

《交通大数据技术》



2024/6/14

交通大数据技术

马晓磊

交通科学与工程学院

2023年xx月xx日

Numpy基础



1. **NumPy概述**
2. **NumPy数组(ndarray)对象**
3. **ufunc函数**
4. **NumPy的函数库**

Numpy简介

NumPy是一个开源的Python库，主要用在数据分析和科学计算，基本上可以把NumPy看做是Python数据计算的基础，因为很多非常优秀的数据分析和机器学习框架底层使用的都是NumPy。比如：Pandas, SciPy, Matplotlib, scikit-learn, scikit-image 等。

NumPy库主要包含多维数组和矩阵数据结构。它为ndarray（一个n维数组对象）提供了对其进行有效操作的方法。NumPy可以用于对数组执行各种数学运算，并且提供了可在这些数组和矩阵上运行的庞大的高级数学函数库。

pip install numpy

```
sofiya@sofiya-VirtualBox:~$ pip install numpy
Collecting numpy
  Downloading https://files.pythonhosted.org/packages/3a/5f/47e578b3ae79e2624e2
05445ab77a1848acdaa2929a00eeef6b16eaaeb20/numpy-1.16.6-cp27-cp27mu-manylinux1_x
86_64.whl (17.0MB)
    100% |#####| 17.0MB 34kB/s
Installing collected packages: numpy
Successfully installed numpy-1.16.6
```



Array和List

Python中有一个数据类型叫做List，list中可以存储不同种类的对象。在应用程序中这样做没有什么问题，但是如果是在科学计算中，我们希望**一个数组中的元素类型必须是一致的**，所以有了NumPy中的Array。

NumPy可以快速的创建Array，并且对其中的数据进行操作。

NumPy中的Array要比Python中的List要快得多，并且占用更少的内存空间。

看下两者之间的性能差异：

```
In [1]: import numpy as np
...: my_arr = np.arange(1000000)
...: my_list = list(range(1000000))
...: %time for _ in range(10): my_arr2 = my_arr * 2
...: %time for _ in range(10): my_list2 = [x * 2 for x in my_list]
...:
CPU times: user 12.3 ms, sys: 7.88 ms, total: 20.2 ms
Wall time: 21.4 ms
CPU times: user 580 ms, sys: 172 ms, total: 752 ms
Wall time: 780 ms
```

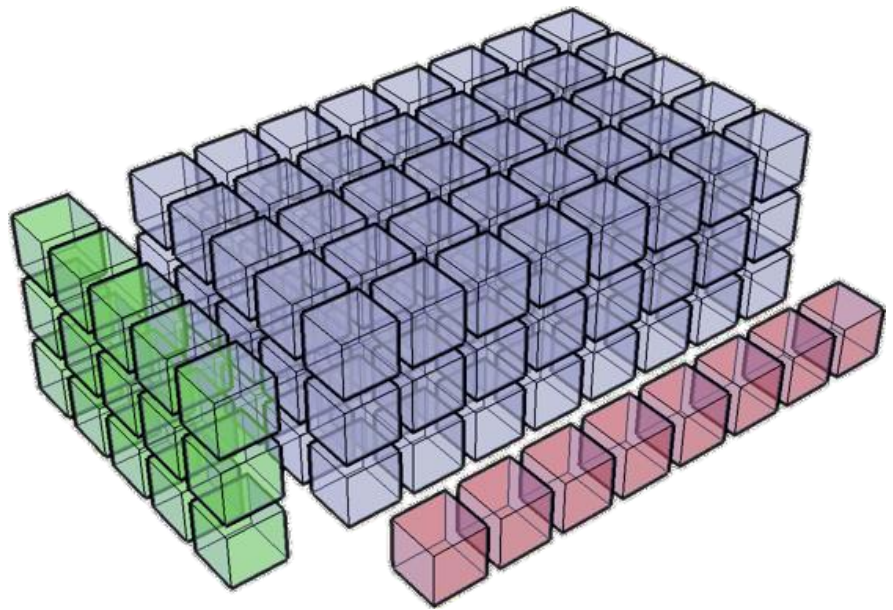
%time测试时间

上面的例子对一个包含一百万的数据进行乘2操作，可以看到，使用NumPy的效率是Python的几十倍，如果在大型数据项目中这个效率会造成非常大的性能影响。

NumPy是什么?

