93, 4.92, 4.90, 4.89],
                          [6.21, 6.19, 6.13, 6.17, 6.20]])
>> print(close_price[bank_array=='农业银行’])

[[3.19, 3.16, 3.17, 3.15, 3.15]]
```

访问数组元素

- 布尔索引可以和整数索引、切片混合使用。

```
>> close_price = np.array([[3.22, 3.21, 3.20, 3.19, 3.20],
                        [3.19, 3.16, 3.17, 3.15, 3.15],
                        [4.96, 4.93, 4.92, 4.90, 4.89],
                        [6.21, 6.19, 6.13, 6.17, 6.20]])
>> print(close_price[bank_array!='农业银行', 2:])
```

```
[[3.2 , 3.19, 3.2 ],
 [4.92, 4.9 , 4.89],
 [6.13, 6.17, 6.2 ]]
```

- 可以使用 &（和）、|（或）之类的布尔运算符组合多个条件。

```
>> close_price = np.array([[3.22, 3.21, 3.20, 3.19, 3.20],
                        [3.19, 3.16, 3.17, 3.15, 3.15],
                        [4.96, 4.93, 4.92, 4.90, 4.89],
                        [6.21, 6.19, 6.13, 6.17, 6.20]])
>> print(close_price[(bank_array=='中国银行') | (bank_array=='工商银行')])
```

```
[[3.22, 3.21, 3.2 , 3.19, 3.2 ],
 [4.96, 4.93, 4.92, 4.9 , 4.89]]
```

访问数组元素

- **3 排序**
- NumPy 提供了两种结果不同的排序形式，分别返回排序结果和排序后的下标索引。
- **3.1　sort() 排序**
  - 该方法与 Python 列表的 sort() 类似，得到值排序后的结果。不同之处在于，对于多 维数组，NumPy 允许通过 axis 参数指定按照特定维度（轴）进行排序：axis=0 代表按列 对元素进行排序，axis=1 代表按行排序，不输入参数时，默认按行排序。

访问数组元素

- **3.1　sort() 排序**
- 【例】 利用第 11.2 节中的股票收盘价数据， 找出中国银行、农业银行、工商银行和建设银行股票在 2020 年 9 月 21 日至 9 月 25 日的最低收盘价和最高收盘价。

```
close_price = np.array([[3.22, 3.21, 3.20, 3.19, 3.20],
                        [3.19, 3.16, 3.17, 3.15, 3.15],
                        [4.96, 4.93, 4.92, 4.90, 4.89],
                        [6.21, 6.19, 6.13, 6.17, 6.20]])
close_price.sort()
print(close_price)
```

```
[[3.19 3.2  3.2  3.21 3.22]
 [3.15 3.15 3.16 3.17 3.19]
 [4.89 4.9  4.92 4.93 4.96]
 [6.13 6.17 6.19 6.2  6.21]]
```

访问数组元素

- **3.2　argsort() 排序**
- NumPy 数组的 argsort() 方法返回的是按照元素排序后，元素在原数组中的下标列表。该方法同样可以使用 axis 参数改变排序规则。
- 【例】利用第 11.2 节中的股票收盘价数据，将中国银行、农业银行、工商银行和建设银行按 2020 年 9 月 22 日收盘价进行排序。

```
close_price = np.array([[3.22, 3.21, 3.20, 3.19, 3.20],
                        [3.19, 3.16, 3.17, 3.15, 3.15],
                        [4.96, 4.93, 4.92, 4.90, 4.89],
                        [6.21, 6.19, 6.13, 6.17, 6.20]])
bank_array = np.array(['中国银行', '农业银行', '工商银行', '建设银行'])
sorted_index = close_price.argsort(axis=0)
print(sorted_index)
```

```
[[1 1 1 1 1]
 [0 0 0 0 0]
 [2 2 2 2 2]
 [3 3 3 3 3]]
```

数组重塑

- 1　resize() 和 reshape() 方法
- 通过传入新的形状，resize() 和 reshape() 方法都可以改变数组的形状。不同之处在于：resize() 方法是进行“就地修改”（即改变了数组本身），reshape() 方法则是生成新的数组。

```
n1 = np.array([1,2,3,4,5,6])
n1.reshape((2,3))
n2=n1.reshape((2,3))
print(n1)
print(n2)
```

```
[1 2 3 4 5 6]

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

```
n1.resize((2,3))
print(n1)
```

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

数组重塑

- 2　transpose()
- 对于多维数组，可以使用 transpose() 方法来转换维度，产生新的数组。 transpose() 方法不带参数是对数组进行转置， 也可以传入新的维度排序。

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

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

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

数组重塑

- 3　flatten() 方法
- flatten() 方法返回一个折叠成一维的数组。 看上去像是 reshape() 方法的反向操作。

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

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

[1 2 3 4 5 6]
```

数组的增删改

- 1. 数组的增加
- 数组的增加可以按照水平方向增加数据（hstack），也可以按照垂直方向增加数据（vstack）。

