Numpy（续）
矩阵分解、数据降噪与聚类 C07

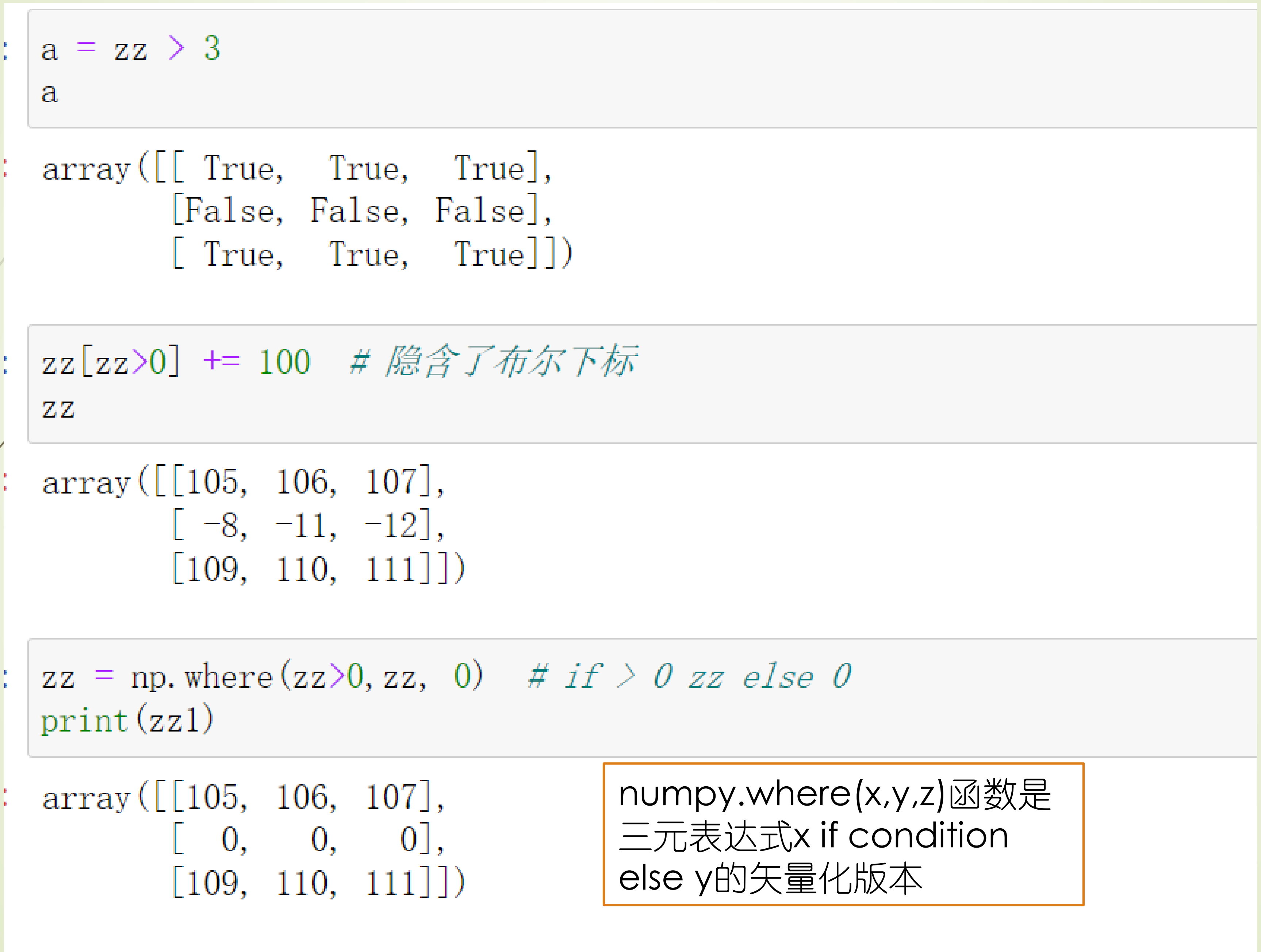
胡俊峰 2022/03/21
北京大学计算机学院

主要内容

- Numpy（续）

- 主成分分析（PCA）与奇异值分解
- K-means聚类算法
- 非负矩阵分解（NMF）
- 模型正则化与Robust PCA

按条件选择下标：



```
a = zz > 3
a
```

```
array([[ True,  True,  True],
       [False, False, False],
       [ True,  True,  True]])
```

```
zz[zz>0] += 100  # 隐含了布尔下标
zz
```

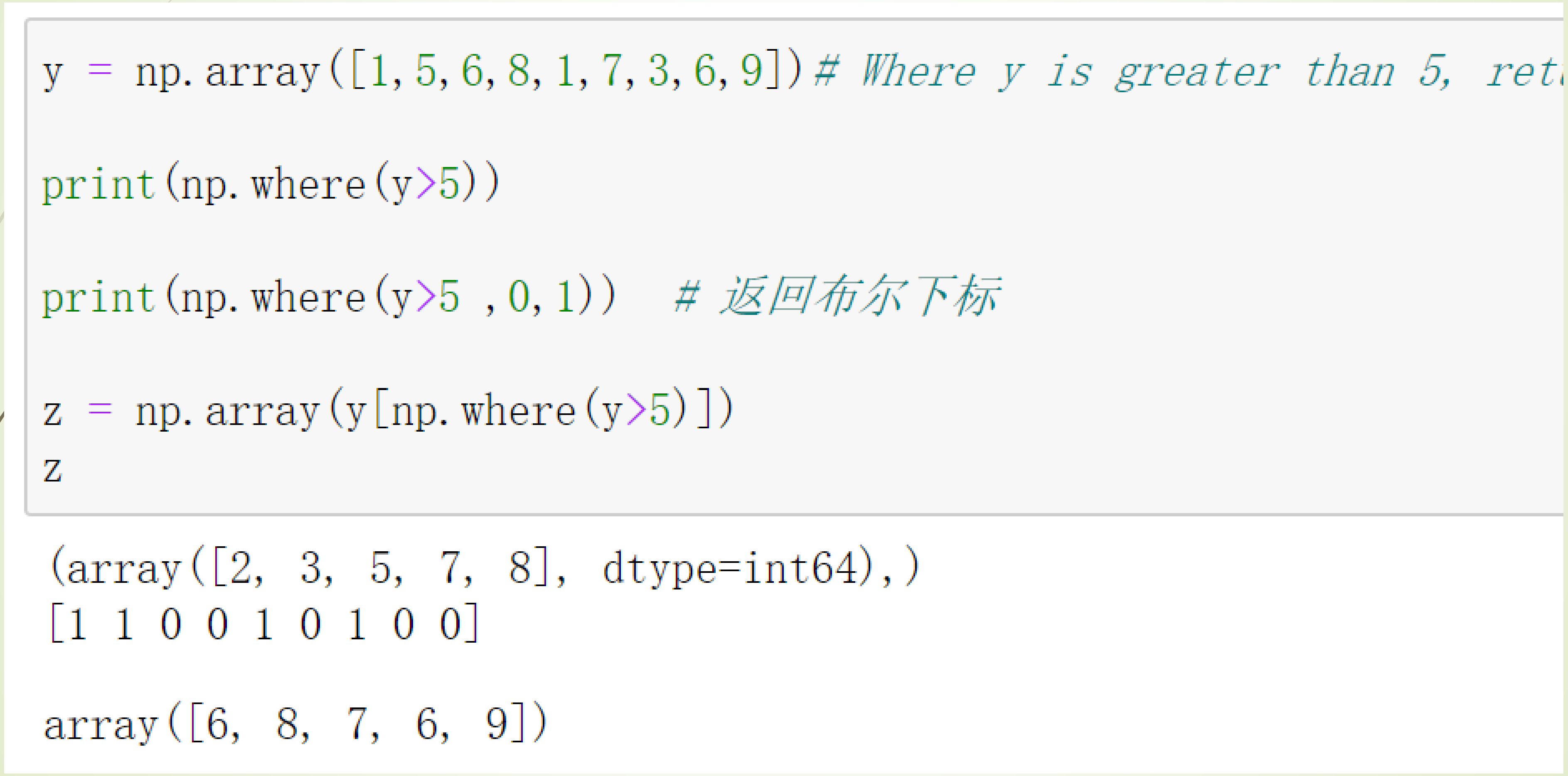
```
array([[105, 106, 107],
       [ -8, -11, -12],
       [109, 110, 111]])
```

```
zz = np.where(zz>0, zz, 0)  # if > 0 zz else 0
print(zz1)
```

```
array([[105, 106, 107],
       [  0,   0,   0],
       [109, 110, 111]])
```