NumPy(Numeric Python)是Python的一种开源的数值计算扩展库。它包含很多功能：

- 创建n维数组（矩阵）
- 对数组进行函数运算
- 数值积分
- 线性代数运算
- 傅里叶变换
- 随机数产生
-



NumPy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/>



NumPy的安装

Anaconda里面已经安装过NumPy。

原生的Python安装：

- 在cmd中输入

```
pip install numpy
```

安装之后，我们用**导入**这个库

```
import numpy as np
```


Numpy 目录

1. NumPy概述
2. NumPy数组(ndarray)对象
3. ufunc函数
4. NumPy的函数库

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]])
```

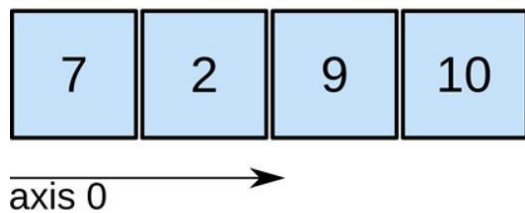
ndarray对维数没有限制。

[]从内到外分别为第0轴，第1轴，第2轴，第3轴。

认识 NumPy 数组对象

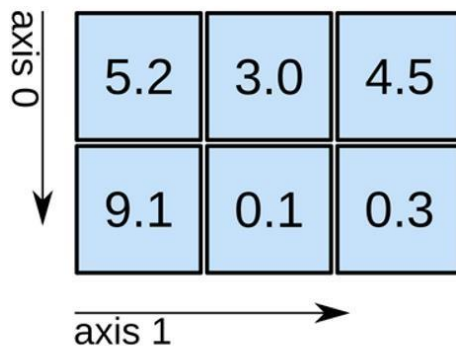
NumPy 数组图示

1D array



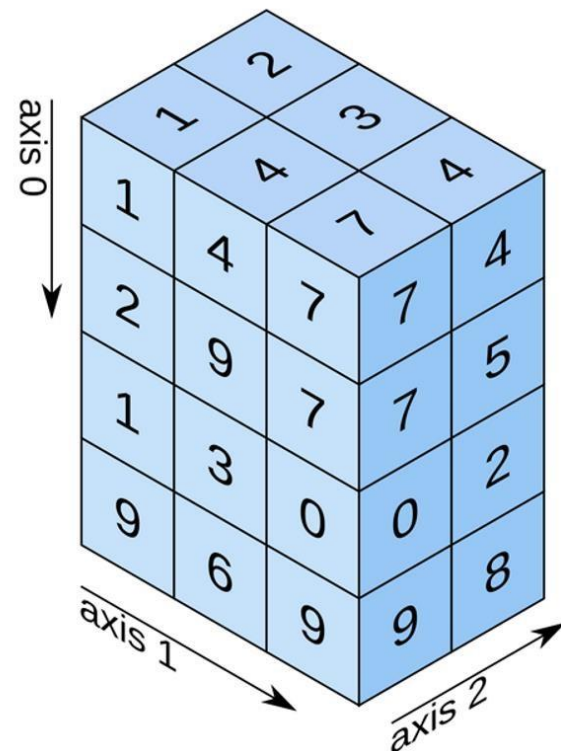
shape(4,)

2D array



shape(3,2)

3D array



shape(4,3,2)

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')
```

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.]])
```

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]])
```

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

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

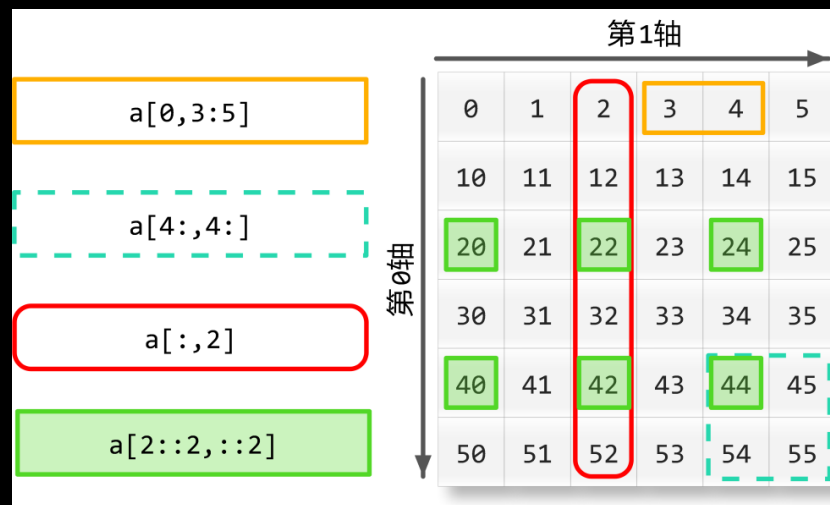
```
> a[2:4] = 100, 101
> a
array([ 0,  1, 100, 101,  4,  5,  6,  7,  8,  9])
```

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]])
```

<code>a[0, 3:5]</code>	<code>a[4:, 4:]</code>	<code>a[2::2, ::2]</code>
-----	-----	-----
[3, 4]	[[44, 45], [54, 55]]	[[20, 22, 24], [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。

多维数组

如果我们想**创立原数组的副本**，我们可以用**整数元组**，**列表**，**整数数组**，**布尔数组**进行切片。

```
>>> 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]])
```

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[(0,1,2,3,4),(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])
```

第 0 轴	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
	第 1 轴					

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)
```

Numpy 目录

1. NumPy概述
2. NumPy数组(ndarray)对象
3. **ufunc函数**
4. NumPy的函数库

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
<code>y=x1==x2</code>	<code>equal(x1,x2[,y])</code>
<code>y=x1!=x2</code>	<code>not_equal(x1,x2[,y])</code>
<code>y=x1<x2</code>	<code>less(x1,x2[,y])</code>
<code>y=x1<=x2</code>	<code>not_equal(x1,x2[,y])</code>
<code>y=x1>x2</code>	<code>greater(x1,x2[,y])</code>
<code>y=x1>=x2</code>	<code>gerater_equal(x1,x2[,y])</code>

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)
```

