Python基础库

本章目录

1.NumPy概述

Python模块-NumPy

numpy

Numpy是一个用python实现的科学计算的扩展程序库，包括：

1、一个强大的N维数组对象Array；

2、比较成熟的（广播）函数库；

3、用于整合C/C++和Fortran代码的工具包；

4、实用的线性代数、傅里叶变换和随机数生成函数。numpy和稀疏矩阵运算包scipy配合使用更加方便。

NumPy（Numeric Python）提供了许多高级的数值编程工具，如：矩阵数据类型、矢量处理，以及精密的运算库。专为进行严格的数字处理而产生。多为很多大型金融公司使用，以及核心的科学计算组织如：Lawrence Livermore，NASA用其处理一些本来使用C++，Fortran或Matlab等所做的任务。

NumPy是什么？

标准的**Python**中用**list**（列表）保存值，可以当做数组使用，但因为列表中的元素可以是任何对象，所以浪费了CPU运算时间和内存。

NumPy诞生为了弥补这些缺陷。它提供了两种基本的对象：

ndarray：全称（n-dimensional array object）是储存单一数据类型的多维数组。

ufunc：全称（universal function object）它是一种能够对数组进行处理的函数。

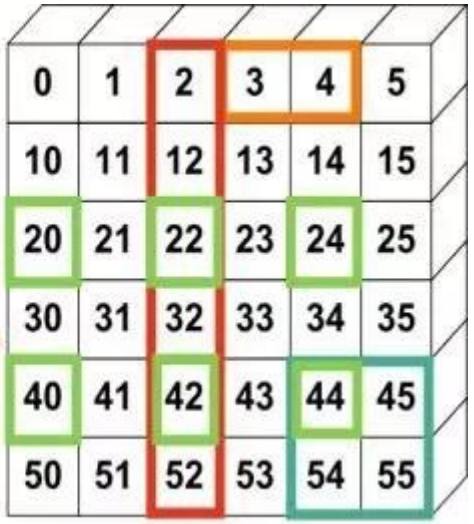
NumPy的官方文档：

https://docs.scipy.org/doc/numpy/reference/

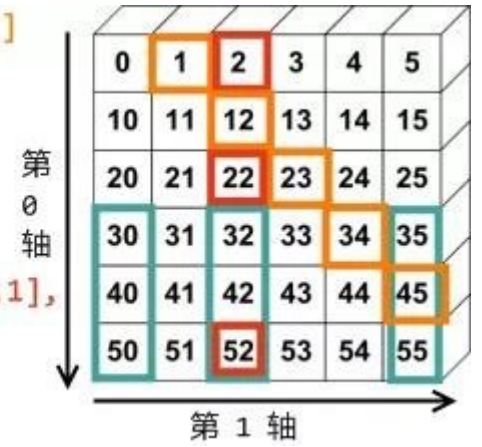
Python模块-NumPy

切片

```
>>> a[0,3:5]
array([3,4])
>>> a[4:,4:]
array([[44,45],[54,55]])
>>> a[:,2]
array([2,12,22,32,42,52])
>>> a[2::2,::2]
array([[20,22,24],
       [40,42,44]])
```

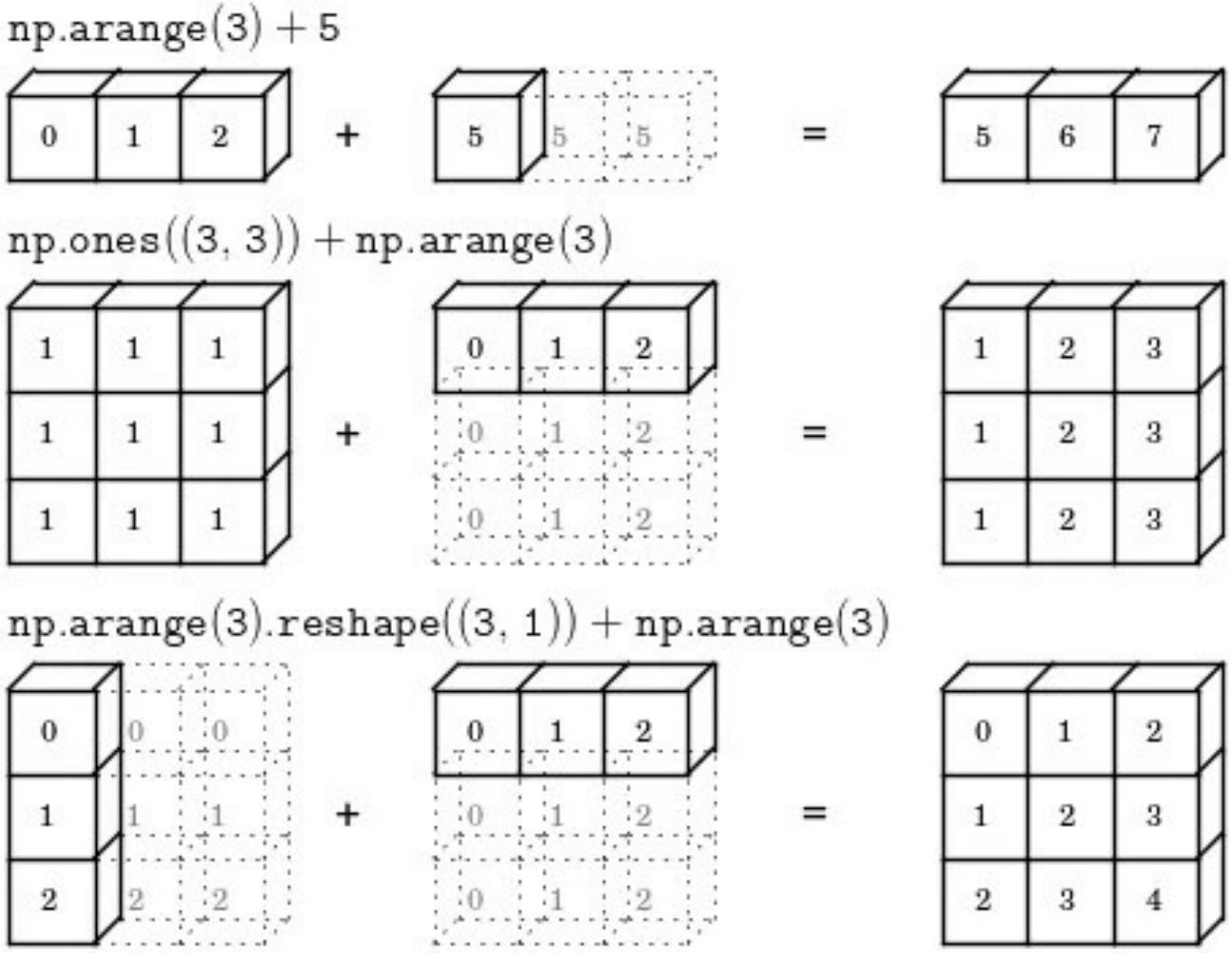



```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([1,12,23,34,45])
>>> a[3:,[0,2,5]]
array([[30,32,35],
       [40,42,45],
       [50,52,55]])
>>> mask=np.array([1,0,1,0,0,1],
                  dtype=np.bool)
>>> a[mask,2]
array([2,22,52])
```