numpy.where(x,y,z)函数是三元表达式x if condition else y的矢量化版本

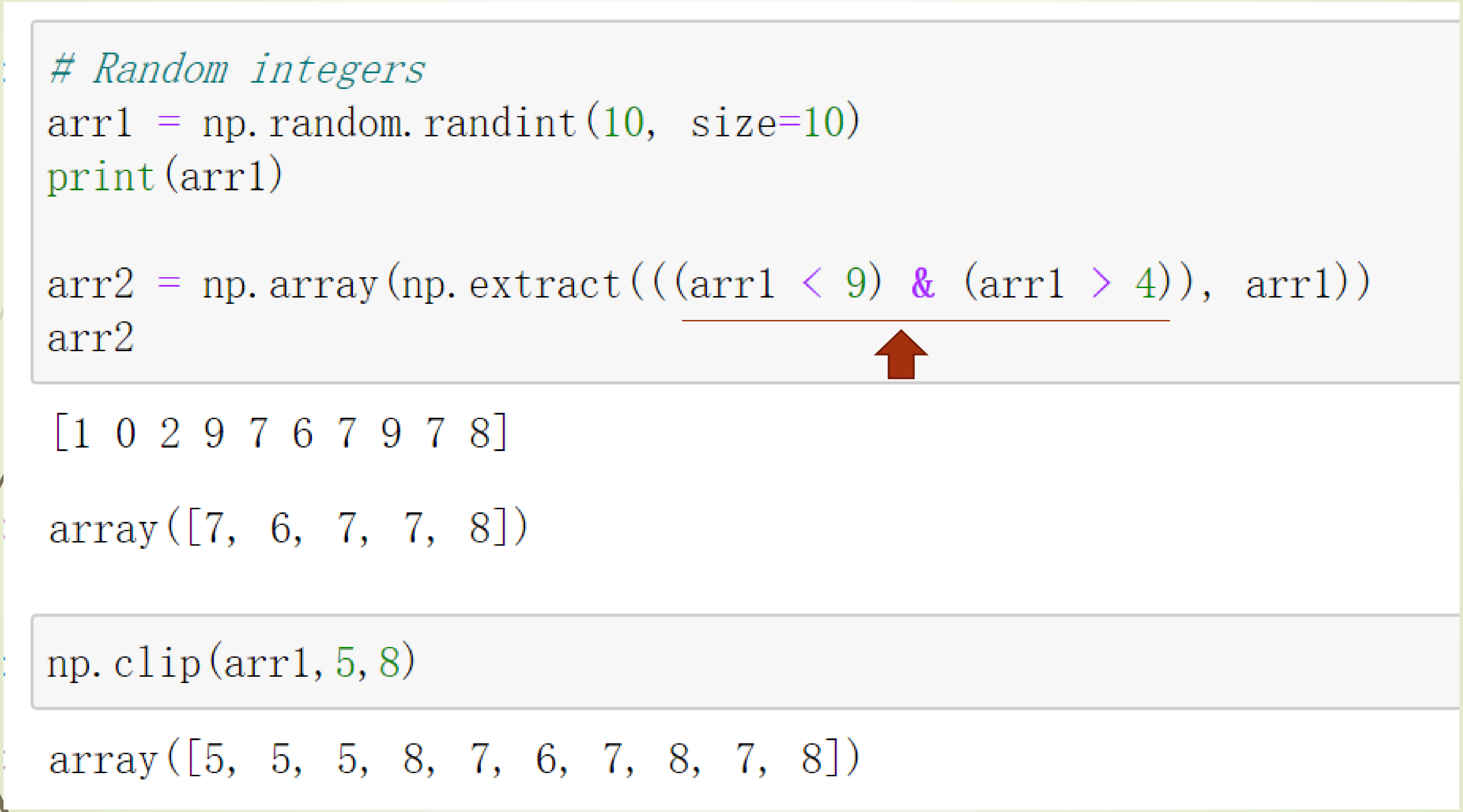
用where()函数进行条件筛选



```
y = np.array([1,5,6,8,1,7,3,6,9])# Where y is greater than 5, ret

print(np.where(y>5))

print(np.where(y>5 ,0,1))   # 返回布尔下标

z = np.array(y[np.where(y>5)])
z
```

```
(array([2,  3,  5,  7,  8], dtype=int64),)
[1 1 0 0 1 0 1 0 0]

array([6,  8,  7,  6,  9])
```

extract(), clip():



```
# Random integers
arr1 = np.random.randint(10, size=10)
print(arr1)

arr2 = np.array(np.extract(((arr1 < 9) & (arr1 > 4)), arr1))
arr2
```

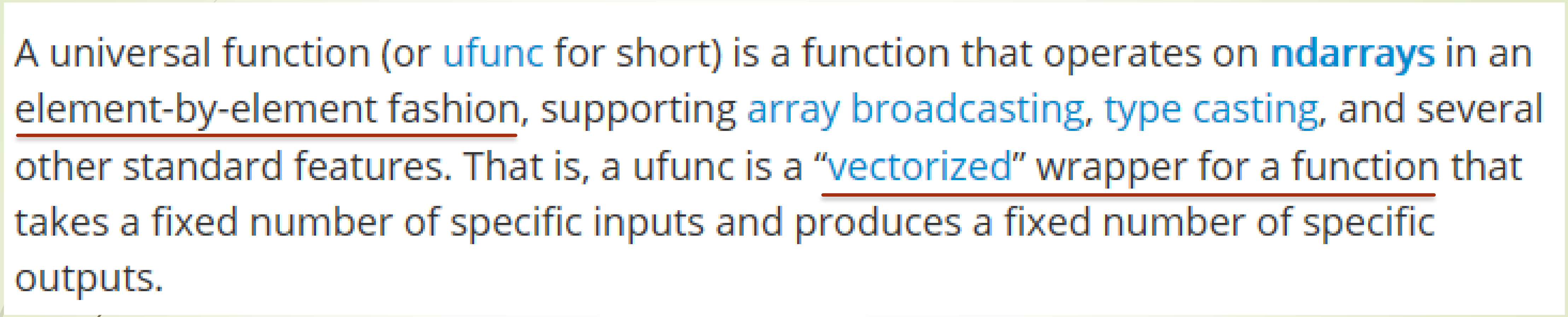
```
[1 0 2 9 7 6 7 9 7 8]
```

```
array([7, 6, 7, 7, 8])
```

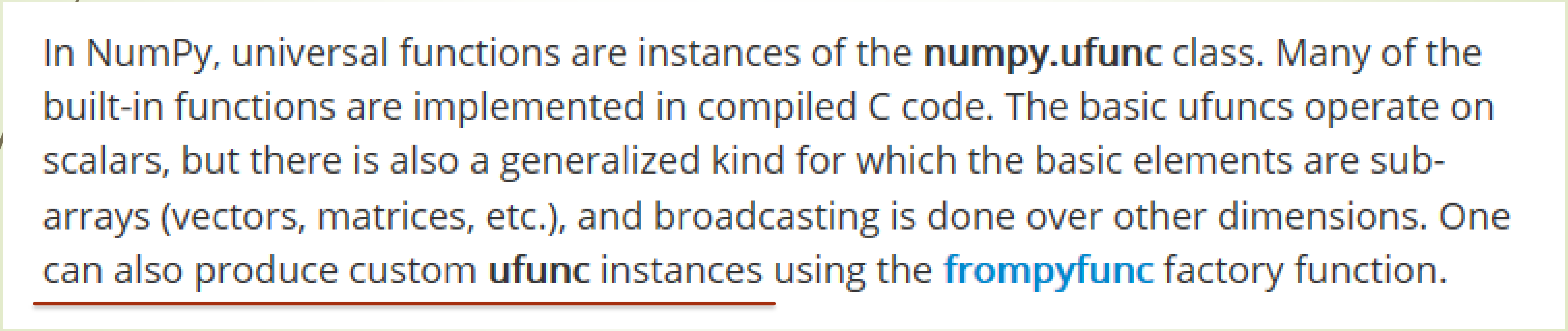
```
np.clip(arr1,5,8)
```

```
array([5, 5, 5, 8, 7, 6, 7, 8, 7, 8])
```

Universal functions (ufunc)：逐元素操作函数



A universal function (or ufunc for short) is a function that operates on **ndarrays** in an element-by-element fashion, supporting array broadcasting, type casting, and several other standard features. That is, a ufunc is a “vectorized” wrapper for a function that takes a fixed number of specific inputs and produces a fixed number of specific outputs.

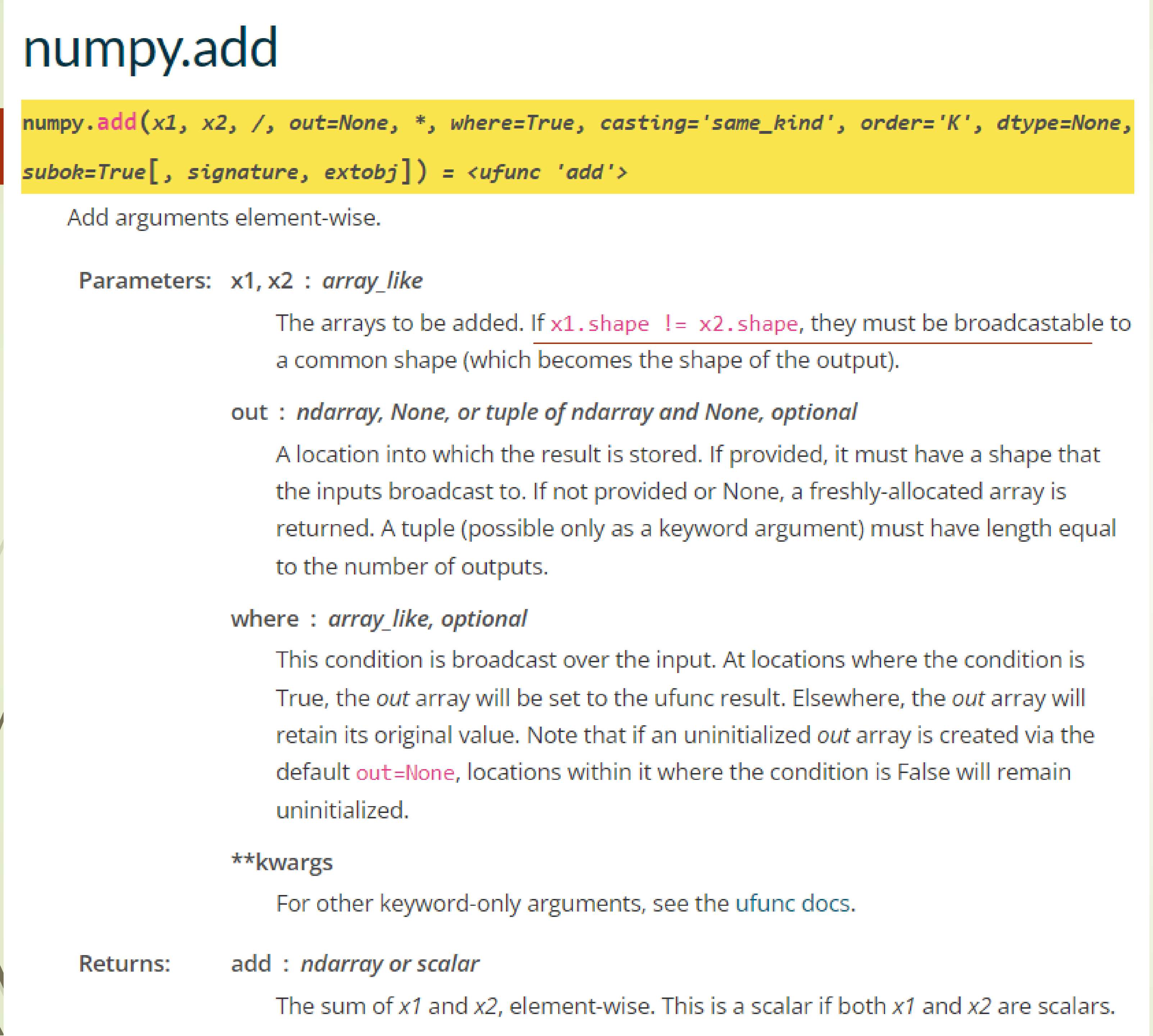


In NumPy, universal functions are instances of the **numpy.ufunc** class. Many of the built-in functions are implemented in compiled C code. The basic ufuncs operate on scalars, but there is also a generalized kind for which the basic elements are sub-arrays (vectors, matrices, etc.), and broadcasting is done over other dimensions. One can also produce custom **ufunc** instances using the **frompyfunc** factory function.