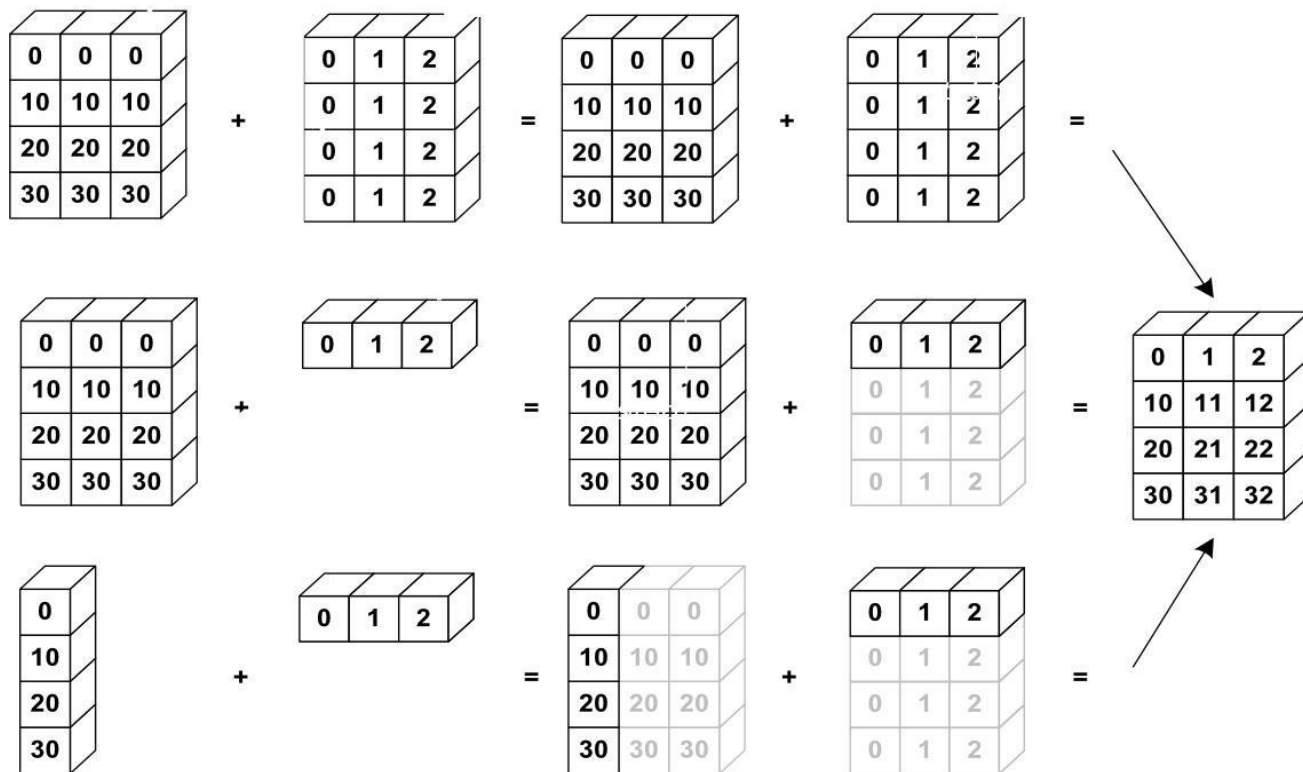
因为最后输出的元素类型是object，所以我们还需要把它转换成整型。

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

广播(broadcasting)

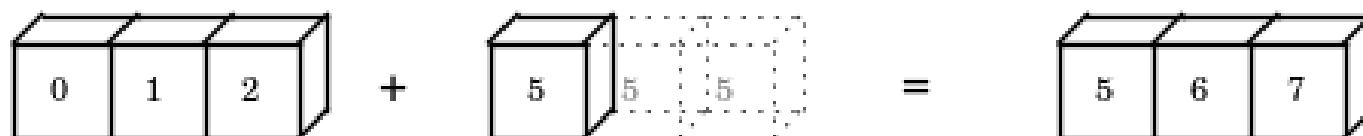
使用ufunc对两个数组进行运算时，ufunc函数会对两个数组的对应元素进行运算。如果数组的形状不相同，就会进行**下广播**处理。

简而言之，就是向**两个数组每一维度上的最大值靠齐**。

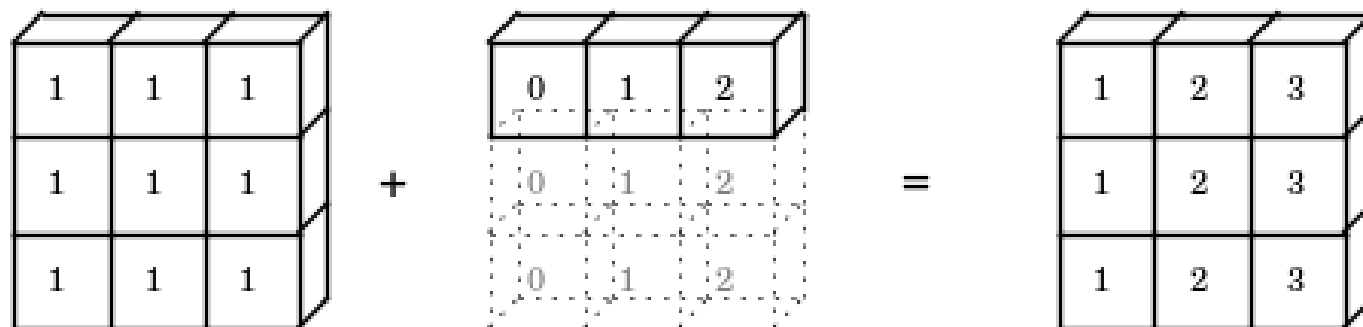


广播(broadcasting)