Python模块-NumPy

广播



2.NumPy数组(ndarry)对象

1.1 认识 NumPy 数组对象

NumPy 最重要的一个特点是其 N 维数组对象 **ndarray**，它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。

```
>import numpy as np # 导入NumPy工具包
>data = np.arange(12).reshape(3, 4) # 创建一个3行4列的数组
>data
```

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

ndarray对维数没有限制。

[ ]从内到外分别为第0轴，第1轴，第2轴，第3轴。

1.1 认识 NumPy 数组对象


```
>type(data)
```

numpy.ndarray

```
>data.ndim # 数组维度的个数，输出结果2，表示二维数组
```

2

```
>data.shape # 数组的维度，输出结果(3,4),表示3行4列
```

(3, 4)

```
data.size # 数组元素的个数，输出结果12，表示总共有12个元素
```

12

```
data.dtype # 数组元素的类型，输出结果dtype('int64'),表示元素类型都是int64
```

dtype('int32')

1.2 创建 NumPy 数组


```
>import numpy as np
>data1 = np.array([1, 2, 3]) # 创建一个一维数组
>data1
```

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

```
data2 = np.array([[1, 2, 3], [4, 5, 6]]) # 创建一个二维数组
data2
```

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

```
np.zeros((3, 4))#创建一个全0数组
```

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

1.2 创建 NumPy 数组


```
>np.ones((3, 4))#创建全一数组
```

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

```
>np.empty((5, 2))# 创建全空数组，其实每个值都是接近于零的数
```

```
array([[ 6.95312756e-310,  2.12199579e-314],
       [ 2.12199579e-314,  4.94065646e-324],
       [ 0.00000000e+000, -7.06252554e-311],
       [ 0.00000000e+000, -8.12021073e-313],
       [ 1.29923372e-311,  2.07507571e-322]])
```

1.2 创建 NumPy 数组


```
>np.arange(1, 20, 5)
```

```
array([ 1,  6, 11, 16])
```

```
>np.array([1, 2, 3, 4], float)
```

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

```
>np.ones((2, 3), dtype='float64')
```

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

ndarray的创建

NumPy提供了**专门用于生成ndarray的函数**，提高创建ndarray的速度。

```
> a = np.arange(0, 1, 0.1)
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])

> b = np.linspace(0, 1, 10)
array([ 0.        ,  0.11111111,  0.22222222,  0.33333333,
0.44444444,0.55555556,  0.66666667,  0.77777778,  0.88888889,  1.        ])

> c = np.linspace(0, 1, 10, endpoint=False)
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])

> d = np.logspace(0, 2, 5)
array([   1. ,    3.16227766,   10.,   31.6227766 ,  100.])
```

ndarray的创建

np.empty((2,3), np.int)	创建2*3的整形型空矩阵，只分配内存
np.zeros(4, np.int)	创建长度为4，值为全部为0的矩阵
np.full(4, np.pi)	创建长度为4，值为全部为pi的矩阵

还可以**自定义函数产生ndarray。**

```
> def func(i):
      return i % 4 + 1
> np.fromfunction(func, (10,))

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

fromfunction第一个参数接收计算函数，第二个参数接收数组的形状。

ndarray的属性

ndarray的元素具有相同的元素类型。常用的有int（整型），float（浮点型），complex（复数型）。

```
> a = np.array([1, 2, 3, 4], dtype=float)
array([ 1.,  2.,  3.,  4.])

> a.dtype
dtype('float64')
```

ndarray的shape属性用来获得它的形状，也可以自己指定

```
> c = np.array([[1, 2, 3, 4], [4, 5, 6, 7], [7, 8, 9, 10]])
> c.shape
(3, 4)

> a = np.array([1, 2, 3, 4])
> d = a.reshape((2,2))
array([[1, 2],
       [3, 4]])
```

ndarray的切片

ndarray的切片和list是一样的。

```
> a = np.arange(10)
> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

a[5]    a[3:5]       a[:5]              a[:-1]
----    ------    --------------    -------------------------
5        [3, 4]    [0, 1, 2, 3, 4]       [0, 1, 2, 3, 4, 5, 6, 7, 8]

 a[1:-1:2]              a[::-1]                      a[5:1:-2]
------------    ----------------------------    ---------
[1, 3, 5, 7]       [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]          [5, 3]
```

可以通过切片的对ndarray中的元素进行**更改**。

```
> a[2:4] = 100, 101
> a
array([  0,   1, 100, 101,   4,   5,   6,   7,   8,   9])
```

ndarray的切片

ndarray通过切片产生一个新的数组b，**b和a共享同一块数据存储空间。**

```
> b = a[3:7]
> b[2] = -10
 b                          a
------------------  ----------------------------------------
[101,   4, -10,   6]    [  0,   1, 100, 101,   4, -10,   6,   7,   8,   9]
```

如果想改变这种情况，我们可以**用列表对数组元素切片。**

```
> b = a[[3, 3, -3, 8]]
> b
array([3, 3, 7, 8])

> b[2] = 100
b                          a
------------------  ----------------------------------------
[3,   3, 100,   8]      [  0,   1,  2,  3,   4,  5,   6,   7,   8,   9]
```

多维数组

NumPy的多维数组和一维数组类似。多维数组有多个轴。
我们前面已经提到从内到外分别是第0轴，第1轴...

```
> a = np.arange(0, 60, 10).reshape(-1, 1) + np.arange(0, 6)
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[0, 3:5]      a[4:, 4:]      a[2::2, ::2]
---------    ----------    --------------
[3, 4]         [[44, 45],     [[20, 22, 24],
                [54, 55]]      [40, 42, 44]]
```

#上面方法对于数组的切片都是共享原数组的储存空间的。

结构数组

C语言中可以通过struct关键字定义结构类型。NumPy中也有类似的**结构数组。**

```
> persontype = np.dtype({
    'names':['name', 'age', 'weight'],
    'formats':['S30','i', 'f']})

> a = np.array([("Zhang", 32, 75.5), ("Wang", 24, 65.2)],
      dtype=persontype)
```

	name	age	weight
0	zhang	32	75.5
1	wang	24	65.2

我们就创建了一个结构数组，并且可以通过索引得到每一行。

```
> print a[0]
('Zhang', 32, 75.5)
```

3. ufunc函数

ufunc函数

ufunc是universal function的简称，它是一种能对**数组每个元素进行运算**的函数。NumPy的许多ufunc函数都是用C语言实现的，因此它们的运算速度非常快。

```
> x = np.linspace(0, 2*np.pi, 10)
> y = np.sin(x)
> y
array([  0.00000000e+00,   6.42787610e-01,   9.84807753e-01,
. . . . . ,
       -2.44929360e-16])
```

值得注意的是，对于**同等长度**的ndarray，np.sin()比math.sin()快
但是对于**单个数值**，math.sin()的速度则更快。

四则运算

NumPy提供了许多ufunc函数，它们和相应的运算符运算结果相同。

```
> a = np.arange(0, 4)
> b = np.arange(1, 5)
> np.add(a, b)
array([1, 3, 5, 7])

> a+b
array([1, 3, 5, 7])