Math operations

add(x1, x2, /[, out, where, casting, order, ...])	Add arguments element-wise.
subtract(x1, x2, /[, out, where, casting, ...])	Subtract arguments, element-wise.
multiply(x1, x2, /[, out, where, casting, ...])	Multipl
matmul(x1, x2, /[, out, casting, order, ...])	Matrix
divide(x1, x2, /[, out, where, casting, ...])	Return
logaddexp(x1, x2, /[, out, where, casting, ...])	Logarit inputs.
logaddexp2(x1, x2, /[, out, where, casting, ...])	Logarit in base
true_divide(x1, x2, /[, out, where, ...])	Return

sign(x, /[, out, where, casting, order, ...])	Returns an e number.
heaviside(x1, x2, /[, out, where, casting, ...])	Compute the
conj(x, /[, out, where, casting, order, ...])	Return the c
conjugate(x, /[, out, where, casting, ...])	Return the c
exp(x, /[, out, where, casting, order, ...])	Calculate the array.



numpy.add

```
numpy.add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj]) = <ufunc 'add'>
```

Add arguments element-wise.

Parameters: x1, x2 : *array_like*

The arrays to be added. If `x1.shape != x2.shape`, they must be broadcastable to a common shape (which becomes the shape of the output).

out : *ndarray, None, or tuple of ndarray and None, optional*

A location into which the result is stored. If provided, it must have a shape that the inputs broadcast to. If not provided or None, a freshly-allocated array is returned. A tuple (possible only as a keyword argument) must have length equal to the number of outputs.

where : *array_like, optional*

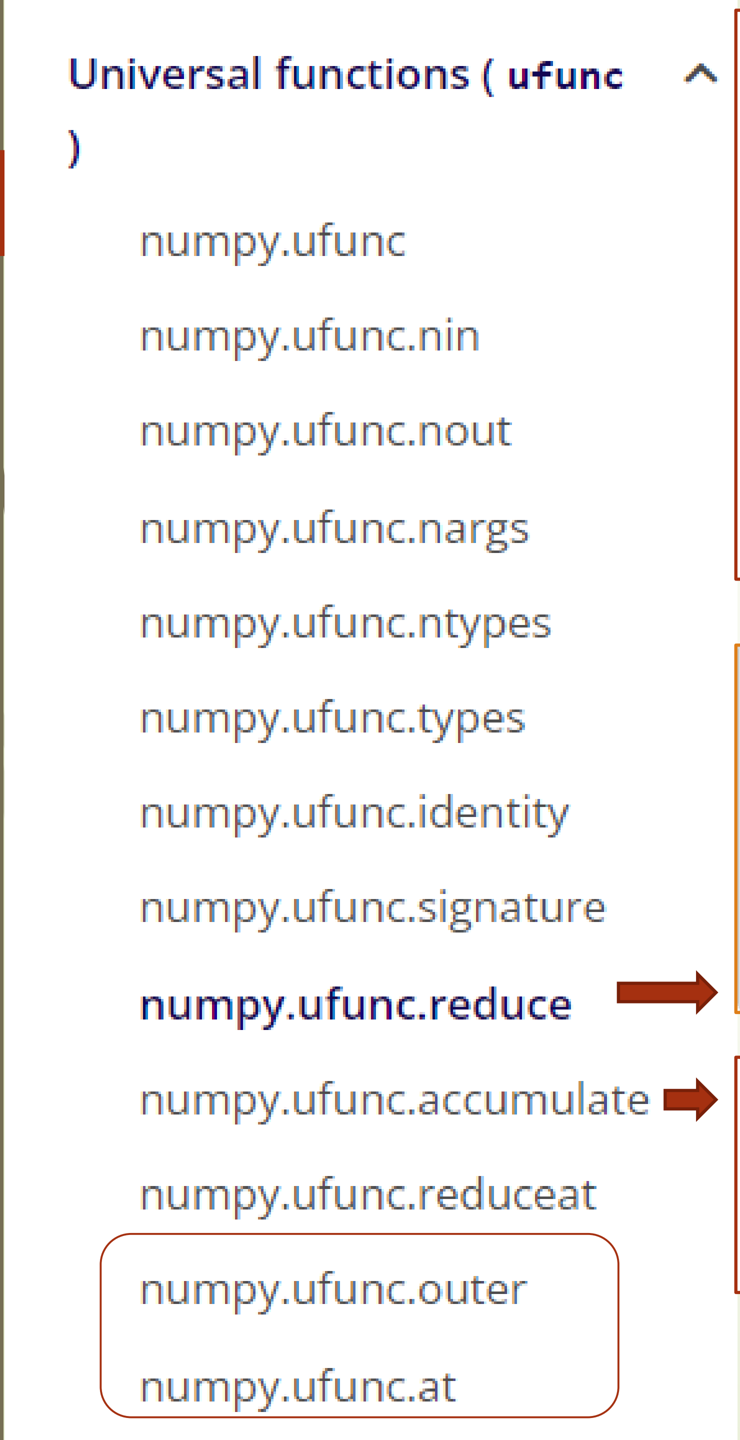
This condition is broadcast over the input. At locations where the condition is True, the *out* array will be set to the ufunc result. Elsewhere, the *out* array will retain its original value. Note that if an uninitialized *out* array is created via the default `out=None`, locations within it where the condition is False will remain uninitialized.

**kwargs

For other keyword-only arguments, see the ufunc docs.

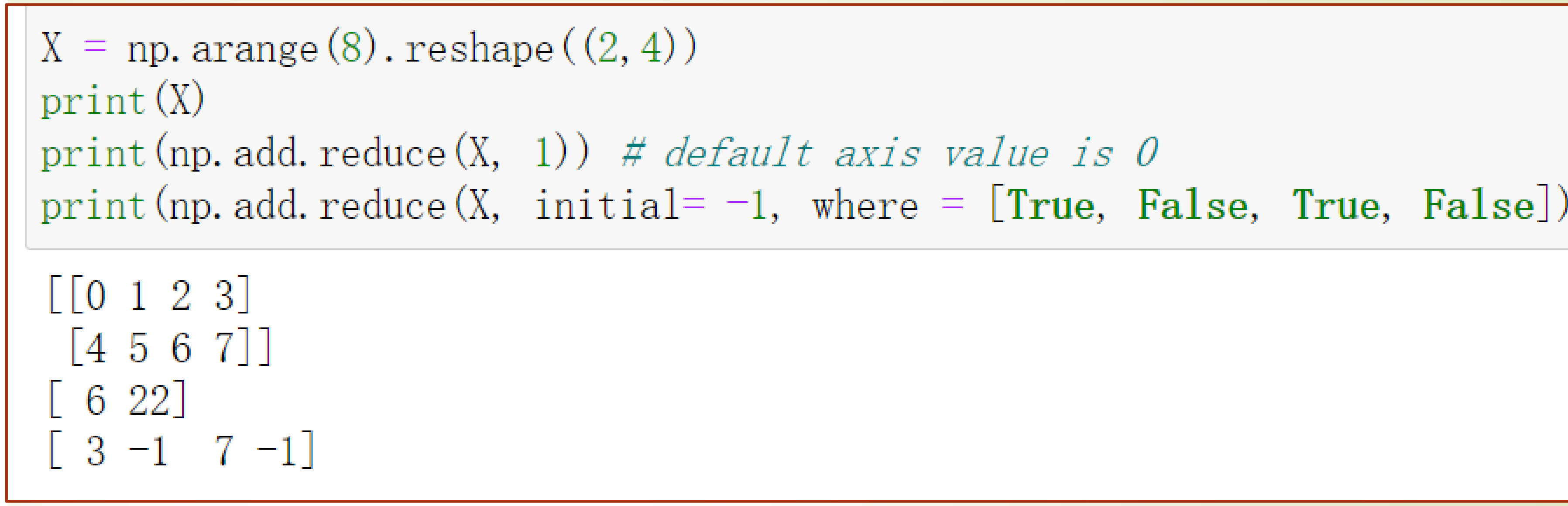
Returns: add : *ndarray or scalar*

The sum of *x1* and *x2*, element-wise. This is a scalar if both *x1* and *x2* are scalars.



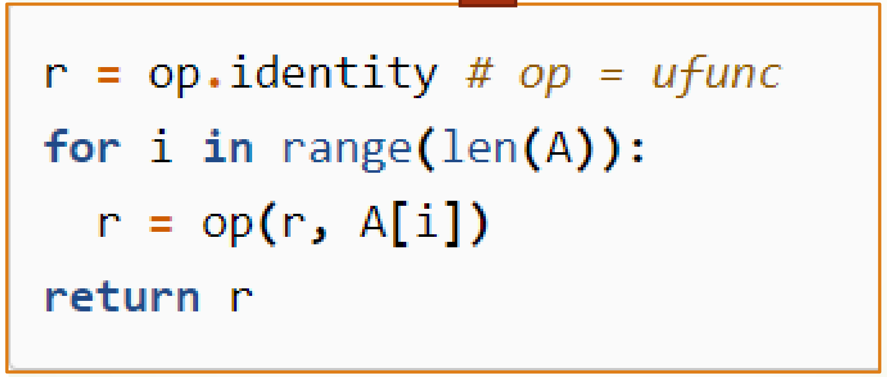
Universal functions (ufunc)

- numpy.ufunc
- numpy.ufunc.nin
- numpy.ufunc.nout
- numpy.ufunc.nargs
- numpy.ufunc.ntypes
- numpy.ufunc.types
- numpy.ufunc.identity
- numpy.ufunc.signature
- **numpy.ufunc.reduce**
- numpy.ufunc.accumulate
- numpy.ufunc.reduceat
- numpy.ufunc.outer
- numpy.ufunc.at