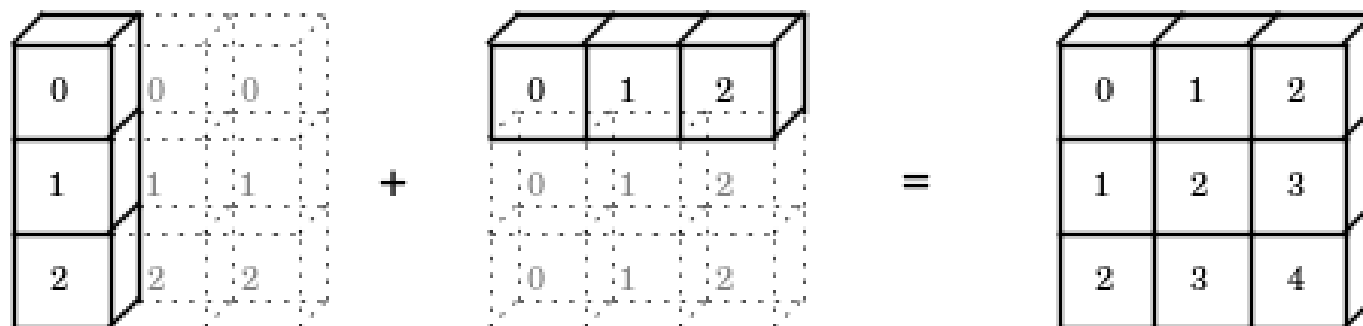
`np.arange(3) + 5`



`np.ones((3, 3)) + np.arange(3)`



`np.arange(3).reshape((3, 1)) + np.arange(3)`



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]]
```

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]]

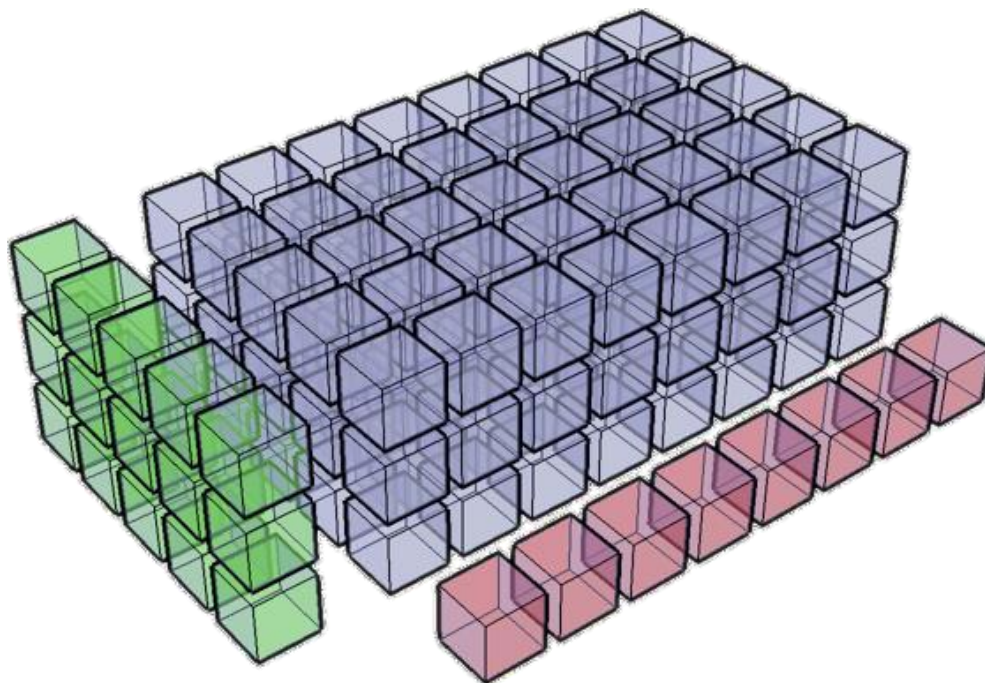
> r2 = nr.poisson(2.0, (4, 3))
[[3, 1, 5],
 [2, 2, 3],
 [2, 4, 4],
 [2, 2, 3]]
```


Numpy 目录

1. NumPy概述
2. NumPy数组(ndarray)对象
3. ufunc函数
4. NumPy的函数库

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
minimum	二元最小值	maximum	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])#maximum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], [10, 11, 12]]	[0, 1, 2, 10, 11, 12]	[[0, 10], [1, 11], [2, 12]]

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：

$$f(x) = a_n(x^n) + a_{n-1}(x^{n-1}) + \dots + a_1(x) + a_0$$

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。

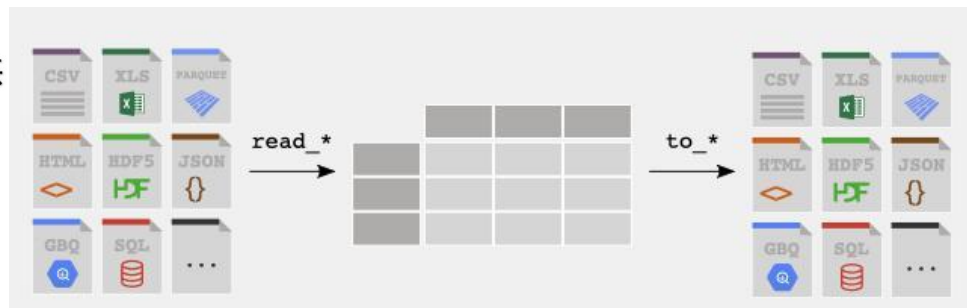
```
> a = np.poly(x )
```

Pandas基础



Pandas基础

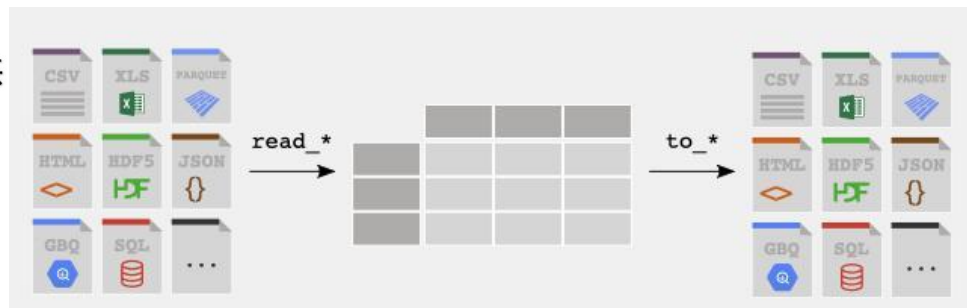
- 开放源码、BSD许可的库，提供高性能、易于使用的数据结构和数据分析工具
- 名字来自术语面板数据(Panel Data)和Python数据分析(Python Data Analysis)
- 是一个强大的结构化数据分析工具集，由Numpy提供高性能的矩阵运算
- 可以从各种文件格式比如csv, json, excel导入数据，也可以从SQL Server, Oracle, My SQL等数据库导入数据
- 可以对各种数据进行运算操作，比如归并、重新组织、索引选择、数据清洗和加工等
- 广泛应用于学术、金融、统计学等各个数据分析领域