> np.subtract(a, b) # 减法
> np.multiply(a, b) # 乘法
> np.divide(a, b) # 如果两个数字都为整数，则为整数除法
> np.power(a, b) # 乘方
```

比较运算和布尔运算

使用==，>对两个数组进行比较，会返回一个**布尔数组**，**每一个元素都是对应元素的比较结果。**

```
> np.array([1, 2, 3]) < np.array([3, 2, 1])
array([ True, False, False], dtype=bool)
```

布尔运算在NumPy中也有对应的ufunc函数。

表达式	ufunc函数
y=x1==x2	equal(x1,x2[,y])
y=x1!=x2	not_equal(x1,x2[,y])
y=x1<x2	less(x1,x2[,y])
y=x1<=x2	not_equak(x1,x2[,y])
y=x1>x2	greater(x1,x2[,y])
y=x1>=x2	gerater_equal(x1,x2[,y])

自定义ufunc函数

NumPy提供的标准ufunc函数可以组合出复合的表达式，但是有些情况下，自己编写的则更为方便。我们可以**把自己编写的函数用frompyfunc()转化成ufunc函数。**

```
> def num_judge(x, a): #对于一个数字如果是3或5的倍数就
      if x%3 == 0:            返回0，否则返回a。
          r = 0
      elif x%5 == 0:
          r = 0
      else:
          r = a
      return r
> x = np.linspace(0, 10, 11)
> y = np.array([num_judge(t, 2) for t in x])#列表生成式
array([0, 2, 2, 0, 2, 0, 0, 2, 2, 0, 0])
```

自定义ufunc函数

使用frompyfunc()进行转化，调用格式如下：

```
frompyfunc(func, nin, nout)
```

func：计算函数
nin：func()输入参数的个数
nout：func()输出参数的个数

```
> numb_judge = np.frompyfunc(num_judge, 2, 1)

> y = numb_judge(x,2)
array([0, 2, 2, 0, 2, 0, 0, 2, 2, 0, 0], dtype=object)
```

因为最后输出的元素类型是object，所以我们还需要把它转换成整型。

```
y.astype(np.int)
```

广播(broadcasting)

我们看一下具体的例子：

```
> a = np.arange(0, 60, 10).reshape(-1, 1)
> b = np.arange(0, 5)
> c = a+b
c                          c.shape
----------------------     -------
[[ 0,  1,  2,  3,  4],          (6, 5)
 [10, 11, 12, 13, 14],
 [20, 21, 22, 23, 24],
 [30, 31, 32, 33, 34],
 [40, 41, 42, 43, 44],
 [50, 51, 52, 53, 54]]
```

广播(broadcasting)

ogrid用来生成广播运算所用的数组。

```
> x, y = np.ogrid[:5, :5]
x           y
-----  -----------------
[[0],     [[0, 1, 2, 3, 4]]
 [1],
 [2],
 [3],
 [4]]
```

下面操作和a.reshape(1,-1), a.reshape(-1,1)相同 。

```
> a = np.arange(4)
a[None, :]    a[:, None]
--------------  ----------
[[0, 1, 2, 3]]  [[0],[1],[2],3]]
```

4. NumPy的函数库

01 NumPy概述

02 NumPy数组(ndarry)对象

03 ufunc函数

04 NumPy的函数库

随机数

除了前面介绍的ufunc()函数之外，NumPy还提供了大量对于数组运算的函数。它们能够简化逻辑，提高运算速度。

我们首先看随机数。NumPy产生随机数的模块在**random里面，其中有大量的分布。**

```
> from numpy import random as nr
> np.set_printoptions(precision=2) #显示小数点后两位数

> r1 = nr.rand(4, 3)
[[ 0.87,  0.42,  0.34],
[ 0.25,  0.87,  0.42],
[ 0.49,  0.18,  0.44],
[ 0.53,  0.23,  0.81]]
```

```
> r2 = nr.poisson(2.0, (4, 3))
[[3, 1, 5],
[2, 2, 3],
[2, 4, 4],
[2, 2, 3]]
```

随机数

rand	0到1之间的随机数	normal	正态分布的随机数
randint	制定范围内的随机整数	uniform	均匀分布
randn	标准正态的随机数	poisson	泊松分布
choice	随机抽取样本	shuffle	随机打乱顺序

求和，平均值，方差

NumPy在**均值等方面常用的函数**如下：

函数名	功能
sum	求和
average	加权平均数
var	方差
mean	期望
std	标准差
product	连乘积

```
> np.random.seed(42)
> a = np.random.randint(0,10,size=(4,5))

> np.sum(a)
96
```

求和，平均值，方差


```
a                    np.sum(a, axis=1)   np.sum(a, axis=0)
--------------       -----------------   --------------------
[[6, 3, 7, 4, 6],    [26, 28, 24, 18]    [22, 13, 27, 18, 16]
 [9, 2, 6, 7, 4],
 [3, 7, 7, 2, 5],
 [4, 1, 7, 5, 1]]
```

keepdims可以保持原来数组的**维数。**

```
np.sum(a,1,keepdims=True)   np.sum(a,0,keepdims=True)
-------------------------   --------------------------
[[26],                          [[22, 13, 27, 18, 16]]
 [28],
 [24],
 [18]]
```

大小与排序

NumPy在**排序等方面常用的函数**如下：

函数名	功能	函数名	功能
min	最小值	max	最大值
ptp	极差	argmin	最小值的下标
mininum	二元最小值	maxinum	二元最大值
sort	数组排序	argsort	数组排序下标
percentile	分位数	median	中位数

min,max都有axis,out,keepdims等参数，我们来看其他函数。

```
> a = np.array([1, 3, 5, 7])
> b = np.array([2, 4, 6])
> np.maximum(a[None, :], b[:, None])#maxinum返回两组
array([[2, 3, 5, 7],                                矩阵广播计算后的
       [4, 4, 5, 7],                                结果
       [6, 6, 6, 7]])
```

大小与排序

sort()对数组进行排序会改变数组的内容，返回一个新的数组。**axis**的默认值都为-1，即按最终轴进行排序。axis=0对每列上的值进行排序。

```
np.sort(a)          np.sort(a, axis=0)
-----------------   ------------------
[[3, 4, 6, 6, 7],     [[3, 1, 6, 2, 1],
 [2, 4, 6, 7, 9],      [4, 2, 7, 4, 4],
 [2, 3, 5, 7, 7],      [6, 3, 7, 5, 5],
 [1, 1, 4, 5, 7]]      [9, 7, 7, 7, 6]]
```

percentile计算处于p%上的值。

```
> r = np.abs(np.random.randn(100000))
> np.percentile(r, [68.3, 95.4, 99.7])
array([ 1.00029686,  1.99473003,  2.9614485 ])
```

统计函数

NumPy中常用的统计函数有：**unique(), bicount(), histogram()**。我们来一个个介绍。首先看**unique()**:

```
> np.random.seed(42)
> a = np.random.randint(0, 8, 10)
> np.unique(a)
a                              np.unique(a)
-----------------------------  -----------------
[6, 3, 4, 6, 2, 7, 4, 4, 6, 1]         [1, 2, 3, 4, 6, 7]
```

unique有两个参数,**return_index=True**同时返回原始数组中的下标,**return_inverse=True**表示原始数据在新数组的下标