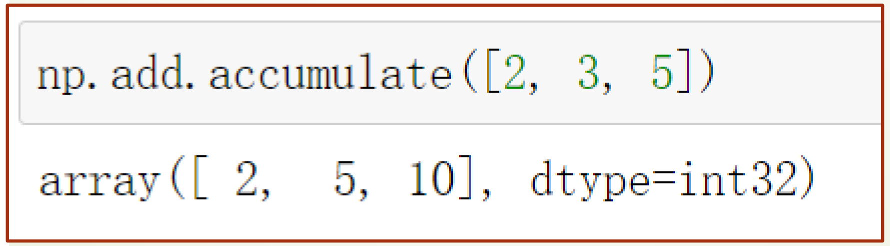


```
X = np.arange(8).reshape((2,4))
print(X)
print(np.add.reduce(X, 1)) # default axis value is 0
print(np.add.reduce(X, initial= -1, where = [True, False, True, False])
```

```
[[0 1 2 3]
 [4 5 6 7]]
[ 6 22]
[ 3 -1  7 -1]
```



```
r = op.identity # op = ufunc
for i in range(len(A)):
  r = op(r, A[i])
return r
```

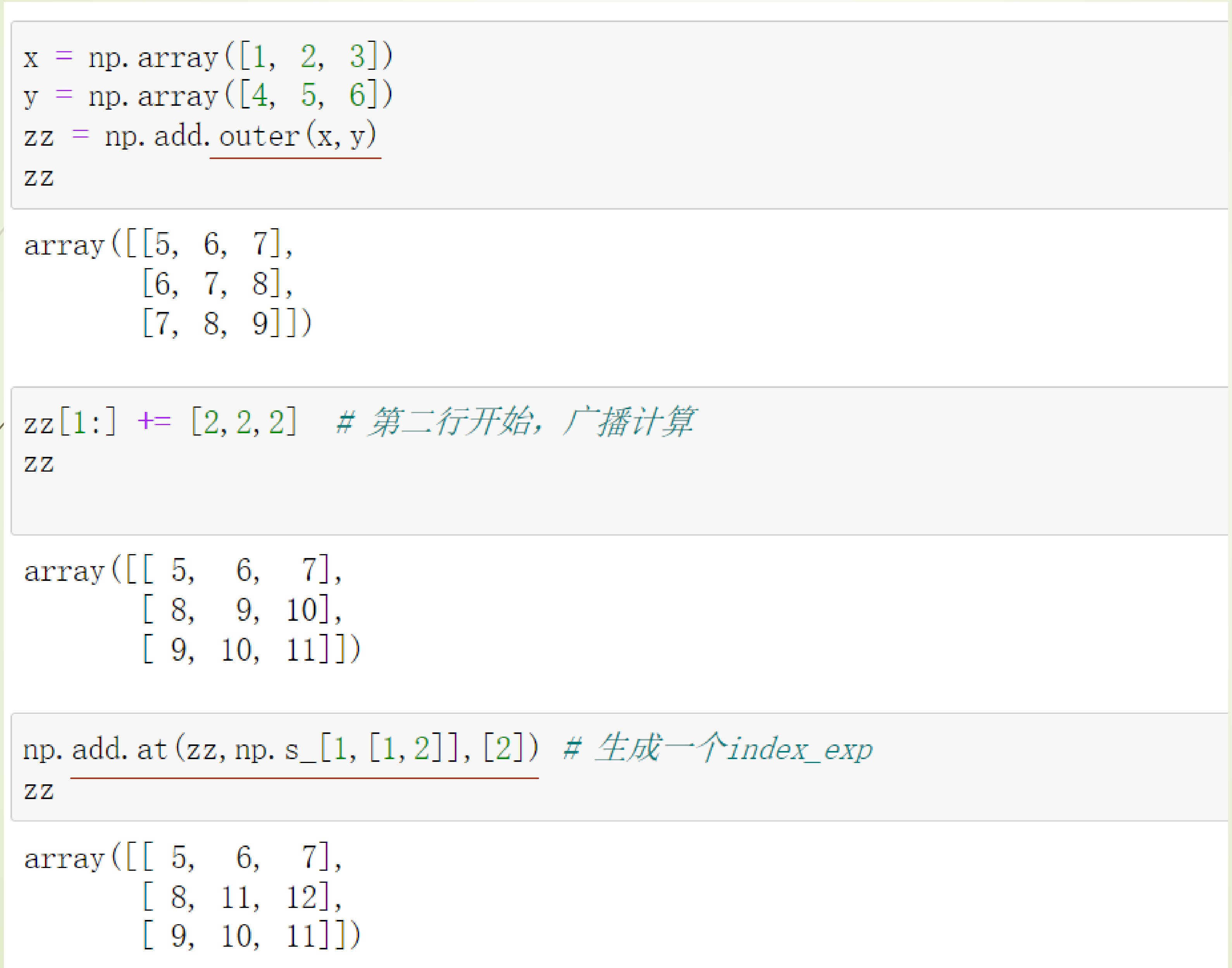


```
np.add.accumulate([2, 3, 5])
```

```
array([ 2,  5, 10], dtype=int32)
```

累进计算模式

ufunc使用例子：



```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
zz = np.add.outer(x,y)
zz
```

```
array([[5, 6, 7],
       [6, 7, 8],
       [7, 8, 9]])
```

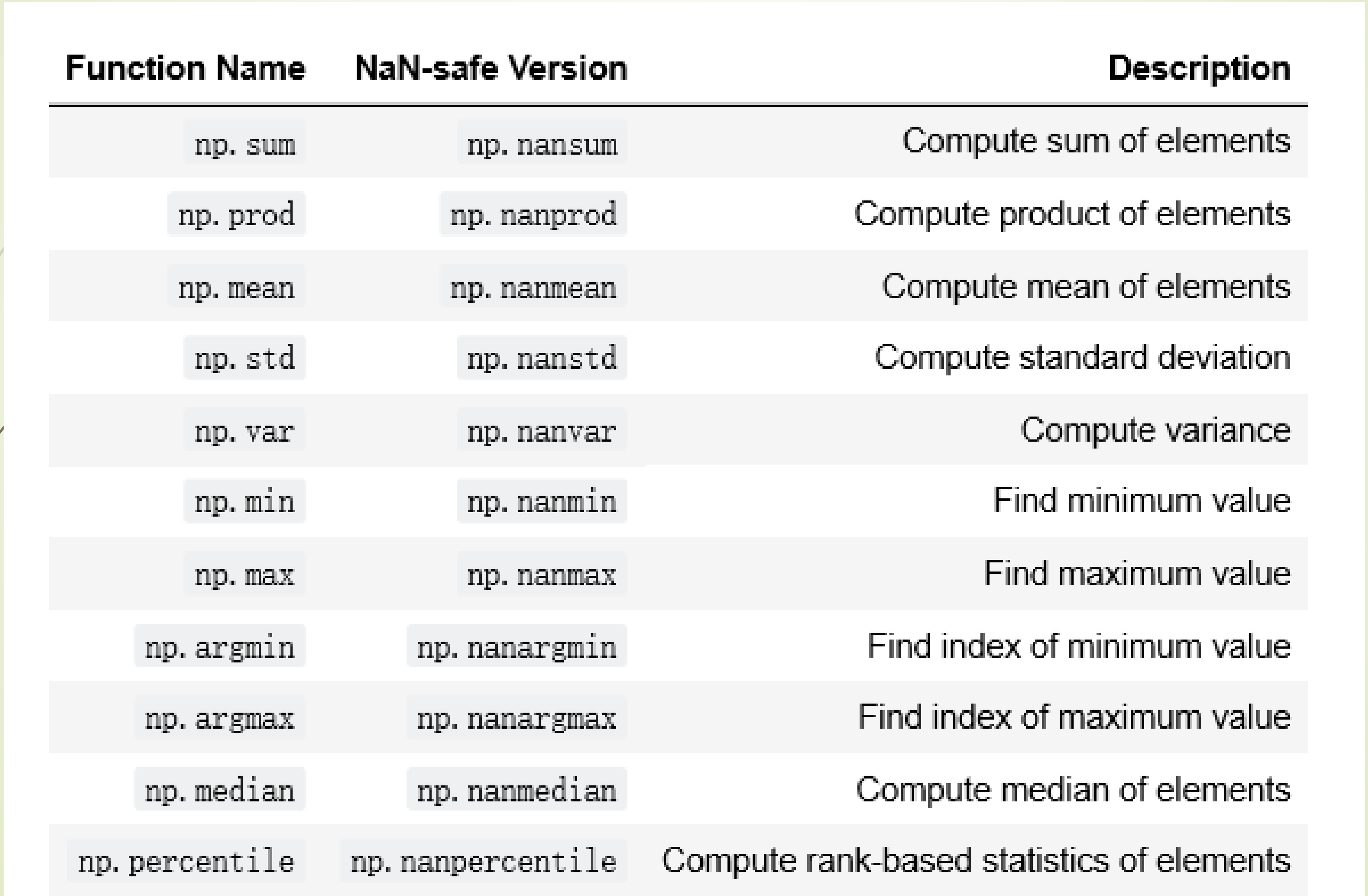
```
zz[1:] += [2,2,2]  # 第二行开始，广播计算
zz
```

```
array([[ 5,  6,  7],
       [ 8,  9, 10],
       [ 9, 10, 11]])
```