*代表文件格式

Pandas基础

- 开放源码、BSD许可的库，提供高性能、易于使用的数据结构和数据分析工具
- 名字来自术语面板数据(Panel Data)和Python数据分析(Python Data Analysis)
- 是一个强大的结构化数据分析工具集，由Numpy提供高性能的矩阵运算
- 可以从各种文件格式比如csv, json, excel导入数据，也可以从SQL Server, Oracle, My SQL等数据库导入数据
- 可以对各种数据进行运算操作，比如归并、重新组织、索引选择、数据清洗和加工等
- 广泛应用于学术、金融、统计学等各个数据分析领域

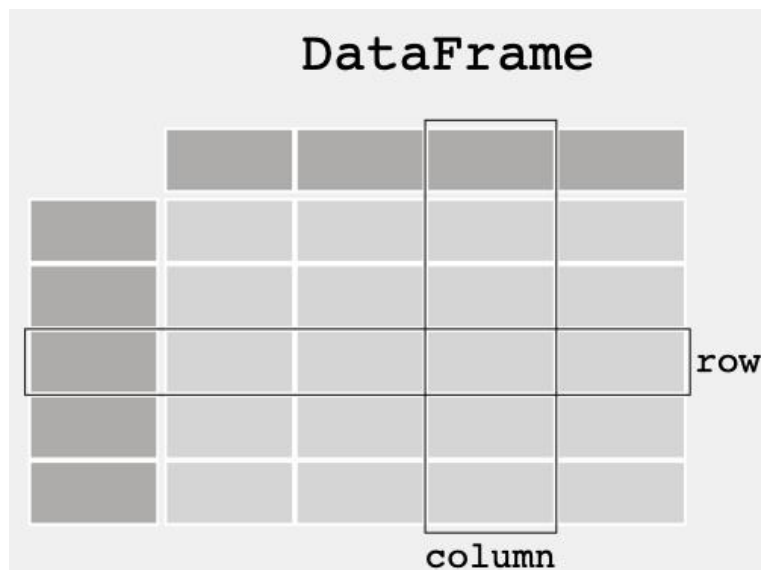


*代表文件格式

Pandas基础

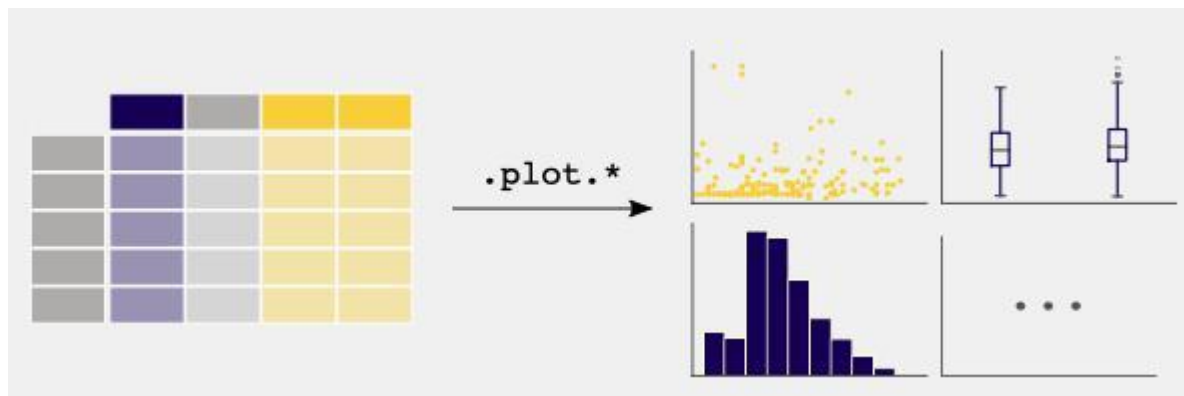
- Pandas主要用于处理二维表格式数据
 - DataFrame是pandas处理的数据对象
 - 包括
 - 行
 - 列
 - 列名
 - 索引
 - 可以把DataFrame想象成一个虚拟的Excel表
- Pandas包的导入

```
import pandas as pd
```