```
a = np.array([[1, 2],[3, 4]])
b = np.array([[10, 20],[30, 40]])

print(np.hstack((a,b)))
print(np.vstack((a,b)))
```

```
[[ 1  2 10 20]
 [ 3  4 30 40]]

[[ 1  2]
 [ 3  4]
 [10 20]
 [30 40]]
```

数组的增删改

- 2. 数组的删除
- 对于不想要的数组元素，可以通过索引和切片选取需要的数组元素，或者通过delete函数进行删除。
- np.delete(array,obj,axis)
  - array:需要处理的矩阵
  - obj:需要处理的位置，比如要删除的第一行或者第一行和第二行，行列的序列是从0开始。
  - axis:
    - 如果输入为None：array会先按行展开，然后按照obj，删除第obj-1(从0开始)位置的数，返回一个行矩阵。
    - 如果输入为0：按行删除
    - 如果输入为1：按列删除

数组的增删改

- 2. 数组的删除

```
n1 = np.array([[1, 2],[3, 4],[5,6]])
n2 = np.delete(n1,2,axis=0) #删除第3行
n3 = np.delete(n1,1,axis=1) #删除第2列
n4 = np.delete(n1,(1,2),0) #删除第2行和第3行
print(n1)
print(n2)
print(n3)
print(n4)
```

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

[[1 2]
 [3 4]]

[[1]
 [3]
 [5]]

[[1 2]]
```

数组的增删改

- 3. 数组的修改
- 修改指定数组或数组元素时，直接为数组或数组元素赋值即可。

```
n1 = np.array([[1, 2],[3, 4],[5,6]])
print(n1)

n1[1] = [20,40] #修改第2行数组
n1[2][1] = 88 #修改第3行第2个元素

print(n1)
```

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

[[ 1  2]
 [20 40]
 [ 5 88]]
```

数组间运算

- NumPy 数组的矢量化除了比较， 更多的是应用于批量运算。 数组与标量进行计算操作时，数组中的每个元素均与标量进行运算。对于形状相同的数组之间的算术运算也会应用到元素级。

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

print(a, b)
print(a-b)
print(a*b)
print(a**b)
print(a>2)
print(np.exp(a))
```

```
[1 2 3 4] 2
[-1  0  1  2]
[2 4 6 8]
[ 1  4  9 16]
[False False  True  True]
[ 2.71828183  7.3890561  20.08553692 54.59815003]
```

数组间运算

- NumPy 中有一种对 ndarray 数组中数据执行元素级运算的函数，称为通用函数。 例 11-4 中的 np.round() 函数就是其中一个。 利用通用函数可以不需要使用循环而快速地 进行元素级的运算。 注意， 这些函数是 NumPy 模块中的函数， 因此调用时， 需要使用 np.函数名()或者NumPy数组.函数名()的形式，其中 np 为 NumPy 的别名。

通用函数

- n1 = np.array([[1, 2],[3, 4],[5,6]])
- print(n1)
- print(sum(n1))

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

[ 9 12]
```

函数	说明
sum	对数组内部元素进行求和运算
mean	对数组内部元素进行求均值运算
prod	对数组内部元素进行求乘积运算
max、min	求数组内部元素最大值、最小值
abs	计算整数、浮点数的绝对值
sqrt	计算数组内部元素个平方根
exp	计算数组内部元素的指数 e^x
ceil	计算大于等于数组内部相应元素的最小整数
floor	计算小于等于数组内部相应元素的最大整数
round()	得到每个元素的四舍五入值

通用函数

- mean()函数的功能是求取平均值，经常操作的参数是axis，
  - axis不设置值，对m*n个数求平均值，返回一个实数
  - axis = 0：压缩行，对各列求均值，返回1*n的矩阵
  - axis = 1: 压缩列，对各行求均值，返回m*1的矩阵
- max(), min()的操作类似。

```
n1 = np.array([[1, 2],[3, 4],[5,6]])
n2 = np.mean(n1, axis=0)
n3 = np.mean(n1, axis=1)
n4 = np.mean(n1)

print(n1)
print(n2)
print(n3)
print(n4)
```

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

[3. 4.]

[1.5 3.5 5.5]

3.5
```

尝试自己学习新模块 https://numpy.org/

- https://numpy.org/doc/stable/reference/index.html#reference

Matrix and vector products

dot(a, b[, out])	Dot product of two arrays.
linalg.multi_dot(arrays, *[, out])	Compute the dot product of two or more arrays in a single function call, while automatically selecting the fastest evaluation order.
vdot(a, b, /)	Return the dot product of two vectors.
inner(a, b, /)	Inner product of two arrays.
outer(a, b[, out])	Compute the outer product of two vectors.
matmul(x1, x2, /[, out, casting, order, ...])	Matrix product of two arrays.
tensordot(a, b[, axes])	Compute tensor dot product along specified axes.
einsum(subscripts, *operands[, out, dtype, ...])	Evaluates the Einstein summation convention on the operands.
einsum_path(subscripts, *operands[, optimize])	Evaluates the lowest cost contraction order for an einsum expression by considering the creation of intermediate arrays.
linalg.matrix_power(a, n)	Raise a square matrix to the (integer) power *n*.
kron(a, b)	Kronecker product of two arrays.