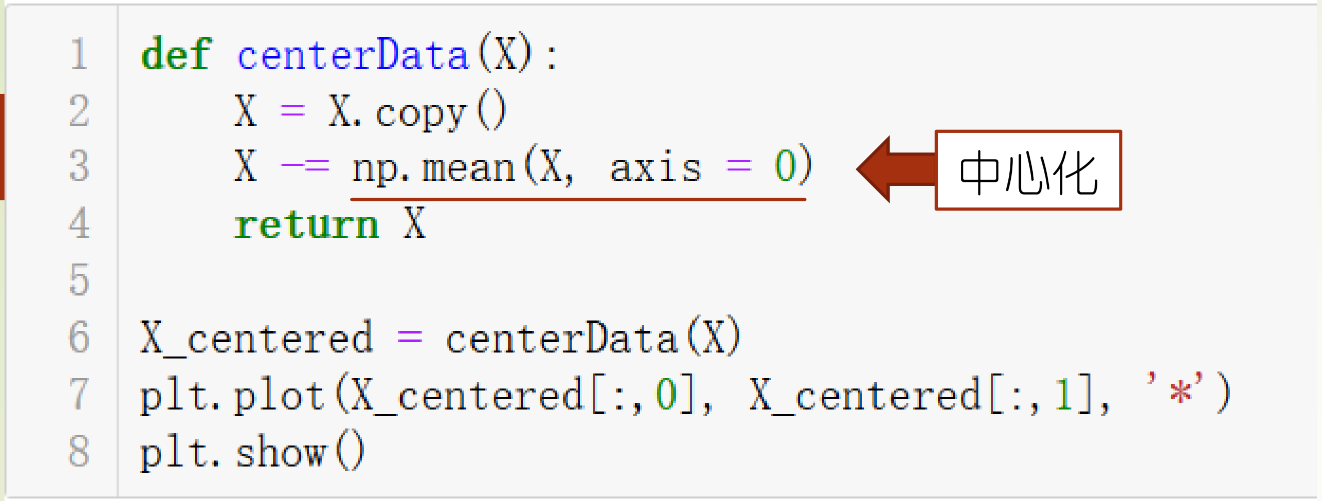
```
np.add.at(zz,np.s_[1,[1,2]],[2]) # 生成一个index_exp
zz
```

```
array([[ 5,  6,  7],
       [ 8, 11, 12],
       [ 9, 10, 11]])
```

Aggregation functions（聚合函数 group by）

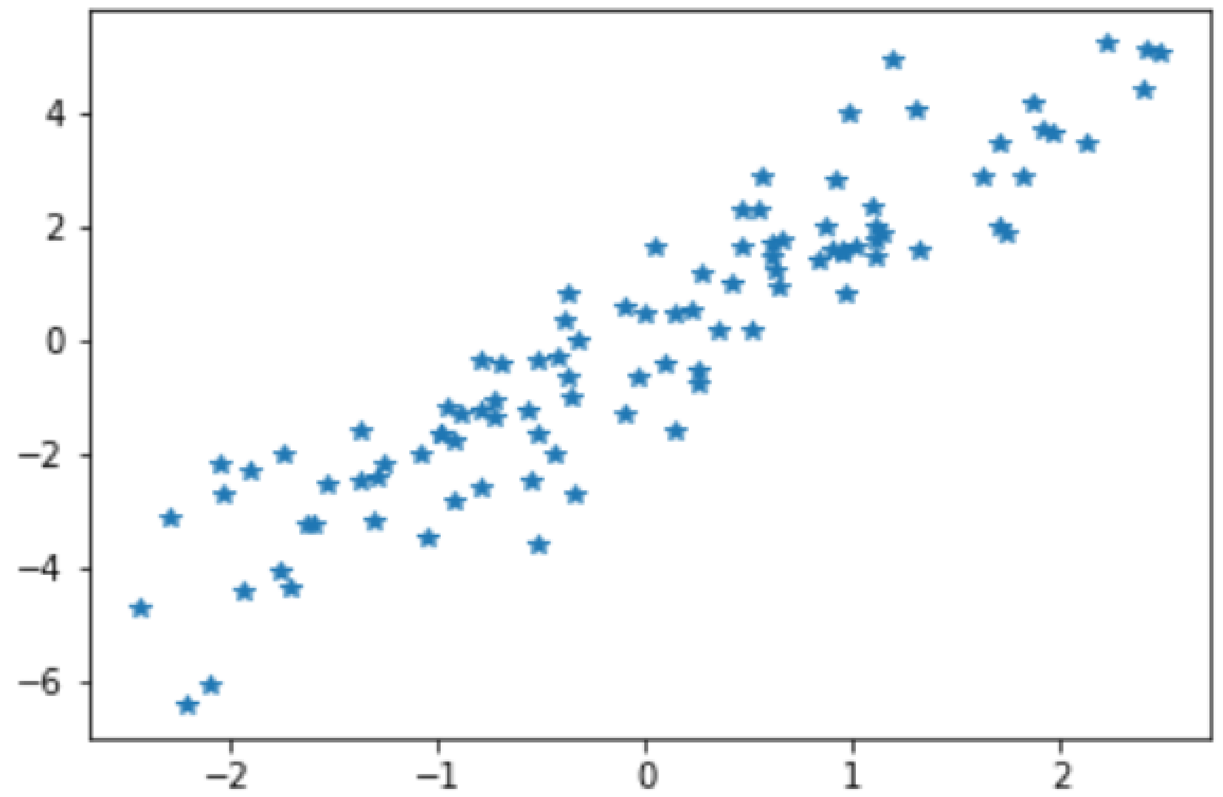


Function Name	NaN-safe Version	Description
`np.sum`	`np.nansum`	Compute sum of elements
`np.prod`	`np.nanprod`	Compute product of elements
`np.mean`	`np.nanmean`	Compute mean of elements
`np.std`	`np.nanstd`	Compute standard deviation
`np.var`	`np.nanvar`	Compute variance
`np.min`	`np.nanmin`	Find minimum value
`np.max`	`np.nanmax`	Find maximum value
`np.argmin`	`np.nanargmin`	Find index of minimum value
`np.argmax`	`np.nanargmax`	Find index of maximum value
`np.median`	`np.nanmedian`	Compute median of elements
`np.percentile`	`np.nanpercentile`	Compute rank-based statistics of elements

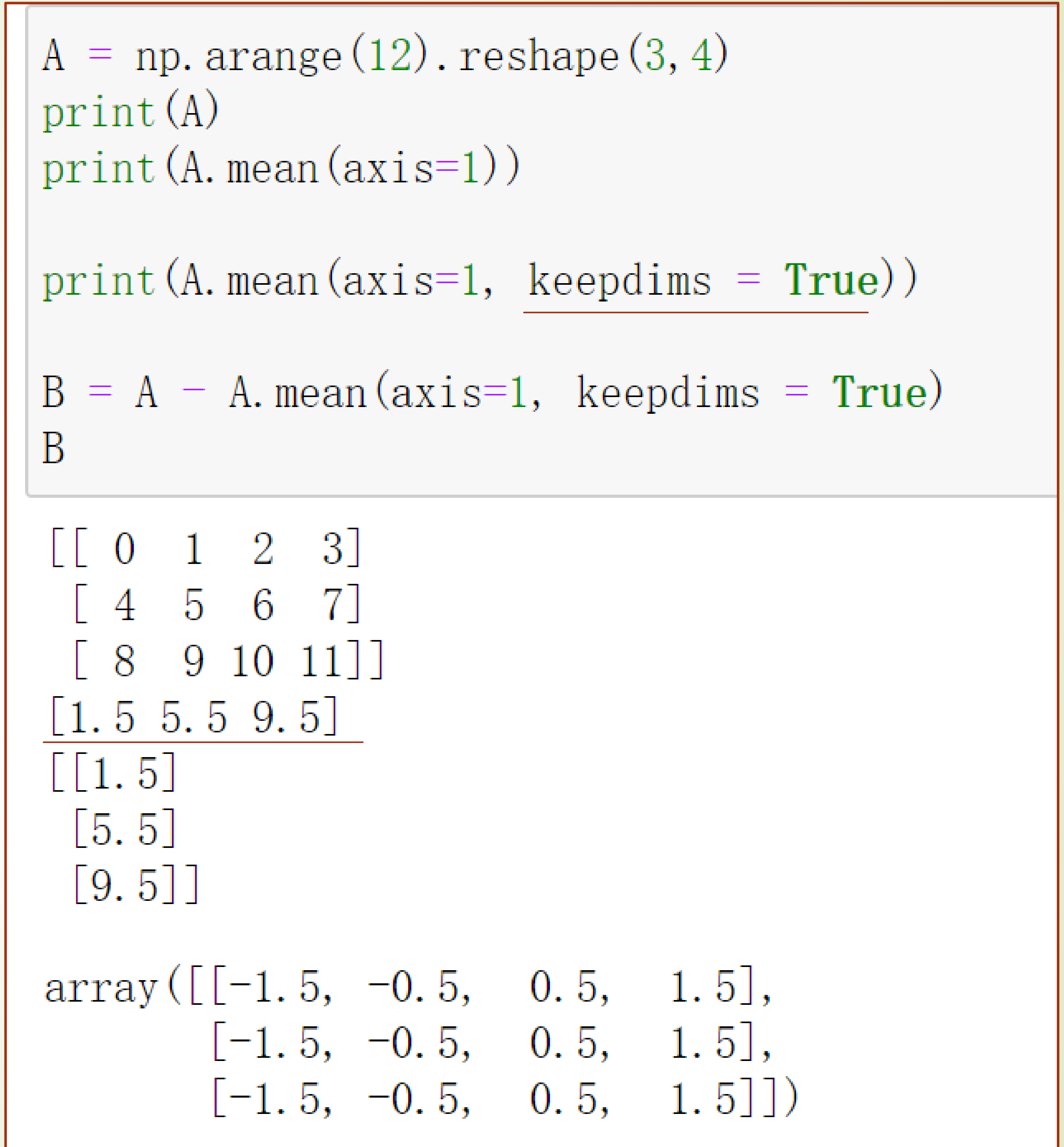


```
def centerData(X):
    X = X.copy()
    X -= np.mean(X, axis = 0)
    return X

X_centered = centerData(X)
plt.plot(X_centered[:,0], X_centered[:,1], '*')
plt.show()
```