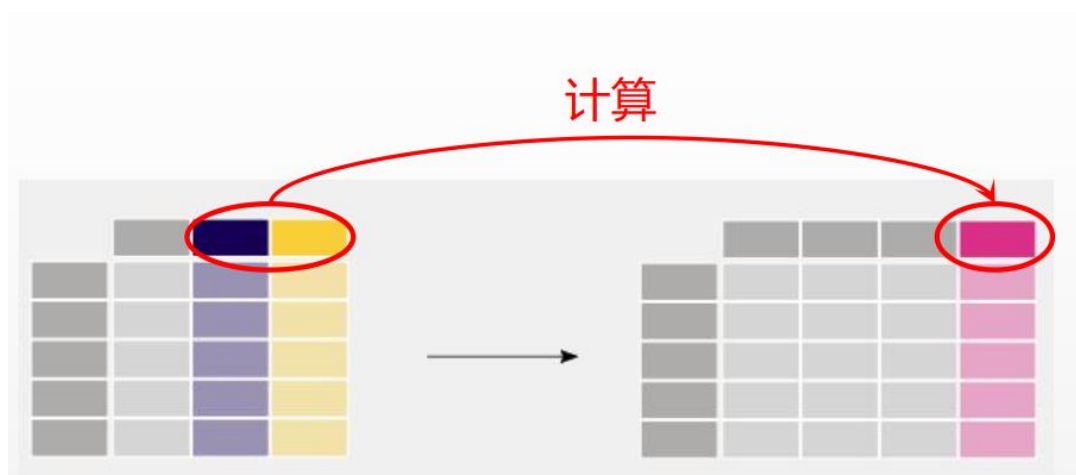


Pandas基础

- Pandas通过与Matplotlib/seaborn集成，可以很容易实现数据的可视化



- 通过多个列数据计算新的列



- 汇总统计分析

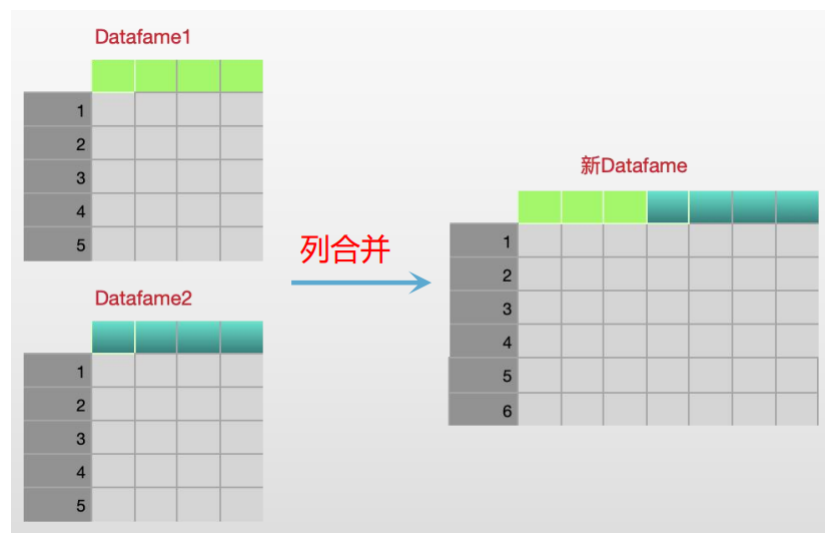
- mean
- median
- min
- max
- counts



- 可以在原始索引上计算或者通过不同条件进行聚合后进行计算

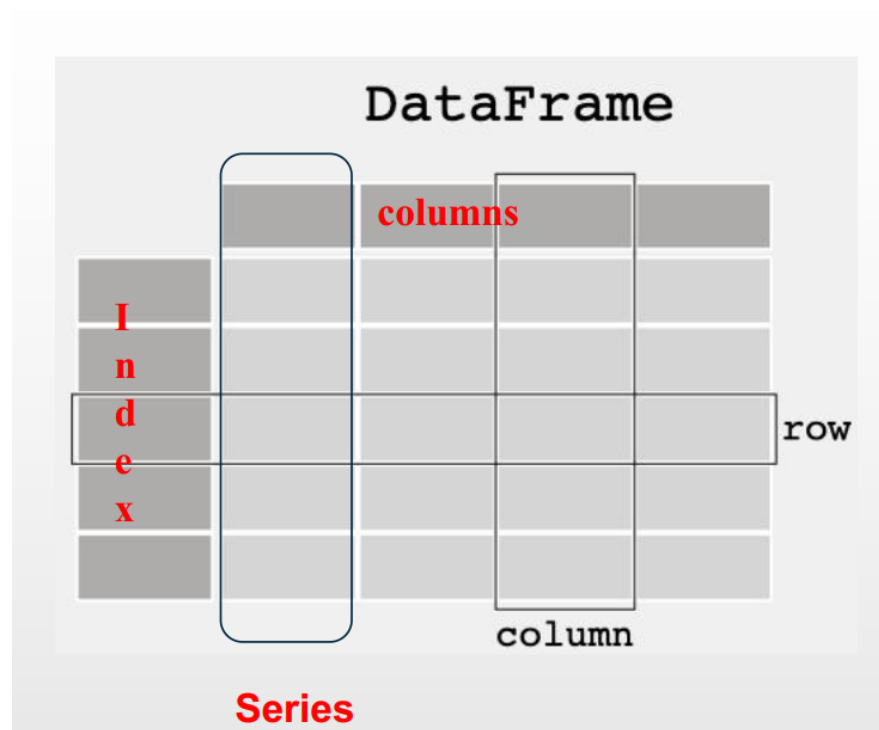
Pandas基础

- 实现多个数据表的合并
 - 行合并
 - 列合并




Pandas基础


- 定义了一种二维数据结构
- 用于存储不同类型的数据如字符串、整数、浮点数、类别数据、布尔数据乃至复杂数据
- DataFrame包括
 - 行
 - 列
 - 索引
- 类似于Excel的表单或关系数据库的表





Pandas基础


pandas一维数据Series

`df = pd.Series(data=None,`  一维数据结构，如list, dict, tuple, array等

`index=None,`  指定数据索引，默认为从0~n-1

`dtype=None,`  数据类型

`name=None,`  数据系列的名称

`copy=False)`  是否产生原始数据的新的副本

Pandas基础

```
A=np.random.randint(60,100,size=20)
d=pd.Series(A,name='Score',copy=False)
```

```
A[0]=-20
d.head(2)
```

```
0    -20
1     94
Name: Score, dtype: int64
```

```
d[0]=-30
A[:2]
```

```
array([-30,  94])
```

```
A=np.random.randint(60,100,size=20)
d=pd.Series(A,name='Score',copy=True)
```

```
A[0]=-20
d.head(2)
```

```
0     67
1     75
Name: Score, dtype: int64
```

```
d[0]=-30
A[:2]
```

```
array([-20,  75])
```