Averages and variances

median(a[, axis, out, overwrite_input, keepdims])	Compute the median along the specified axis.
average(a[, axis, weights, returned, keepdims])	Compute the weighted average along the specified axis.
mean(a[, axis, dtype, out, keepdims, where])	Compute the arithmetic mean along the specified axis.
std(a[, axis, dtype, out, ddof, keepdims, where])	Compute the standard deviation along the specified axis.
var(a[, axis, dtype, out, ddof, keepdims, where])	Compute the variance along the specified axis.
nanmedian(a[, axis, out, overwrite_input, ...])	Compute the median along the specified axis, while ignoring NaNs.
nanmean(a[, axis, dtype, out, keepdims, where])	Compute the arithmetic mean along the specified axis, ignoring NaNs.
nanstd(a[, axis, dtype, out, ddof, ...])	Compute the standard deviation along the specified axis, while ignoring NaNs.
nanvar(a[, axis, dtype, out, ddof, ...])	Compute the variance along the specified axis, while ignoring NaNs.

借鉴代码的正确姿势

- 函数和类等模块化知识隐含：为了提高编程效率，需要相互借鉴/继承
- 开源运动使得编程者愿意分享自己的知识（知乎、B站.....）
- vs
- “抄”感觉不太对
- 掌握知识需要勤学苦练

准则1：参考网上例程完成作业是正确的
准则2：独立思考完成作业是正确的，遇到难题时参考准则1
准则3：依赖网上例程，不判断/适应地照搬网上例程是错误的（无论对学习还是未来人生）

国外一开源，国内就自主

2018年的一件事

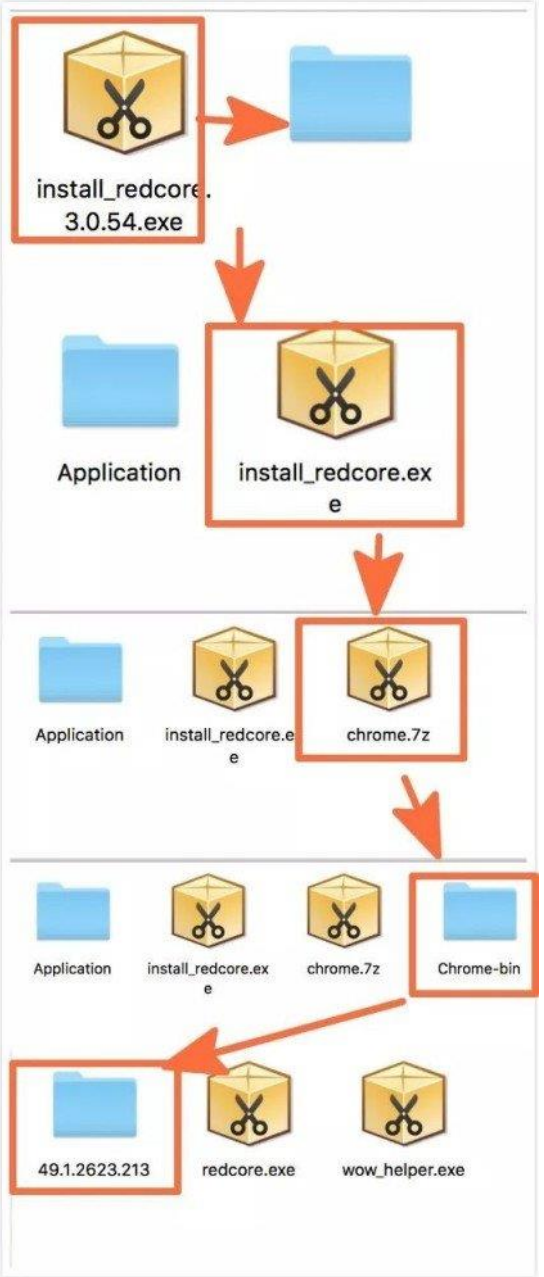
文丨赵宇航

来源丨钛媒体（ID：taimeiti）

昨日（8月16日），一则创投公司的融资信息引起了科技圈的关注，即红芯浏览器被指套壳谷歌浏览器Chrome，还过于强调民族自豪感，夸大自主创新，有网友将其归拢于“汉芯”事件之流。

8月15日，自主研发浏览器核心产品的“红芯”公司宣布完成2.5亿C轮系列融资。本轮投资方是上市公司及政府客户投资，以及晨兴资本、达晨创投、IDG资本等机构继续跟投。红芯创始人兼CEO陈本峰提到，本轮融资主要来源于红芯目标行业市场客户的战略投资，并即将开启新一轮融资。

但随即便出现网友及媒体爆料，红芯浏览器被指是一个经通过谷歌浏览器内核进行二次开发的浏览器，疑似套壳浏览器。



该浏览器解压发现，红芯浏览器最终会得到一个chrome文件，其版本号是49.1.2623.213，该版本正是Chrome最后一个支持XP系统的(过时）版本

其中一个插件“密码管家”的源代码中，一个仅350行的文件有100行代码重复。说明程序员不懂封装，即把代码中相同的部分抽象成一个单独的函数

“如果红芯真的对代码进行了全面掌握和理解，应该会对代码进行大量修订工作。”一位来自公安部系统的专家表示，角度、任务、场景、需求等不同，代码必定不同。此外，红芯使用谷歌老旧版本，既未能自主演进，又无法跟从新版本进化，也表明红芯并未“吃透”源代码，而仅仅是开源代码的“搬运工”