中心化

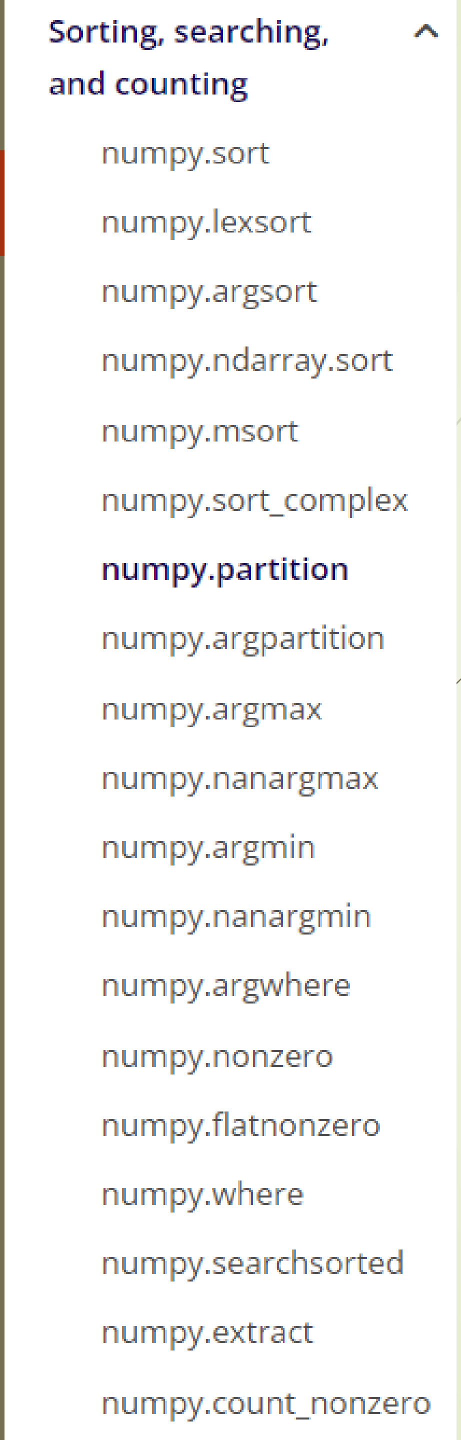


按行中心化：



```python
A = np.arange(12).reshape(3,4)
print(A)
print(A.mean(axis=1))

print(A.mean(axis=1, keepdims = True))

B = A - A.mean(axis=1, keepdims = True)
B
```

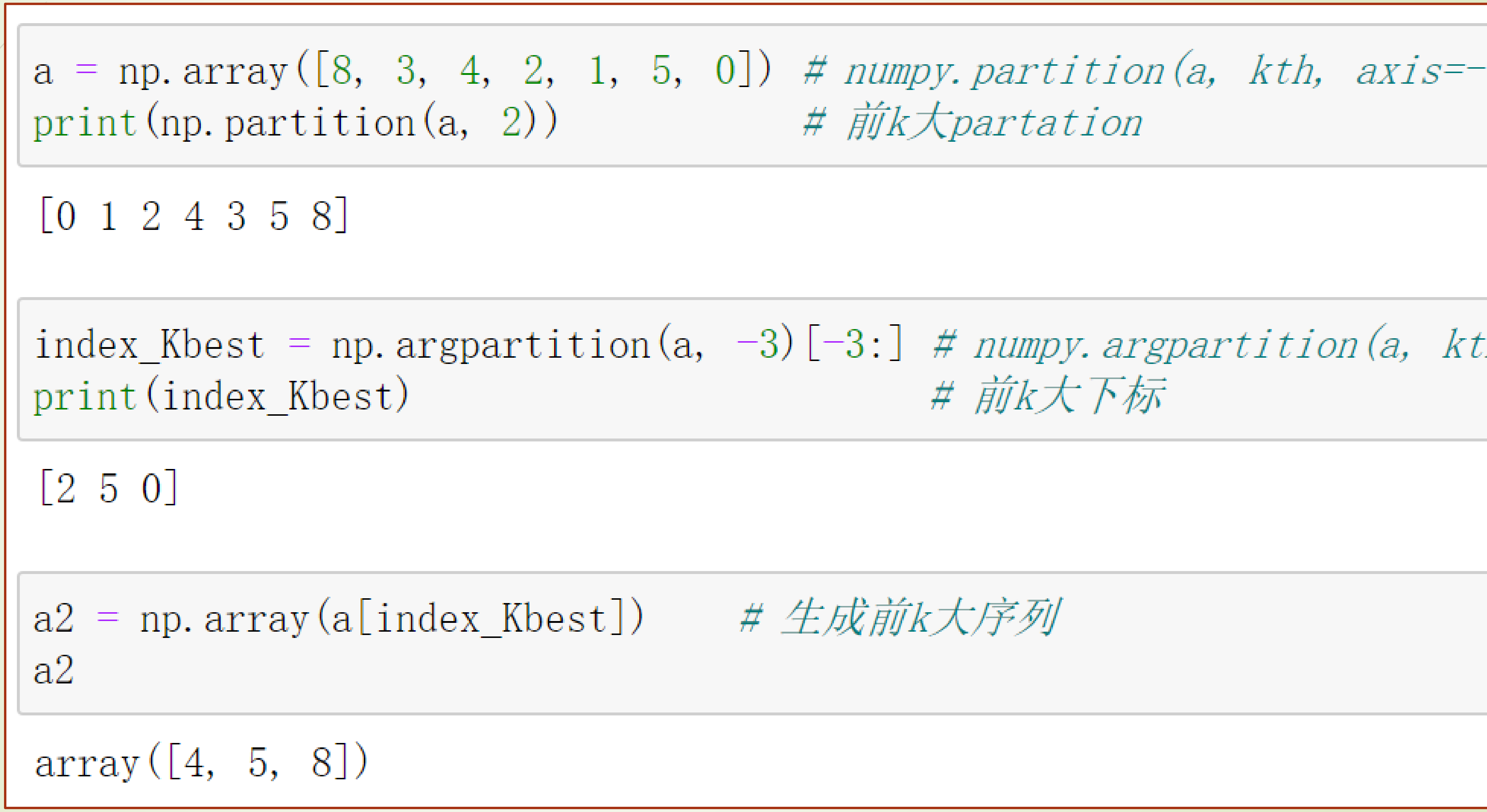
```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[1.5 5.5 9.5]
[[1.5]
 [5.5]
 [9.5]]

array([[-1.5, -0.5,  0.5,  1.5],
       [-1.5, -0.5,  0.5,  1.5],
       [-1.5, -0.5,  0.5,  1.5]])
```



Sorting, searching, and counting

numpy.sort
numpy.lexsort
numpy.argsort
numpy.ndarray.sort
numpy.msort
numpy.sort_complex
numpy.partition
numpy.argpartition
numpy.argmax
numpy.nanargmax
numpy.argmin
numpy.nanargmin
numpy.argwhere
numpy.nonzero
numpy.flatnonzero
numpy.where
numpy.searchsorted
numpy.extract
numpy.count_nonzero

排序与计数函数：



```
a = np.array([8, 3, 4, 2, 1, 5, 0]) # numpy.partition(a, kth, axis=-
print(np.partition(a, 2))           # 前k大partation
```

```
[0 1 2 4 3 5 8]
```

```
index_Kbest = np.argpartition(a, -3)[-3:] # numpy.argpartition(a, kt
print(index_Kbest)                        # 前k大下标
```

```
[2 5 0]
```

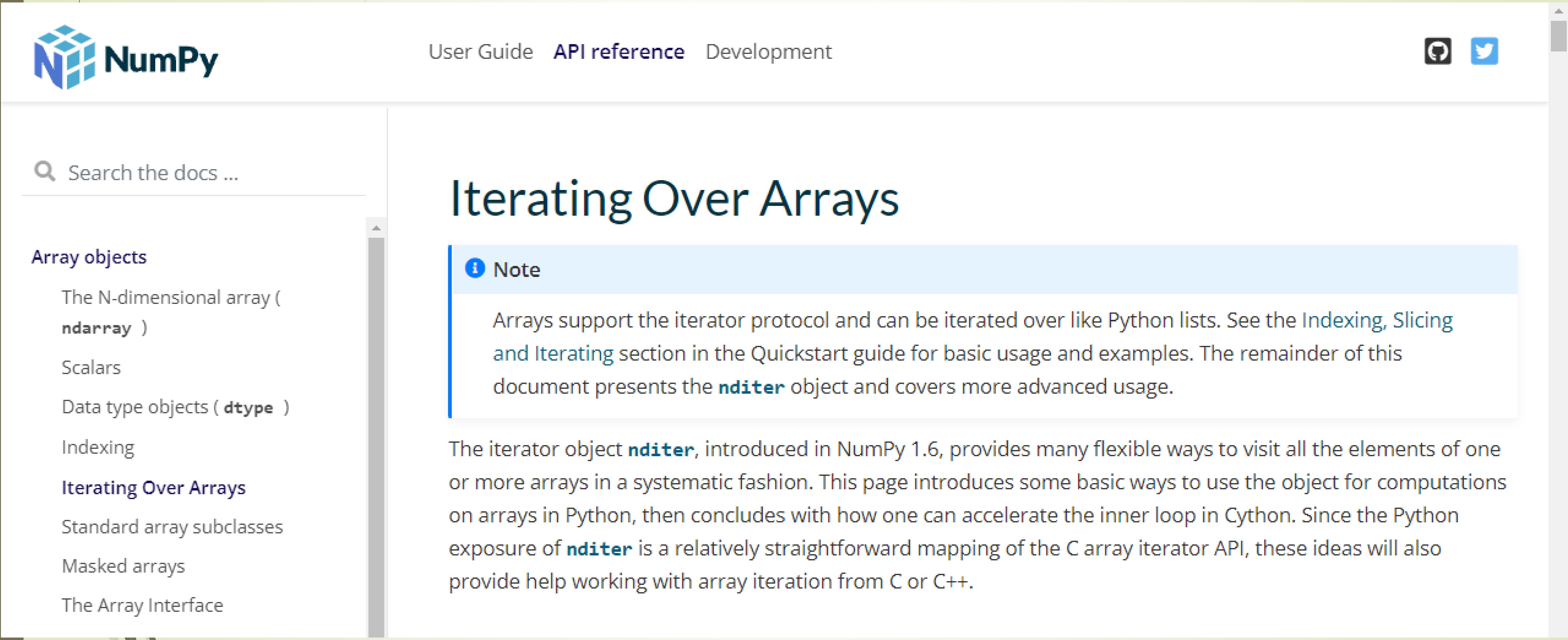
```
a2 = np.array(a[index_Kbest])    # 生成前k大序列
a2
```

```
array([4, 5, 8])
```

内置的ufunc还包括有：

- 三角函数 trigonometric functions
- 位运算函数： Bit-twiddling functions
- 比较函数: greater, less, equal,…
- 浮点数操作函数： floor， trunc， fabs…
- 统计函数： var， std， correlate, corrcoef, …