```
> x, index = np.unique(a, return_index=True)]
x                          index                  a[index]
-----------------  -----------------  -----------------
[1, 2, 3, 4, 6, 7]      [9, 4, 1, 2, 0, 5]     [1, 2, 3, 4, 6, 7]
```

统计函数


```
> x, rindex = np.unique(a, return_inverse=True)
 rindex                                  x[rindex]
----------------------------  -------------------------------
[4, 2, 3, 4, 1, 5, 3, 3, 4, 0]         [6, 3, 4, 6, 2, 7, 4, 4, 6, 1]
```

bincount()对**非负整数数组**中的各个元素出现的次数进行统计，返回数组中的第i个元素是整数i出现的次数。

```
> a = np.array([6, 3, 4, 6, 2, 7, 4, 4, 6, 1])
> np.bincount(a)
array([0, 1, 1, 1, 3, 0, 3, 1])

> x = np.array([0 ,  1,  2,  2,  1,  1,  0])
> w = np.array([0.1, 0.3, 0.2, 0.4, 0.5, 0.8, 1.2])
> np.bincount(x, w)
array([ 1.3,  1.6,  0.6])
```

统计函数

histogram()对以为数组进行直方图统计，其参数为：
histogram(a, bins=10, range=None, weights=None)
函数返回两个一维数组，**hist**是每个区间的统计结果，
bin_edges返回区间的边界值。

```
> a = np.random.rand(100)
> np.histogram(a, bins=5, range=(0, 1))
(array([28, 18, 17, 19, 18]),
 array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ]))

> np.histogram(a, bins=[0, 0.4, 0.8, 1.0])
 (array([46, 36, 18]), array([ 0. ,  0.4,  0.8,  1. ]))
```

操作多维数组

多维数组可以进行**连接，分段**等多种操作。我们先来看**vstack(),hstack(),column_stack()**函数。

```
> a = np.arange(3)
> b = np.arange(10, 13)

> v = np.vstack((a, b)) # 按第1轴连接数组
> h = np.hstack((a, b)) # 按第0轴连接数组
> c = np.column_stack((a, b)) # 按列连接多个一维数组

 v                  h                           c
--------------  ------------------------  ----------
[[ 0,  1,  2],     [ 0,  1,  2, 10, 11, 12]      [[ 0, 10],
 [10, 11, 12]]                                    [ 1, 11],
                                                  [ 2, 12]]
```

操作多维数组

split()函数进行分段。

```
> a = np.array([6, 3, 7, 4, 6, 9, 2, 6, 7, 4, 3, 7])
> b = np.array([ 1,  3,  6,  9, 10]
> np.split(a, idx) # 按元素位置进行分段
[array([6]),
array([3, 7]),
array([4, 6, 9]),
array([2, 6, 7]),
array([4]),
array([3, 7])]

> np.split(a, 2) # 按数组个数进行分段
[array([6, 3, 7, 4, 6, 9]),
array([2, 6, 7, 4, 3, 7])]
```

多项式函数

多项式函数是整数的次幂与系数的乘积，如：
f(x)= an(x^n)+ an-1(x^(n-1))+...+ a1(x)+ a0
NumPy中多项式函数可以用**一维数组**表示。a[0]为最高次，a[-1]为常数项。

```
> a = np.array([1.0, 0, -2, 1])
> p = np.poly1d(a)
> print type(p)
<class 'numpy.lib.polynomial.poly1d'>

> p(np.linspace(0, 1, 5))
 array([ 1.      ,  0.515625,  0.125   , -0.078125,  0.      ])
```

多项式函数可以进行**四则运算**，其中**运算的列表自动化成多项式函数。**

多项式函数


```
> p + [-2, 1]
poly1d([ 1.,  0., -4.,  2.])

> p * p
poly1d([ 1.,  0., -4.,  2.,  4., -4.,  1.])

> p / [1, 1] # 分别为商和余
(poly1d([ 1., -1., -1.]), poly1d([ 2.]))
```

多项式也可以进行**积分和求导。**

```
> p.deriv()
poly1d([ 3.,  0., -2.])

> p.integ()
poly1d([ 0.25,  0.  , -1.  ,  1.  ,  0.  ])
```

多项式函数

Roots可以求多项式的根。

```
> r = np.roots(p)
> r
array([-1.61803399,  1.        ,  0.61803399])
```

polyfit()可以对数据进行多项式拟合。x, y为数据点，deg为多项式最高阶数。