陈本峰，中科大CS本科，港科大CS硕士

高婧，港科大商学院本科

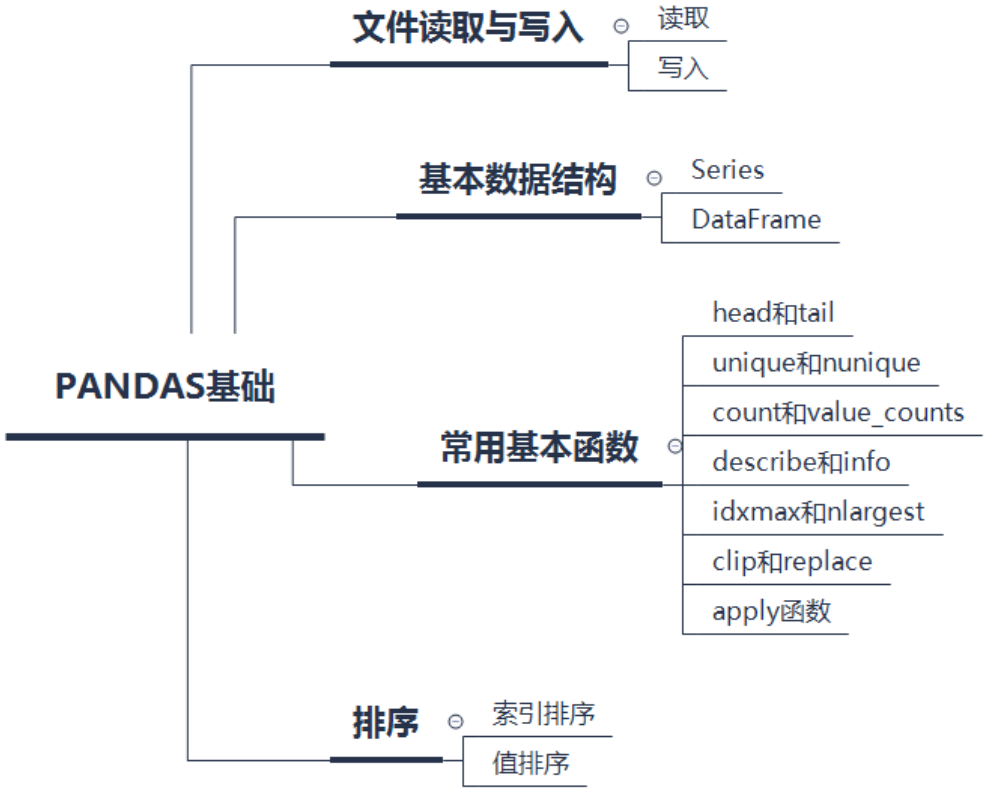
Pandas 数据分析

- Pandas是使用Python语言开发的用于数据处理和数据分析的第三方库。它擅长处理数字型数据和时间序列数据，当然文本型的数据也能轻松处理。

- 换句话说，支持很多应用的文件（office）

```
import pandas as pd
d=pd.read_csv('iris.csv')
```

- Pandas常用的基本功能如下：



如何安装Pandas

- `pip install pandas`

- 引用 Pandas 库并将其缩写为 pd:
- import pandas as pd

打开文件后呢？

- Pandas 中有两种非常重要的数据结构， 分别是 Series 和 DataFrame。Series 类似于 NumPy 中的一维数组， 除了一维数组所用的函数和方法， 还能通过索引进行数据选择， 其索引自动对齐的功能也为计算提供了方便。 而 DataFrame 类似于 NumPy 中的二维数组，同样适用 NumPy 数组的函数和方法。

- 可以从 Pandas 库中单独引入 Series 和 DataFrame：
- from pandas import Series, DataFrame

Pandas和Numpy的区别

- 从功能定位上看:
  - numpy主要是用于数值计算，尤其是矩阵计算
  - pandas主要用于数据处理与分析，支持包括数据读写、数值计算、数据处理、数据分析和数据可视化全套流程操作

- 从数据结构上看:
  - numpy的核心是任意维数的数组，但要求单个数组内所有数据类型必须相同;
  - pandas的核心是series和dataframe，仅支持一维和二维数据，但数据内部可以是异构数据，仅要求同列数据类型一致

Series

- **1. 构建方法**
- Series 由一组数据（可以是不同数据类型）和与之对应的索引组成。 创建 Series 对象可以通过向 pd.Series 传递一个列表、字典或一维数组。

```
import numpy as np
import pandas as pd

s1 = pd.Series([1, 2, 3, np.nan]) # 传入列表
print(s1)

arr = np.arange(5)
s3 = pd.Series(arr) # 传入一维数组
print(s3)
```

```
0    1.0
1    2.0
2    3.0
3    NaN
dtype: float64

0    0
1    1
2    2
3    3
4    4
dtype: int32
```