- 学生多学科成绩的序列
 - 以学生姓名作为索引
 - 多个序列共享同一个索引

```
A1=np.random.randint(60,100,size=20)
A2=np.random.randint(60,100,size=20)
idx=['Name'+str(i) for i in range(20)]
d1=pd.Series(A1,name='Math',dtype=np.float32,index=idx)
d2=pd.Series(A2,name='Chinese',dtype=np.float32,index=idx)
d1.head(3),d2.head(3)
```

```
(Name0    64.0
 Name1    82.0
 Name2    67.0
 Name: Math, dtype: float32,
 Name0    85.0
 Name1    79.0
 Name2    87.0
 Name: Chinese, dtype: float32)
```

- pandas序列切片与Numpy数组类似，通过位置切片
- 如果序列指定了索引，还可以通过索引切片

```
idx=['Name'+str(i) for i in range(20)]  
d1=np.random.randint(60,100,size=20)  
s1=pd.Series(d1,name='Math',index=idx)  
s1.head()
```

```
Name0    76  
Name1    72  
Name2    75  
Name3    95  
Name4    92  
Name: Math, dtype: int64
```

```
s1[:10]  
s1[:-2]  
s1[3:6]  
s1['Name5':'Name8']
```

```
Name5    85  
Name6    80  
Name7    63  
Name8    80  
Name: Math, dtype: int64
```

- 序列的切片操作与Numpy数组类似，同时也可以通过索引进行数据检索
- `s.head(n)` #显示序列是的前n行数据
- `s.tail(n)` #显示后n行数据
- `s.unique` #获取不重复的数据列表
- `s1+s2` #两个序列相加

```
d1[:10]
d1[3]
d1[3:5]
d1['Name0':'Name5']
```

```
Name0    87.0
Name1    74.0
Name2    89.0
Name3    63.0
Name4    99.0
Name5    86.0
Name: Math, dtype: float32
```

```
d1.head(5)
d1.tail(5)
d1.unique()
```

```
array([87., 74., 89., 63., 99., 86., 66., 67., 61., 64., 71., 97., 88.,
       91., 85., 68., 98.], dtype=float32)
```

`d1+d2[:-2]`

```
Name0    169.0
Name1    144.0
Name10    146.0
Name11    182.0
Name12    169.0
Name13    160.0
Name14    152.0
Name15    193.0
Name16    175.0
Name17    165.0
Name18      NaN
Name19      NaN
Name2     164.0
Name3     124.0
Name4     164.0
Name5     179.0
Name6     139.0
Name7     160.0
Name8     158.0
Name9     131.0
dtype: float32
```

Pandas基础

- `s.isnull()` #空值则返回True , 否则返回False
- `s.notnull()` #返回非空值

```
d.isnull()
```

```
Name0      False
Name1      False
Name10     False
Name11     False
Name12     False
Name13     False
Name14     False
Name15     False
Name16     False
Name17     False
Name18      True
Name19      True
Name2      False
Name3      False
Name4      False
Name5      False
Name6      False
Name7      False
Name8      False
Name9      False
dtype: bool
```

▼ #返回非空值

```
d[d.notnull()]
```

#返回空值

```
d[d.isnull()]
```

```
Name18      NaN
Name19      NaN
dtype: float32
```

- 通过字典创建
- 通过二维数组创建

DataFrame创建

```
d1=np.random.randint(60,100,size=20)
d2=np.random.randint(60,100,size=20)
d3=np.random.randint(60,100,size=20)
idx=['Name'+str(i) for i in range(20)]
df1=pd.DataFrame({'Math':d1,'Chinese':d2,'English':d3},index=idx)
d=np.column_stack((d1,d2,d3))
df2=pd.DataFrame(d,columns=['Math','Chinese','English'],index=idx)
df1.head(2),df2.head(2)
```

```
(
   Math  Chinese  English
Name0    72      72      67
Name1    85      94      67,
   Math  Chinese  English
Name0    72      72      67
Name1    85      94      67)
```

column_stack 按列堆叠

Pandas基础

- 用pandas 读取数据

```
In [ ]: #Read csv file  
df = pd.read_csv("https://github.com/Apress/data-analysis-and-visualization-  
using-python/blob/master/Ch07/Salaries.csv")
```

有许多 pandas 命令可以读取其他数据格式:

```
pd.read_excel('myfile.xlsx', sheet_name='Sheet1', index_col=None,  
na_values=['NA'])  
pd.read_stata('myfile.dta')  
pd.read_sas('myfile.sas7bdat')  
pd.read_hdf('myfile.h5', 'df')
```

数据框架

- 举例来看

```
In [3]: #List first 5 records  
df.head()
```

```
Out[3]:
```

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

数据类型

Pandas 数据类型	原生 Python 类型	描述
object	string	最通用的 dtype。如果列具有混合类型（数字和字符串），则将分配给您的列
int64	int	数字字符。64 是指分配给保存此字符的内存
float64	float	带小数的数字字符。如果一列包含数字和 NaN，则 pandas 将默认为 float64，以防缺失值为小数
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	用于保存时间数据的值

数据类型

```
In [4]: #Check a particular column type  
df['salary'].dtype
```

```
Out[4]: dtype('int64')
```

```
In [5]: #Check types for all the columns  
df.dtypes
```

```
Out[4]: rank          object  
         discipline   object  
         phd          int64  
         service      int64  
         sex          object  
         salary       int64  
         dtype: object
```

Data Frame属性

df.attribute	描述
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Data Frame方法

与属性不同，Data Frame方法有括号。所有属性和方法都可以用`dir(df)`函数列出：

df.method()	描述
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

groupby方法

使用“group by”方法，我们可以：

- 根据某些标准将数据分成不同的组
- 对每个组计算统计信息(或应用一个函数)
- 类似于dplyr()函数

```
In [ ]: #Group data using rank  
df_rank = df.groupby(['rank'])
```

```
In [ ]: #Calculate mean value for each numeric column per each group  
df_rank.mean()
```

	phd	service	salary
rank			
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

groupby方法

一旦创建了groupby对象，我们可以计算每个组的各种统计信息：

```
In [ ]: #Calculate mean salary for each professor rank:  
df.groupby('rank')[['salary']].mean()
```

salary	
rank	
AssocProf	91786.230769
AsstProf	81362.789474
Prof	123624.804348

注意:如果使用单括号指定列(例如salary)，则输出为Pandas Series对象。当使用双括号时，输出是一个Data Frame。

思考：此处的 df.groupby 用法是否能用SQL编程实现？

groupby方法

groupby说明:

- 除非需要, 否则不会进行分组/拆分。创建groupby对象只验证您传递了一个有效的映射
- 默认情况下, groupby操作对组键进行排序。您可能需要传递sort=False以获得潜在的加速:

```
In [ ]: #Calculate mean salary for each professor rank:  
df.groupby(['rank'], sort=False)[['salary']].mean()
```

Data Frame: 过滤数据

对于数据的子集，我们可以应用布尔索引。这种索引通常被称为过滤器。例如，如果我们想要对工资值大于\$ 120,000的行进行子集：

```
In [ ]: #Calculate mean salary for each professor rank:  
df_sub = df[ df['salary'] > 120000 ]
```

任何布尔运算符都可以用于数据子集：

> greater; >= greater or equal;

< less; <= less or equal;

== equal; != not equal;

```
In [ ]: #Select only those rows that contain female  
professors:  
df_f = df[ df['sex'] == 'Female' ]
```

Data Frame:切片

有几种方法可以将Data Frame **切片/子集化**（选择数据的一部分）：

- 一个或多个列
- 一行或多行
- 行和列的子集

可以根据行和列的位置或标签来选择它们

Data Frame:切片

当**选择一列**时，可以使用一组括号，但结果**对象将是一个Series**(而不是Data Frame):

```
In [ ]: #Select column salary:  
df['salary']
```

当我们需要**选择多个列**使输出为Data Frame时，我们应该使用**双括号**:

```
In [ ]: #Select column salary:  
df[['rank', 'salary']]
```

Data Frame:loc

如果我们需要**选择一个行范围**，使用它们的标签，我们可以使用方法**loc**:

```
In [ ]: #Select rows by their labels:  
df_sub.loc[10:20,['rank','sex','salary']]
```

Out []:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

Data Frame:iloc

如果我们需要选择一个范围的行和/或列，使用他们的位置，我们可以使用方法**iloc**:

```
In [ ]: #Select rows by their labels:  
df_sub.iloc[10:20,[0, 3, 4, 5]]
```

Out []:

	rank	service	sex	salary
26	Prof	19	Male	148750
27	Prof	43	Male	155865
29	Prof	20	Male	123683
31	Prof	21	Male	155750
35	Prof	23	Male	126933
36	Prof	45	Male	146856
39	Prof	18	Female	129000
40	Prof	36	Female	137000
44	Prof	19	Female	151768
45	Prof	25	Female	140096

Data Frame:iloc

```
df.iloc[0]    # First row of a data frame
df.iloc[i]    #(i+1)th row
df.iloc[-1]   # Last row
```

```
df.iloc[:, 0] # First column
df.iloc[:, -1] # Last column
```

```
df.iloc[0:7]          #First 7 rows
df.iloc[:, 0:2]       #First 2 columns
df.iloc[1:3, 0:2]     #Second through third rows and first 2 columns
df.iloc[[0,5], [1,3]] #1st and 6th rows and 2nd and 4th columns
```

Data Frame:Sorting

我们可以按列中的值对数据进行**排序**。默认情况下，排序将按升序进行，并返回一个新的Data Frame。

```
In [ ]: # Create a new data frame from the original sorted by  
# the column Salary  
df_sorted = df.sort_values( by = 'service')  
df_sorted.head()
```

```
Out [ ]
```

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

Data Frame:Sorting

我们可以使用**2或更多列**对数据进行**排序**:

```
In [ ]: df_sorted = df.sort_values( by=['service', 'salary'],  
                                     ascending=[True, False])  
df_sorted.head(10)
```

```
Out [ ]:
```

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

缺失值

```
In [ ]: # Read a dataset with missing values
        flights = pd.read_csv("../flights.csv")
```

```
In [ ]: # Select the rows that have at least one missing value
        flights[flights.isnull().any(axis=1)].head()
```

```
Out[ ]:
```

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest	air_time	distance	hour	minute
330	2013	1	1	1807.0	29.0	2251.0	NaN	UA	N31412	1228	EWB	SAN	NaN	2425	18.0	7.0
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EHAA	791	LGA	DFW	NaN	1389	NaN	NaN
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	1925	LGA	MIA	NaN	1096	NaN	NaN
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	1299	EWB	RSW	NaN	1068	21.0	45.0
858	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	133	JFK	LAX	NaN	2475	NaN	NaN

处理Data Frame中的缺失值

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

缺失值

- 在对**数据求和**时，**缺失值将被视为零**
- 如果所有值都缺失，则总和将等于NaN
- **GroupBy方法不涉及缺失值**
- 许多描述性统计方法都有skipna选项来控制是否应该排除缺失值，该值默认设置为True

Pandas中的聚合函数

聚合:

- 计算和或均值
- 计算组大小/计数

聚合函数:

min, max

count, sum, prod

mean, median, mode, mad

std, var

当每列计算多个统计信息时, **Agg()方法**很有用:

```
In [ ]: flights[['dep_delay', 'arr_delay']].agg(['min', 'mean', 'max'])
```

Out []:

	dep_delay	arr_delay
min	-16.000000	-62.000000
mean	9.384302	2.298675
max	351.000000	389.000000

Pandas中的聚合函数

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation
sem	Standard error of mean
skew	Sample skewness
kurt	kurtosis