```
> a = np.polyfit(x , y, deg)
```

poly()返回多项式系数构成的数组。

```
> a = np.poly(x )
```

Python模块-pandas

- pandas

pandas 是基于NumPy的一种工具，该工具是为了解决数据分析任务而创建的。

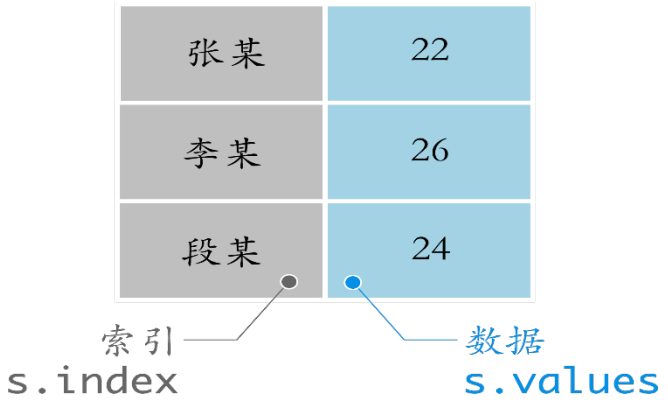
pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。pandas提供了大量能使我们快速便捷地处理数据的函数和方法。你很快就会发现，它是使Python成为强大而高效的数据分析环境的重要因素之一。

Python模块-pandas

●基本数据结构

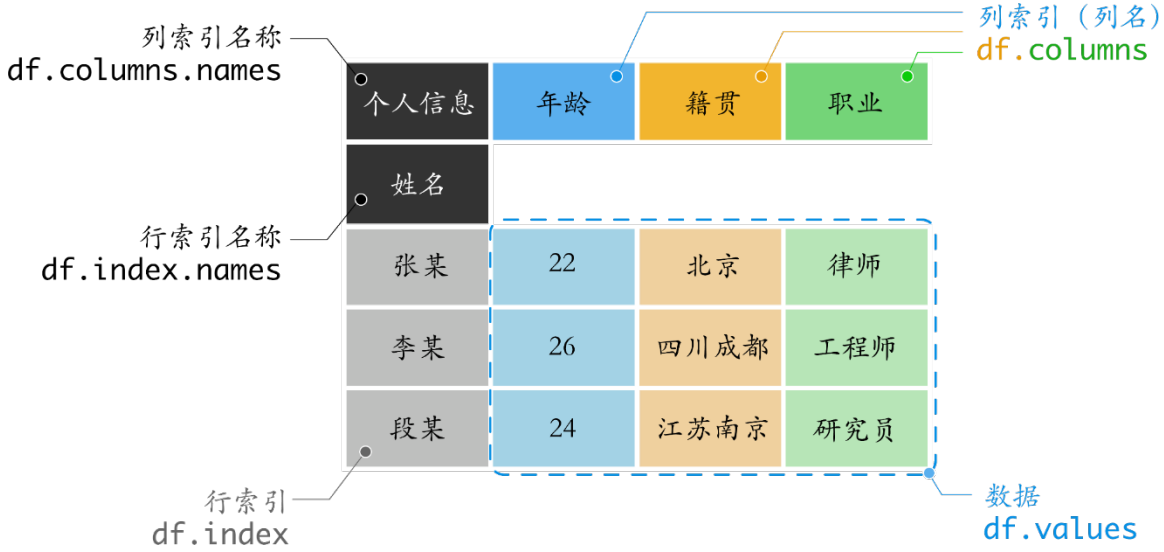
Series

一维数据结构，包含行索引和数据两个部分.



DataFrame

二维数据结构，包含带索引的多列数据，各列的数据类型可能不同.



Python模块-pandas

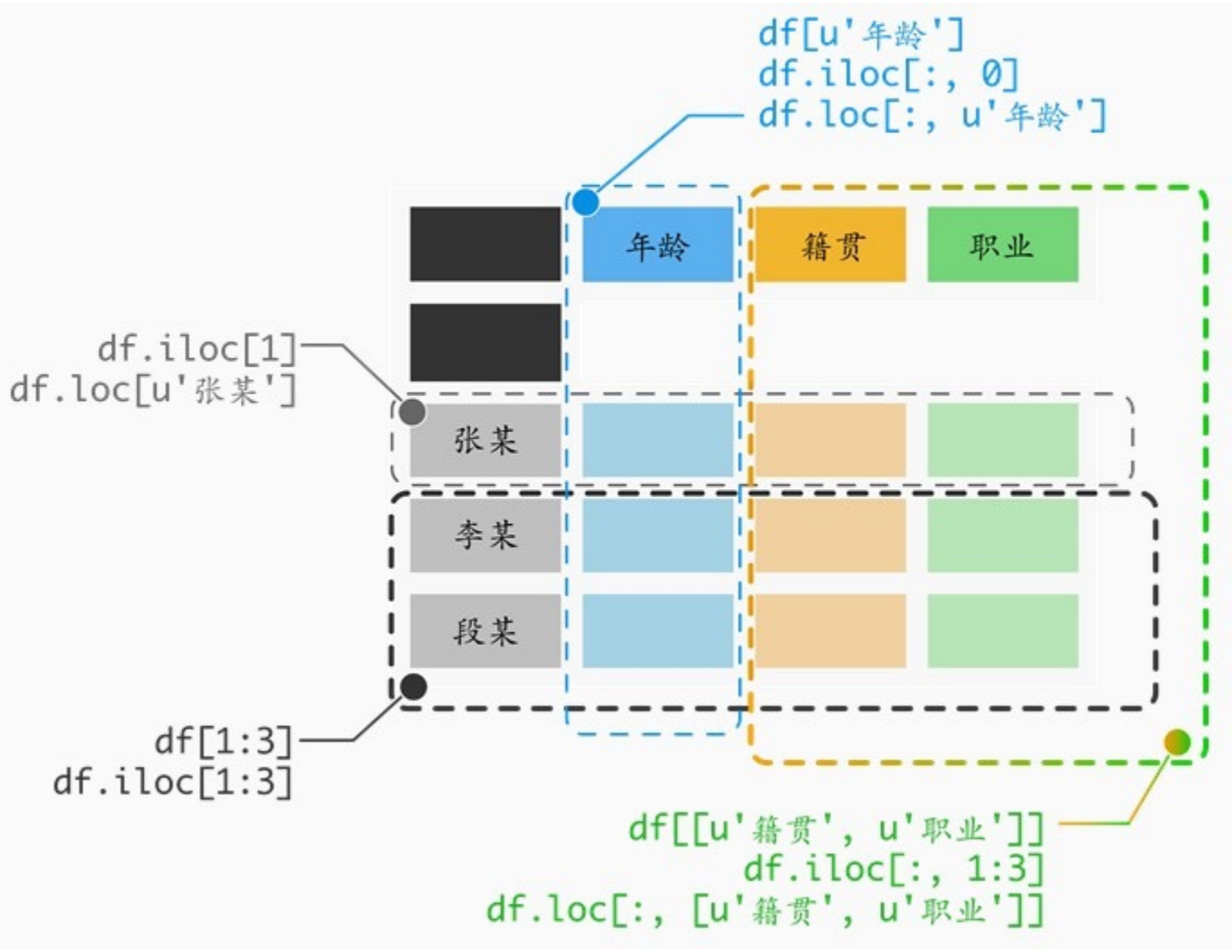
●数据索引

df[5:10]
通过切片方式选取多行.

df[col_label] or df.col_label
选取列.

df.loc[row_label, col_label]
通过标签选取行/列.

df.iloc[row_loc, col_loc]
通过位置（自然数）选取行/列
.



Python模块-pandas

●数据合并

`pd.merge(left, right)` 类数据库的数据融合操作.

参数：`how`，融合方式，包括左连接、右连接、内连接（默认）和外连接；`on`，连 接键；`left_on`，左键；`right_on`，右键；`left_index`，是否将left行索引作 为左键；`right_index`，是否将right行索引作为右键.

	姓名	年龄
0	张某	22
1	李某	26
2	段某	24

left

	姓名	籍贯
7	张某	北京
8	李某	四川成都
9	钱某	江苏南京

right

inner

	姓名	年龄	籍贯
0	张某	22	北京
1	李某	26	四川成都

```
pd.merge(left, right,
         how='inner', on=u'姓名')
```

outer

	姓名	年龄	籍贯
0	张某	22.0	北京
1	李某	26.0	四川成都
2	段某	24.0	NaN
3	钱某	NaN	江苏南京

```
pd.merge(left, right,
         how='outer', on=u'姓名')
```

left

	姓名	年龄	籍贯
0	张某	22	北京
1	李某	26	四川成都
2	段某	24	NaN