Series

- **1. 构建方法**
- 在创建 Series 时，也可以传入不同类型的一组数据：

```
dic = {'浦发银行': 9.72, '民生银行': 5.39, '招商银行': 39.78 }
s2 = pd.Series(dic) # 传入字典
print(s2)
```

```
浦发银行    9.72
民生银行    5.39
招商银行   39.78
dtype: float64
```

Series

- **1. 构建方法**
  - 在创建的 Series 中， 元素的数据类型为 Python 基础类型 object，即对象。
  - 由此可以看出，Series 的基本形式由索引和对应值构成。 可以通过 Series 的 values 和 index 属性查看对象值和对应的索引。

```
s4 = pd.Series([9.53,9.62,9.72])
print(s4.values)
print(s4.index)
```

```
[9.53 9.62 9.72]
RangeIndex(start=0, stop=3, step=1)
```

- 在创建 Series 时，也可以指定索引。

```
s5 = pd.Series([1, np.nan, 7.0, 'abc'], index=['a','b','c','d'])
print(s5)
```

```
a      1
b    NaN
c    7.0
d    abc
dtype: object
```

Series

- **2. 数据选择**
  - 与 NumPy 数组不同的是，Pandas 中的 Series 可以通过索引选取某个或某些数据。
  - 可以通过索引在原始数据上直接修改值。

```
s5 = pd.Series([1, np.nan, 7.0, 'abc'], index=['a','b','c','d’])

print(s5['a'])
print(s5[['a','b’]])

s5['d'] = 2
print(s5)
```

```
1

a     1
b    NaN
dtype: object

a     1
b    NaN
c    7.0
d     2
dtype: object
```

Series

- **2. 数据选择**
  - 传入字典时会直接将字典的键作为 Series 的索引， 可以在传入字典的同时指定索引， 则指定索引会与字典先匹配再放到对应的位置上， 若没有匹配的值则会显示 NaN（not a number）， 表示缺失值。
  - Pandas 中的 isnull 和 notnull 函数可以检测缺失数据。

```
dic = {'浦发银行': 9.72,'民生银行': 5.39,'招商银行': 39.78}
index = ['招商银行','浦发银行', '民生银行', '工商银行']
s6 = pd.Series(dic, index)

print(s6)
print(pd.isnull(s6))
```

```
招商银行    39.78
浦发银行     9.72
民生银行     5.39
工商银行      NaN
dtype: float64

招商银行    False
浦发银行    False
民生银行    False
工商银行     True
dtype: bool
```

DataFrame

- **1. 构建方法**
- DataFrame 是一个（二维）表格型数据结构，它由多个列组成，且每列的数据类型可以不同 （字符串、数值等）。DataFrame 既有行索引也有列索引。创建 DataFrame 对象可以通过传入字典或二维数组完成。

```
dic1 = {'浦发银行':{'2020-10-14':9.53,'2020-10-15':9.62,'2020-10-16':9.72},
        '民生银行':{'2020-10-14':5.33,'2020-10-15':5.35,'2020-10-16':5.39}}
df1 = pd.DataFrame(dic1)

print(df1)
```

	浦发银行	民生银行
2020-10-14	9.53	5.33
2020-10-15	9.62	5.35
2020-10-16	9.72	5.39

DataFrame

- **1. 构建方法**
- DataFrame 可以指定列的顺序，这样得到的 DataFrame 就会按照指定方式排列。

```
dic2 = {'浦发银行':{'2020-10-14':9.53,'2020-10-15':9.62,'2020-10-16':9.72},
        '民生银行':{'2020-10-14':5.33,'2020-10-15':5.35,'2020-10-16':5.39}}
df3 = pd.DataFrame(dic2, columns=[‘民生银行’,‘浦发银行’])
df4 = pd.DataFrame(dic2, columns=['民生银行','浦发银行','招商银行'])
print(df3)
print(df4)
```

```
   民生银行  浦发银行
0  5.33      9.53
1  5.35      9.62
2  5.39      9.72
   民生银行  浦发银行  招商银行
0  5.33      9.53      NaN
1  5.35      9.62      NaN
2  5.39      9.72      NaN
```

DataFrame

- 2. 查看数据
- 利用 head() 或 tail() 方法分别查看 DataFrame 头部和尾部的几行数据

```
dic1 = {'浦发银行':{'2020-10-14':9.53,'2020-10-15':9.62,'2020-10-16':9.72},
        '民生银行':{'2020-10-14':5.33,'2020-10-15':5.35,'2020-10-16':5.39}}
df1 = pd.DataFrame(dic1)
print(df1.head(2))
print(df1.tail(2))
```

	浦发银行	民生银行
2020-10-14	9.53	5.33
2020-10-15	9.62	5.35
	浦发银行	民生银行
2020-10-15	9.62	5.35
2020-10-16	9.72	5.39