Ndarray的迭代器接口：



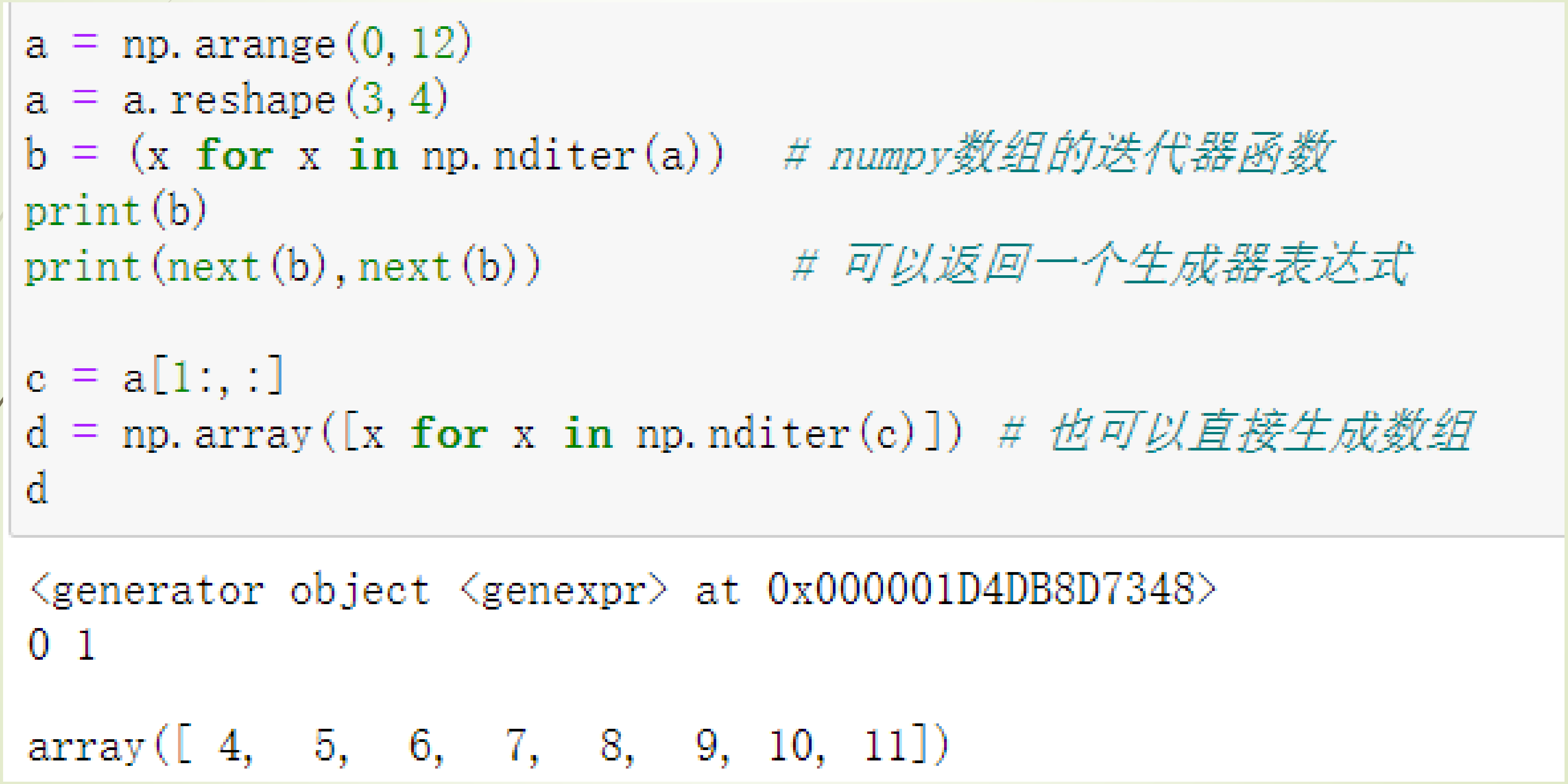
Iterating Over Arrays

Note

Arrays support the iterator protocol and can be iterated over like Python lists. See the Indexing, Slicing and Iterating section in the Quickstart guide for basic usage and examples. The remainder of this document presents the **`nditer`** object and covers more advanced usage.

The iterator object **`nditer`**, introduced in NumPy 1.6, provides many flexible ways to visit all the elements of one or more arrays in a systematic fashion. This page introduces some basic ways to use the object for computations on arrays in Python, then concludes with how one can accelerate the inner loop in Cython. Since the Python exposure of **`nditer`** is a relatively straightforward mapping of the C array iterator API, these ideas will also provide help working with array iteration from C or C++.

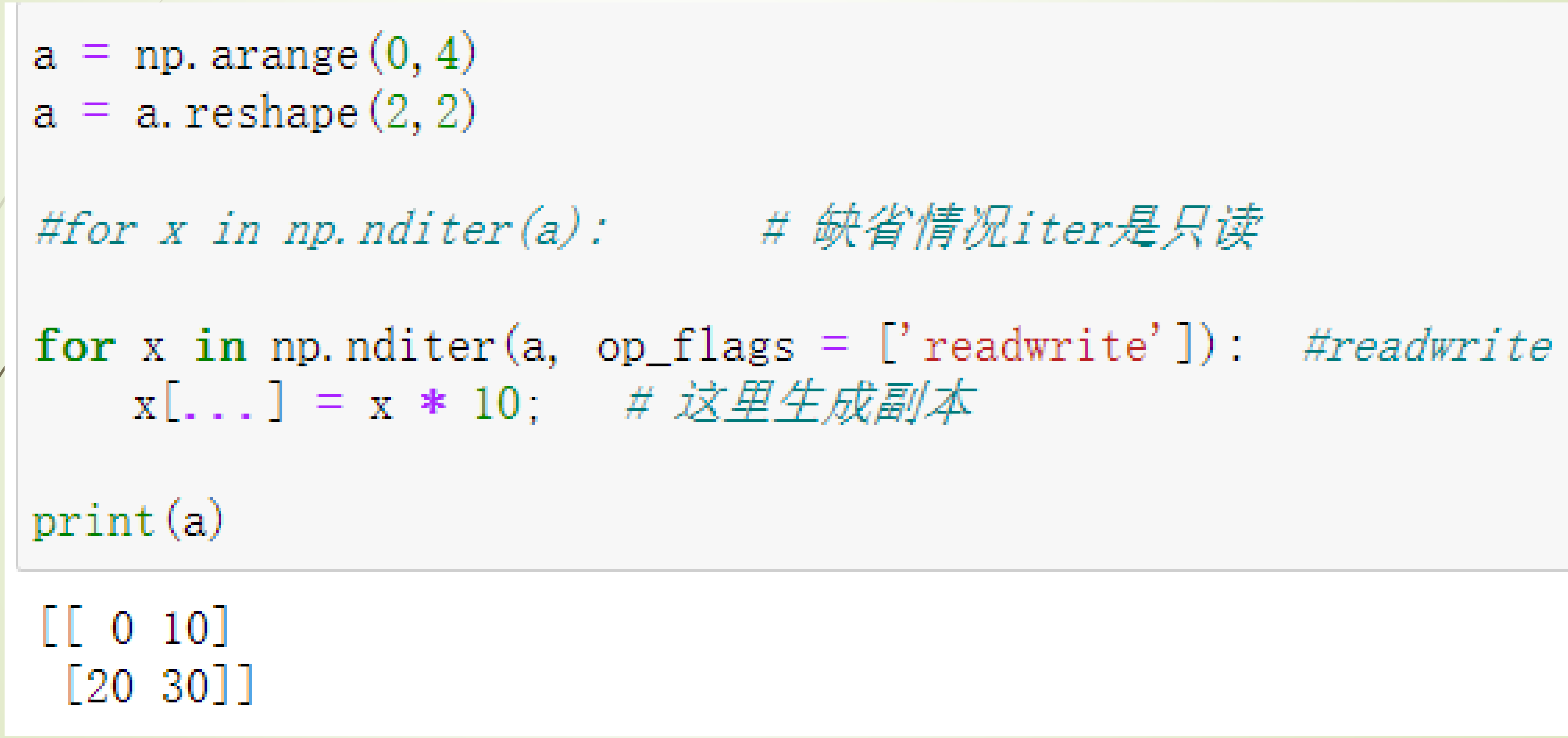
Numpy数组可以返回一个（单维）迭代器



```python
a = np.arange(0,12)
a = a.reshape(3,4)
b = (x for x in np.nditer(a))   # numpy数组的迭代器函数
print(b)
print(next(b),next(b))           # 可以返回一个生成器表达式

c = a[1:,:]
d = np.array([x for x in np.nditer(c)]) # 也可以直接生成数组
d
```

```
<generator object <genexpr> at 0x000001D4DB8D7348>
0 1

array([ 4,  5,  6,  7,  8,  9, 10, 11])
```