```
pd.merge(left, right,
         how='left', on=u'姓名')
```

right

	姓名	年龄	籍贯
0	张某	22.0	北京
1	李某	26.0	四川成都
2	钱某	NaN	江苏南京

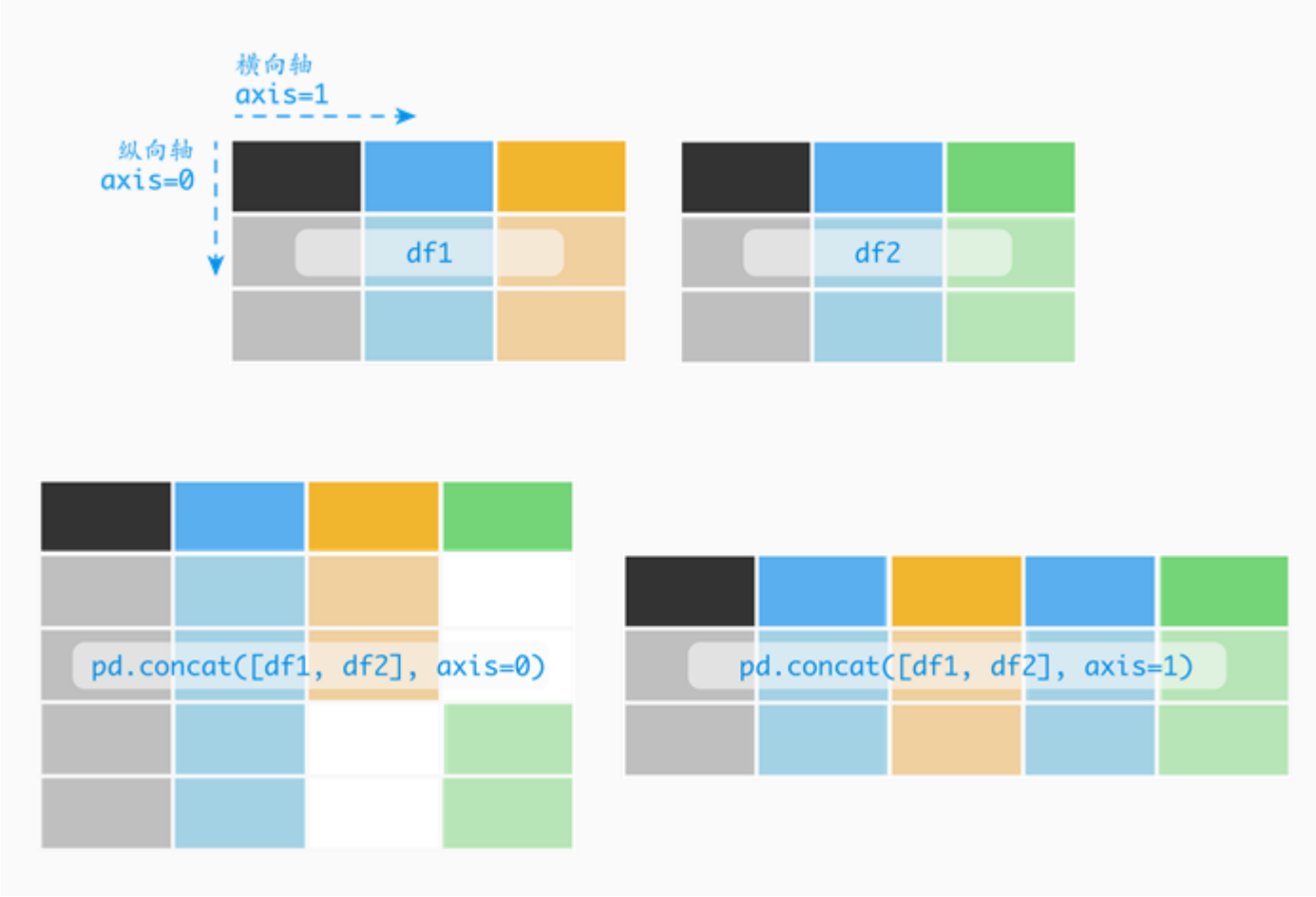
```
pd.merge(left, right,
         how='right', on=u'姓名')
```

Python模块-pandas

●数据融合

pd.concat([df1, df2])

轴向连接多个DataFrame.



Python模块-pandas

文件读写

从文件中读取数据（DataFrame）

pd.read_csv() | 从CSV文件读取.
pd.read_table() | 从制表符分隔文件读取，如TSV.
pd.read_excel() | 从 Excel 文 件 读 取 .
pd.read_sql() | 从 SQL 表 或 数 据 库 读 取 .
pd.read_json() | 从JSON格式的URL或文件读取.
pd.read_clipboard() | 从剪切板读取.

将DataFrame写入文件

df.to_csv() | 写 入 CSV 文 件 .
df.to_excel() | 写 入 Excel 文 件 .
df.to_sql() | 写入SQL表或数据库.
df.to_json() | 写入JSON格式的文件.
df.to_clipboard() | 写入剪切板.

Python模块-Scipy

●Scipy

scipy是构建在numpy的基础之上的，它提供了许多的操作numpy的数组的函数。