DataFrame

- 2. 查看数据
- 显示索引、列名以及底层的 NumPy 数据
- 对数据做转置

```
dic1 = {'浦发银行':{'2020-10-14':9.53,'2020-10-15':9.62,'2020-10-16':9.72},
        '民生银行':{'2020-10-14':5.33,'2020-10-15':5.35,'2020-10-16':5.39}}
df1 = pd.DataFrame(dic1)
print(df1.columns)
print(df1.values)
print(df1.T)
```

```
Index(['浦发银行', '民生银行'], dtype='object')
[[9.53 5.33]
 [9.62 5.35]
 [9.72 5.39]]
            2020-10-14  2020-10-15  2020-10-16
浦发银行        9.53        9.62        9.72
民生银行        5.33        5.35        5.39
```

DataFrame

- 3. 数据获取
- 如果想从 DataFrame 中获取某一列数据为一个 Series， 可以通过类似字典标记或属性的方式

方法	说明	
df[val]	从 DataFrame 中选取单列或多列。val 为布尔型数组时， 过滤行；val 为切片时， 行切片	
df.loc[val]	通过 val 选取单行或多行	根据Val进行内容匹配
df.loc[:,val]	通过 val 选取单列或多列	
df.loc[val1, val2]	选取 val1 行、val2 列的值	
df.iloc[val]	通过整数位置，选取单行或多行	根据整数进行索引匹配
df.iloc[:,val]	通过整数位置，选取单列或多列	
df.iloc[val1, val2]	通过整数位置，选取 val1 行、val2 列的值	

DataFrame

- 3. 数据获取

```
dic1 = {'浦发银行':{'2020-10-14':9.53,'2020-10-15':9.62,'2020-10-16':9.72},
        '民生银行':{'2020-10-14':5.33,'2020-10-15':5.35,'2020-10-16':5.39}}
df1 = pd.DataFrame(dic1)

print(df1['浦发银行'])
print(df1[df1['浦发银行'] > 9.6])
print(df1.iloc[1])
print(df1.iloc[1:,:])
print(df1.loc['2020-10-15','浦发银行'])
```

```
2020-10-14    9.53
2020-10-15    9.62
2020-10-16    9.72
Name: 浦发银行, dtype: float64

            浦发银行  民生银行
2020-10-15  9.62  5.35
2020-10-16  9.72  5.39

浦发银行    9.62
民生银行    5.35
Name: 2020-10-15, dtype: float64

            浦发银行  民生银行
2020-10-15  9.62  5.35
2020-10-16  9.72  5.39

9.62
```

DataFrame

- 4. 赋值
- 对 DataFrame 中某一列的值进行修改，可通过直接赋值一个标量值或一组值

```
df5 = pd.DataFrame({'a':[1,2,3],'b':[5,6,7],'c':[9,10,11],'d':[13,14,15]},
index=['x','y','z'])
print(df5)
df5['d'] = 1
print(df5)
df5['d'] = np.arange(3)
print(df5)
```

```
 a  b   c   d
x  1  5   9  13
y  2  6  10  14
z  3  7  11  15

  a  b   c  d
x  1  5   9  1
y  2  6  10  1
z  3  7  11  1

  a  b   c  d
x  1  5   9  0
y  2  6  10  1
z  3  7  11  2
```

DataFrame

- 4. 赋值
- 如果赋值给一个 Series，则会精准匹配对应索引的数值，若 Series 缺失 DataFrame 某些索引，则对应位置为空

```
df5 = pd.DataFrame({'a':[1,2,3],'b':[5,6,7],'c':[9,10,11],'d':[13,14,15]},
index=['x','y','z’])
print(df5)
df5[‘d’] = pd.Series([2,5,9],index = [‘y’,‘z’,‘a’])  #不存在索引’a’
print(df5)
df5.at[df5.index[0], 'a'] = 0 #通过标签赋值
print(df5)
df5.iat[0,3] = 0 #通过位置赋值
print(df5)
df5[df5 > 5] = -df5 #通过 where 操作赋值
print(df5)
```

```
 a b  c   d
x 1 5  9 NaN
y 2 6 10 2.0
z 3 7 11 5.0
  a b  c   d
x 0 5  9 NaN
y 2 6 10 2.0
z 3 7 11 5.0
  a b  c   d
x 0 5  9 0.0
y 2 6 10 2.0
z 3 7 11 5.0
  a b  c   d
x 0 5 -9 0.0
y 2 -6 -10 2.0
z 3 -7 -11 5.0
```

Pandas 的常用方法

- 合并（行）
- Pandas 中提供了大量的能够轻松对 Series 和 DataFrame 对象进行不同逻辑关系的合并操作的方法。首先，可以通过 concat 来连接 pandas 对象：确定按某个轴进行连接（可横向可纵向），也可以指定连接方法。concat 的参数如表所示。

方法	意义
objs	合并的对象集合，可以是 Series、DataFrame
axis	合并方法。默认 0 表示纵向，1 表示横向
join	默认 outer 并集，inner 交集，只有这两种
join_axes	按哪些对象的索引保存
ignore_index	默认 Fasle 忽略，是否忽略原 index
keys	为原始 DataFrame 添加一个键，默认无