SciPy是一款方便、易于使用、专为科学和工程设计的python工具包，它包括了统计、优化、整合以及线性代数模块、傅里叶变换、信号和图像图例，常微分方差的求解等

scipy.cluster	向量量化
scipy.constants	数学常量
scipy.fftpack	快速傅里叶变换
scipy.integrate	积分
scipy.interpolate	插值
scipy.io	数据输入输出
scipy.linalg	线性代数
scipy.ndimage	N维图像
scipy.odr	正交距离回归
scipy.optimize	优化算法
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	特殊数学函数
scipy.stats	统计函数

SciPy

SciPy 建立在 NumPy 基础之上,集成了众多的数学、科学以及工程计算中常用库函数的 Python 模块,例如线性代数、常微分方程数值求解、信号处理、图像处理、稀疏矩阵等。通过给用户提供一些高层的命令和类,SciPy 在 Python 交互式会话中大大增加了操作和可视化数据的能力。通过 SciPy，Python 的交互式会话变成了一个数据处理和 system-prototyping 的环境,足以和 MATLAB 、IDL 、 Octave 、 R-Lab 以及 SciLab 抗衡。 例如,用 optimize 实现最优化:

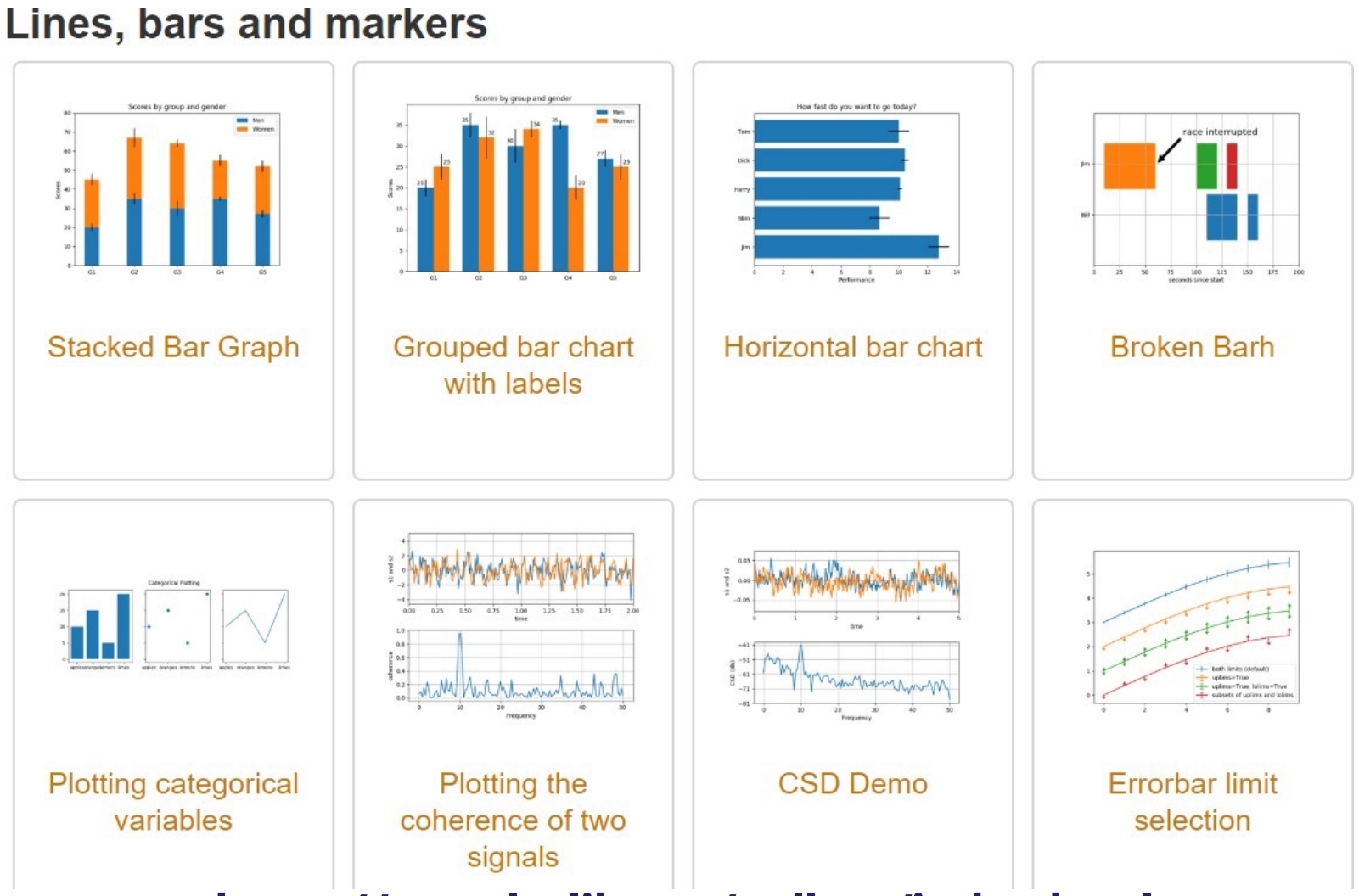
```
In[42]:
fromscipyimport*
importmatplotlib.pyplotasplt
importnumpyasnp
fromscipyimportoptimize
# 最优化问题（寻找函数的最大值或者最小值）是数学中的一大领域，复杂函数的最优化问题
# 或者多变量的最优化问题，可能会非常复杂
x=linspace(-5,3,100)