Pandas 的常用方法

- 合并(行)

```
s1 = pd.Series(np.random.randint(0, 3, size=3))
s2 = pd.Series(np.random.randint(4, 6, size=3))
print(s1)
print(s2)
s = pd.concat([s1,s2])
print(s)
```

```
0   0
1   2
2   2
dtype: int32
```

```
0   4
1   4
2   5
dtype: int32
```

```
0   0
1   2
2   2
0   4
1   4
2   5
dtype: int32
```

Pandas 的常用方法

- 合并（列）
- 其次，merge 函数通过一个或多个键将数据集的行连接起来。 针对同一个主键存在的两张包含不同特征的表，通过主键的连接，将两张表进行合并。合并之后，两张表的行数不增加，列数是两张表的列数之和。merge 参数及其含义如表所示。

方法	意义
how	数据融合的方法，存在不重合的键，方式：inner、outer、left、right
on	用来对齐的列名，一定要保证左表和右表存在相同的列名
left_on	左表对齐的列，可以是列名，也可以是 DataFrame 同长度的 arrays
right_on	右表对齐的列，可以是列名
left_index	将左表的 index 用作连接键
right_index	将右表的 index 用作连接键
suffixes	左右对象中存在重名列，结果用后缀名区分
copy	默认True。将数据复制到数据结构中，设置为 False 提高性能

Pandas 的常用方法

- 合并（列）

```
df1 = pd.DataFrame({'key': ['one', 'two', 'three'], 'data1': np.arange(3)})
df2 = pd.DataFrame({'key': ['one', 'three', 'four'], 'data2': np.arange(3)})
df3 = pd.merge(df1, df2)
print(df3)
df4 = pd.merge(df1, df2, how='left')
print(df4)
```

```
   key  data1  data2
0   one      0      0
1 three      2      1


    key  data1  data2
0   one      0    0.0
1   two      1    NaN
2 three      2    1.0
```

导入导出数据

- 1 导入excel文件
- 导入excel文件主要使用Pandas的read_excel方法。
- pandas.read_excel (io,sheet_name=0,header=0,names=None,index_col=None, dtype=None)
  - io: 文件路径，要指定文件所在的路径，可以是相对路径，也可以是绝对路径。
  - Sheet_name: 默认值为0，表示导入第一个sheet的数据作为DataFrame对象。
    “1”, 表示导入第二个sheet的数据，以此类推。
    “sheet1”, 表示导入名为sheet1的页中的数据。
    “[0,1,’sheet3’]”, 表示导入第一个、第二个和名为“sheet3”的页中的数据。
  - Header：指定作为列名的行，默认值为0，即取第一行为列名，除此以外为数据。Header=None时，表示不含数据列名。
  - Names：默认为None，表示要使用的列名列表。
  - Index_col: 指定列为索引列，默认值为None，索引0是DataFrame对象的行标签。
  - Dtype: 数据类型名称或字典，默认值为None。例如：{‘a’:np.float64, ‘b’:np.int32}

导入导出数据

- 1 导入excel文件

```
df=pd.read_excel('iris.xlsx')
print(df.head())        #输出工作部第一工作表前5条数据
df=pd.read_excel('iris.xlsx',sheet_name='zc')
print(df.head())        #输出工作表”zc”里面前5条数据
```

导入导出数据

- 2. 导入csv数据
- 主要使用Pandas的read_csv方法
- pandas.read_csv (io,sep=‘,’,header=0,names=None,index_col=None, dtype=None)
- io: 文件路径，要指定文件所在的路径，可以是相对路径，也可以是绝对路径。
- Sep: 字符串分隔符
- Header：指定作为列名的行，默认值为0，即取第一行为列名，除此以外为数据。Header=None时，表示不含数据列名。
- Names：默认为None，表示要使用的列名列表。
- Index_col: 指定列为索引列，默认值为None，索引|0是DataFrame对象的行标签。
- Dtype: 数据类型名称或字典，默认值为None。例如：{‘a’:np.float64, ‘b’:np.int32}

导入导出数据

2. 导入csv数据

```
import pandas as pd
df=pd.read_csv('iris.csv')
print(df.head())        #输出前5条数据
```

导入导出数据

- DataFrame.to_excel(excel_writer, sheet_name='Sheet1', na_rep='', float_format=None, columns=None, header=True, encoding=None)
  - excel_writer :文件路径
  - sheet_name :字符串,默认"Sheet1"，指包含DataFrame的表的名称。
  - na_rep : 字符串,默认' '，缺失数据表示方式
  - float_format : 字符串,默认None，格式化浮点数的字符串
  - columns : 序列,可选，要编写的列
  - header : 布尔或字符串列表，默认为Ture。写出列名。如果给定字符串列表，则假定它是列名称的别名。
  - Encoding:指定Excel文件的编码方式，默认值为None。