deff(x):
    return4*x**3+(x-2)**2+ x**4
# 局部最小值
x _ min _ local=optimize.fmin _ bfgs(f,2)
print(x _ min _ local)
# 全局最小值
x _ max _ global=optimize.fminbound(f,-10,10)
print(x _ max _ global)
```

Python模块-matplotlib

- **Matplotlib**

Matplotlib 是一个 Python 的 2D绘图库，它以各种硬拷贝格式和跨平台的交互式环境生成出版质量级别的图形 。

通过 Matplotlib，开发者可以仅需要几行代码，便可以生成绘图，直方图，功率谱，条形图，错误图，散点图等。



https://matplotlib.org/gallery/index.html

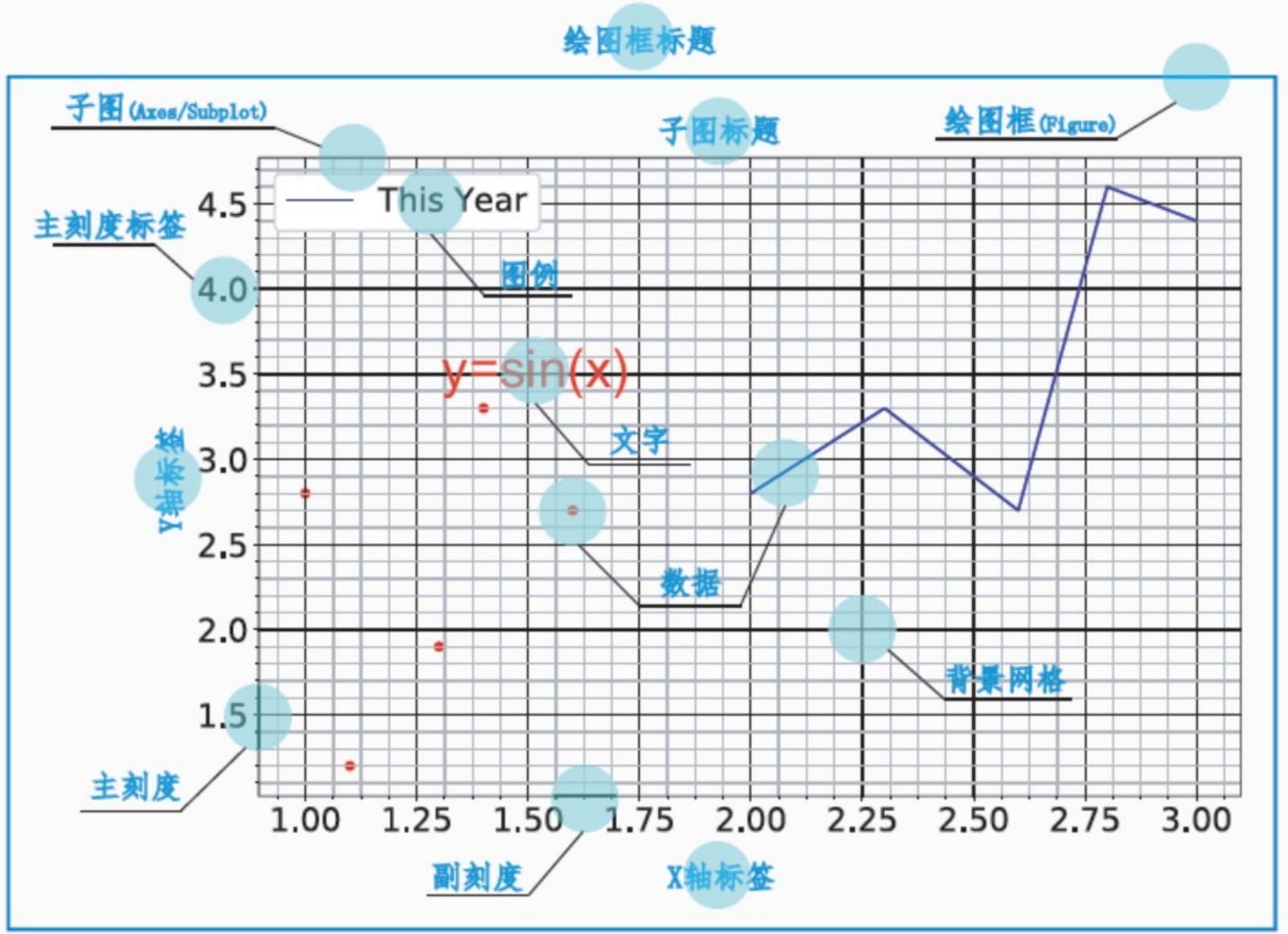
Python模块-matplotlib

图形的各元素名称如下：

绘图框 是图形的最高容器，所有图形必须放置在绘图框中.

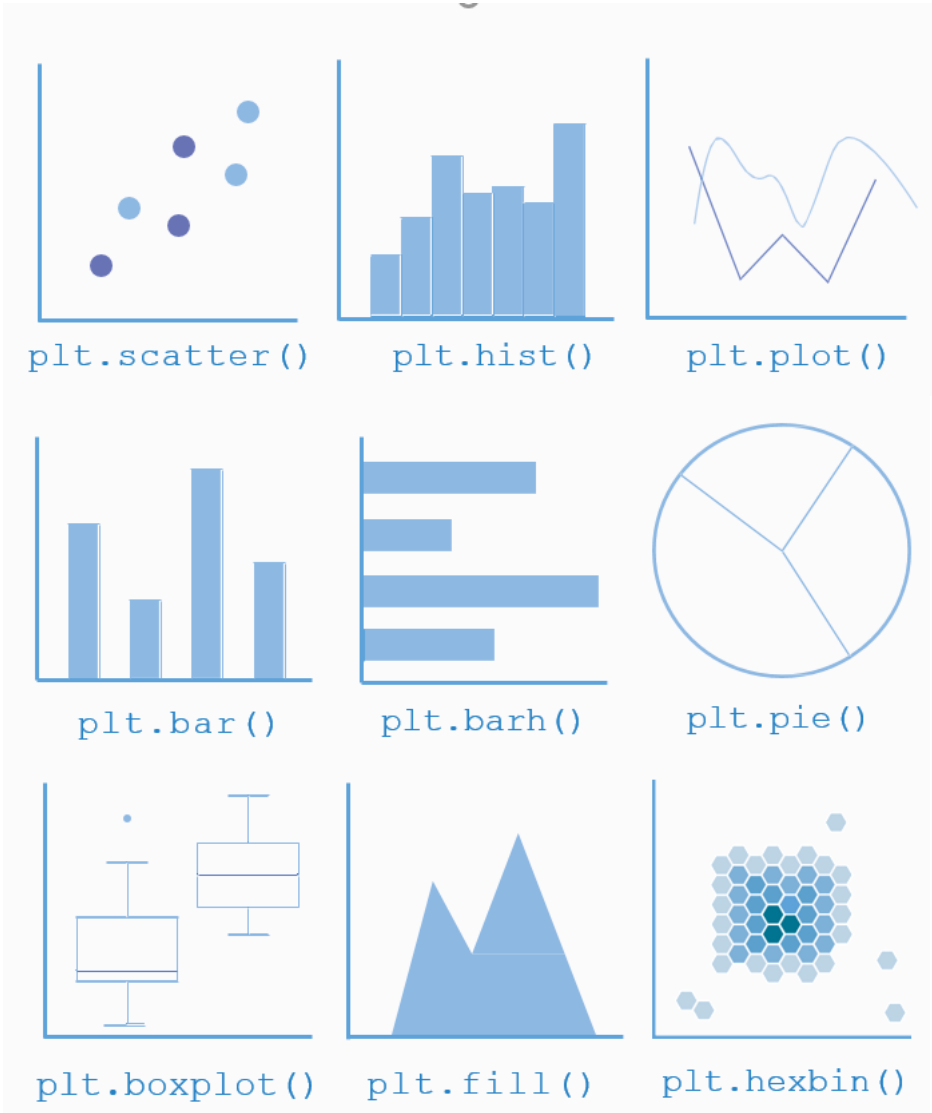
子图 是绘图框中所包含的图形，即便绘图框只包含一幅图，也称之为子图.

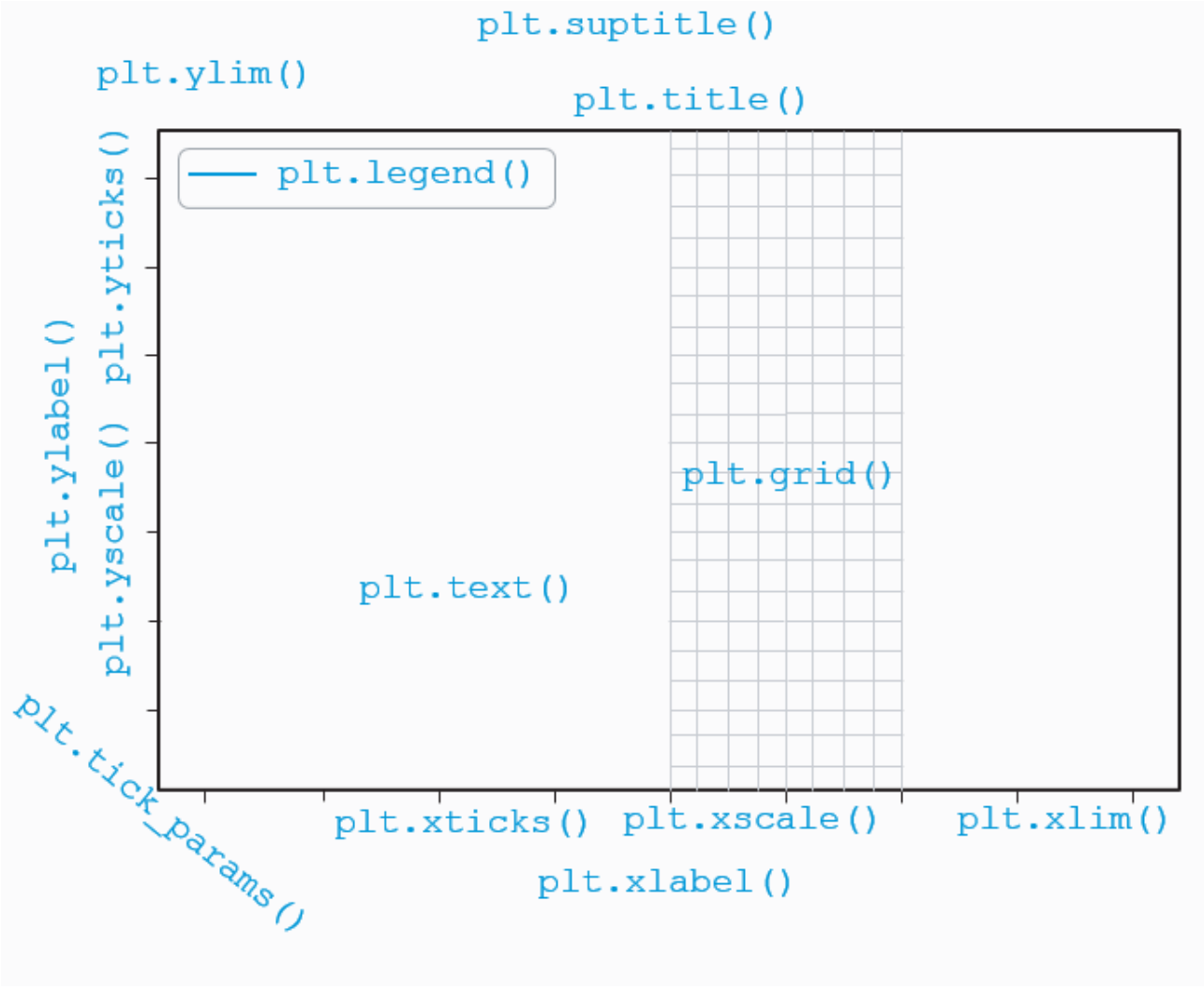
元素 是组成子图的部件，从子图最内部的数据线条到外围的坐标轴标签等都属于元素.



Python模块-matplotlib

图形样式





参考文献

1. NumPy 官网 http://www.numpy.org/
2. NumPy 源代码：https://github.com/numpy/numpy