导入导出数据

- DataFrame.to_csv(path, sep=‘,’, na_rep=‘’, float_format=None, columns=None, header=True, encoding=None)
  - path:文件路径
  - sep: 分隔符，默认值为“,”。
  - na_rep : 字符串,默认‘ ’，缺失数据表示方式
  - float_format : 字符串,默认None，格式化浮点数的字符串
  - columns : 序列,可选，要编写的列
  - header : 布尔或字符串列表，默认为Ture。写出列名。如果给定字符串列表，则假定它是列名称的别名。
  - Encoding:指定Excel文件的编码方式，默认值为None。

https://pandas.pydata.org/docs/user_guide/io.html

Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	LaTeX		Styler.to_latex
text	XML	read_xml	to_xml
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	to_orc
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

Scipy简易入门

- Scipy是一个用于数学、科学、工程领域的常用软件包。它用于有效计算Numpy矩阵，使Numpy和Scipy协同工作，高效解决问题。
- Scipy包含的功能：最优化、线性代数、积分、插值、拟合、特殊函数、快速傅里叶变换、信号处理、图像处理、常微分方程求解器等
- 应用场景:Scipy是高端科学计算工具包，用于数学、科学、工程学等领域

Scipy简易入门

模块	功能
scipy.cluster	矢量量化 / K-均值
scipy.constants	物理和数学常数
scipy.fftpack	傅里叶变换
scipy.integrate	积分程序
scipy.interpolate	插值
scipy.io	数据输入输出
scipy.linalg	线性代数程序
scipy.ndimage	n维图像包
scipy.odr	正交距离回归
scipy.optimize	优化
scipy.signal	信号处理
scipy.sparse	稀疏矩阵
scipy.spatial	空间数据结构和算法
scipy.special	任何特殊数学函数
scipy.stats	统计

Scipy stats模块概述

- stats模块包括了大量的概率分布，可以根据分为数量分为单变量与单变量。单变量又可根据变量类型分为：连续概率分布和离散概率分布。

模块名	
Continuous distributions	连续概率分布
Multivariate distributions	多变量概率分布
Discrete distributions	离散概率分布
Summary statistics	概括统计
Frequency statistics	频率统计
Correlation functions	相关性分析
Statistical tests	统计测试（假设检验）
Transformations	变换
Statistical distances	距离统计
Contingency table functions	列联表
Plot-tests	图示测试

Scipy stats模块概述

在连续概率分布中存在两个影响分布特性的关键参数:loc:平移系数； scale: 缩放系数。对输出y = (x - loc) / scale

方法	功能
rvs(loc=0, scale=1, size=1, random_state=None)	对变量进行随机取值，通过size指定输出数组尺寸
pdf(x, loc=0, scale=1)	概率密度函数
logpdf(x, loc=0, scale=1)	log形式概率密度函数
cdf(x, loc=0, scale=1)	累计概率分布
logcdf(x, loc=0, scale=1)	log形式累计概率分布
sf(x, loc=0, scale=1)	生存函数，等于1-cdf
ppf(q, loc=0, scale=1)	累计概率分布的反函数Percent point function
isf(q, loc=0, scale=1)	生存函数的反函数
stats(loc=0, scale=1, moments='mv')	状态统计返回Mean('m'), variance('v'), skew('s'), and/or kurtosis('k').
fit(data)	对一组参数进行拟合，找出估计的概率密度系数
median/mean/var/std(loc=0, scale=1)	中值/均值/方差/标准差/
expect(func, args=(), loc=0, scale=1, lb=None, ub=None, conditional=False, **kwds)	关于分布的其他参数， 比如lb下边界，up上边界

Scipy stats模块概述

- 以正态分布的常见需求为例了解scipy.stats的基本使用方法。
- 调用scipy.stats包
- import scipy.stats as st

- 1. 生成服从指定分布的随机数

```
a = st.norm.rvs(loc=0, scale=0.1, size =10)
b = st.norm.rvs(loc=0, scale=0.1, size =(2,2))
print (a)
print (b)
```

```
[ 0.11596917  0.01002346  0.05329526  0.09760632  0.12078423 -0.03895703
  0.10174817 -0.12608227 -0.01950582  0.067366  ]

[[-0.0068651   0.06957532]
 [ 0.10489572 -0.07945789]]
```

Scipy stats模块概述

- 2. 求概率密度函数指定点的函数值
  - stats.norm.pdf正态分布概率密度函数。

```
a = st.norm.pdf(0,loc = 0,scale = 1)
print (a)
b = st.norm.pdf(np.arange(3),loc = 0,scale = 1)
print (b)
```

```
0.398942280401432 7

[0.39894228 0.24197072 0.05399097]
```

Scipy stats模块概述

- 3. 求累计分布函数指定点的函数值
  - stats.norm.cdf正态分布累计概率密度函数。

```
a = st.norm.cdf(0,loc=3,scale=1)
print (a)
b = st.norm.cdf(0,0,1)
print (b)
```

```
0.0013498980316300933

0.5
```

Scipy stats模块概述

- 2. 累计分布函数的逆函数
  - stats.norm.ppf正态分布的累计分布函数的逆函数，即下分位点。

```
a = st.norm.ppf(0.05)
print (a)
b = st.norm.cdf(a)
print (b)
```

```
-1.644853626951472 9

0.04999999999999975
```