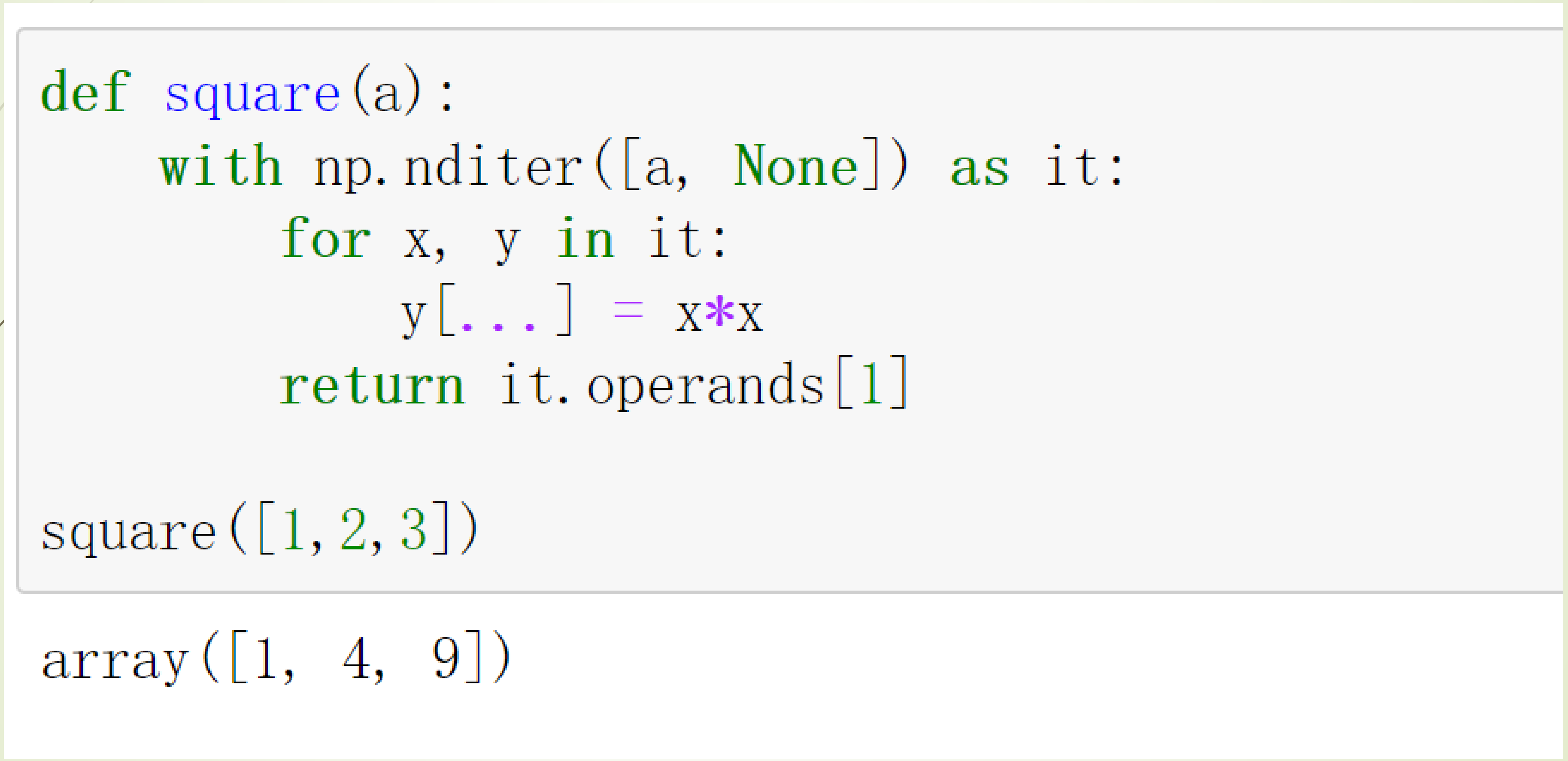
nditer用于循环结构



```
a = np.arange(0,4)
a = a.reshape(2,2)

#for x in np.nditer(a):        # 缺省情况iter是只读

for x in np.nditer(a, op_flags = ['readwrite']):  #readwrite
    x[...] = x * 10;     # 这里生成副本

print(a)
```

```
[[ 0 10]
 [20 30]]
```

可以实现为函数（C接口）



```
def square(a):
    with np.nditer([a, None]) as it:
        for x, y in it:
            y[...] = x*x
        return it.operands[1]

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

```
array([1, 4, 9])
```

把数组保存到文件与读取数组文件:

```python
# 文件保存方式:  save ,savetext,savez_compressed
ar1 = np.arange(10)
ar2 = np.arange(-5,6,2).reshape(2,3)
np.save(’some_array’,ar1)          # 保存数组
ar3 = np.load(’some_array.npy’)   # 读取数组
print(ar3)
np.savez(’array_archive.npz’,a = ar1,b =ar2) # 可将多个数组保存到一个未压缩文件中
zipfiles = np.load(’array_archive.npz’)
print(zipfiles.files)
ar6 = zipfiles[’b’]
ar6
```

```
[0 1 2 3 4 5 6 7 8 9]
[’a’, ’b’]

array([[-5, -3, -1],
       [ 1,  3,  5]])
```

Numpy部分内容总结

- Numpy提供了一个面向矩阵访问，矩阵算术计算，矩阵向量计算及广播机制的软件平台
  - 与C语言的函数街廓和数据结构兼容性保证了其执行效率
  - 附带的科学计算函数集为相关应用提供了方便
  - 矩阵运算也是当前神经网络计算的基础
- 矩阵分解本质上可以认为是一种反向的模型参数计算方案。在给定损失目标的情况下，返回最优解或可行解。通常可以由一组矩阵计算的流程迭代实现

数据降维、特征压缩、特征分解

- SVD：特征空间变换、推荐与协同过滤
- PCA：数据降维、数据降噪、最优数据编码
- NMF（VQ）：特征分解-特征组合泛化

奇异值分解（SVD）与 特征分解

- 奇异值分解的数学方案
- 特征分解的物理本质
- 隐含语义挖掘（LSI 或称 LSA）

奇异值分解的数学表达

- $N \times M$矩阵X，把每一行当成一个点。设$r = rank(X)$。
- $X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T = \begin{bmatrix} | & & | \\ \vec{u}_1 & \cdots & \vec{u}_r \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \begin{bmatrix} -\vec{v}_1^T - \\ \cdots \\ -\vec{v}_r^T \end{bmatrix} = UDV^T$
- U, D, V分别为$N \times r, r \times r, M \times r$矩阵
  - 特征空间变换
- $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$称为奇异值
  - 其中：$\vec{u}_i \vec{v}_i^T$ 要满足归一化和正交化的要求

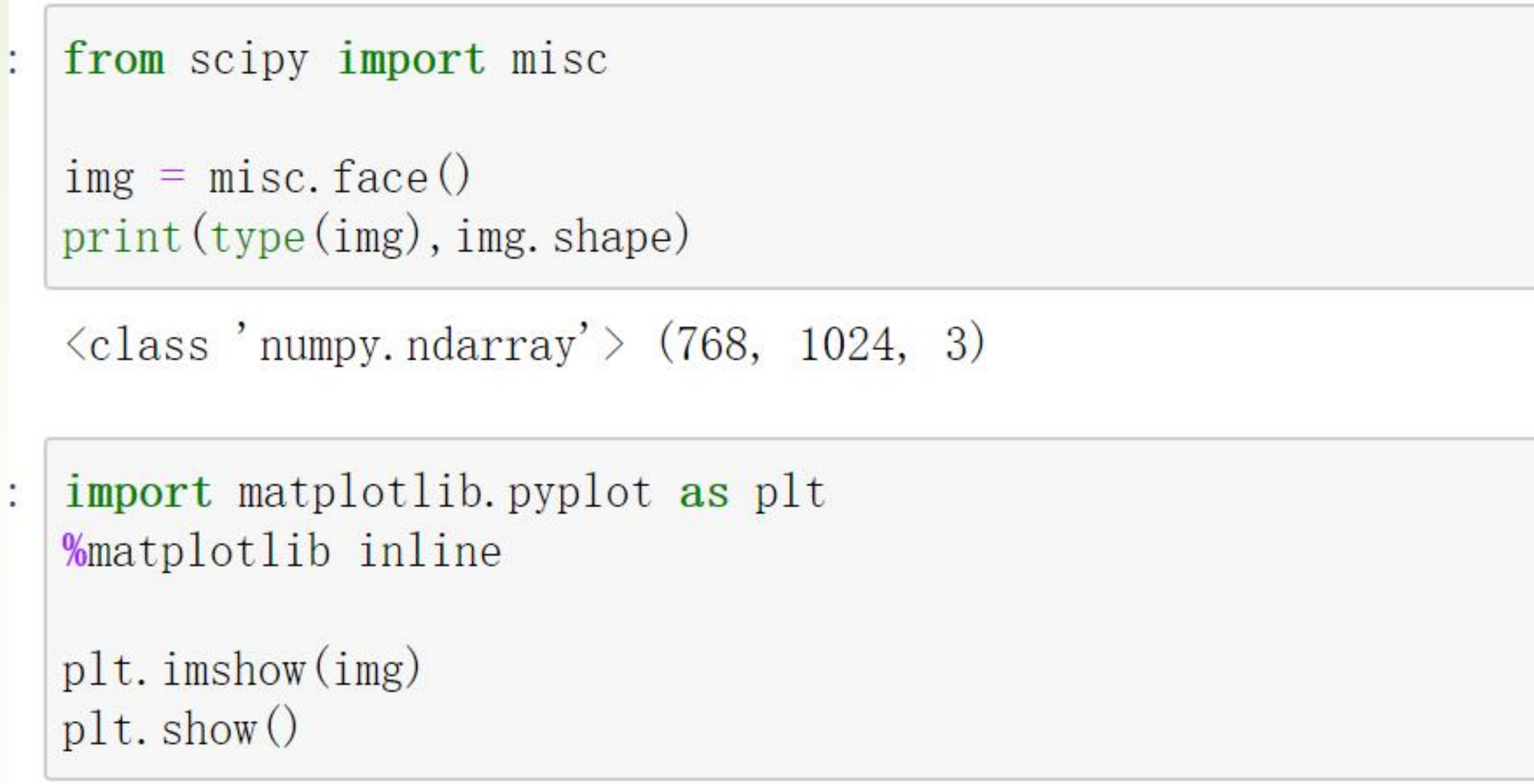
奇异值分解的基本求解流程

- $N \times M$矩阵X，把每一行当成一个点。设$r = rank(X)$。
- $\vec{v}_1 = argmax_{|\vec{v}|=1} |X\vec{v}|$
- $\vec{v}_2 = argmax_{|\vec{v}|=1, \vec{v}\perp\vec{v}_1} |X\vec{v}|$ ⟸ 保证正交的前提下的下一个可最大拉伸的方向
- ……
- $\vec{v}_r = argmax_{|\vec{v}|=1, \vec{v}\perp\vec{v}_1, \vec{v}\perp\vec{v}_2 \ldots, \vec{v}\perp\vec{v}_{r-1}} |X\vec{v}|$
- 令$\sigma_i = |X\vec{v}_i|, \overrightarrow{u_i} = \frac{1}{\sigma_i} X\vec{v}_i, 1 \le i \le r$
- 则$X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T = \begin{bmatrix} | & & | \\ \vec{u}_1 & \ldots & \vec{u}_r \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \begin{bmatrix} -\vec{v}_1^T - \\ \ldots \\ -\vec{v}_r^T \end{bmatrix} = UDV^T$

奇异值分解与降维：泛化与泛化损失

- $X = UDV^T$
- 按方差大小排列，得到第1,2,…, r个奇异值
- 奇异值小的几个主成分可以认为是不显著特征，将其丢弃——降维
- 则$X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T = \begin{bmatrix} | & & | \\ \vec{u}_1 & \dots & \vec{u}_r \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \boxed{\ddots \quad \\ \quad \sigma_r} \end{bmatrix} \begin{bmatrix} -\vec{v}_1^T - \\ \dots \\ -\vec{v}_r^T - \end{bmatrix} = UDV^T$

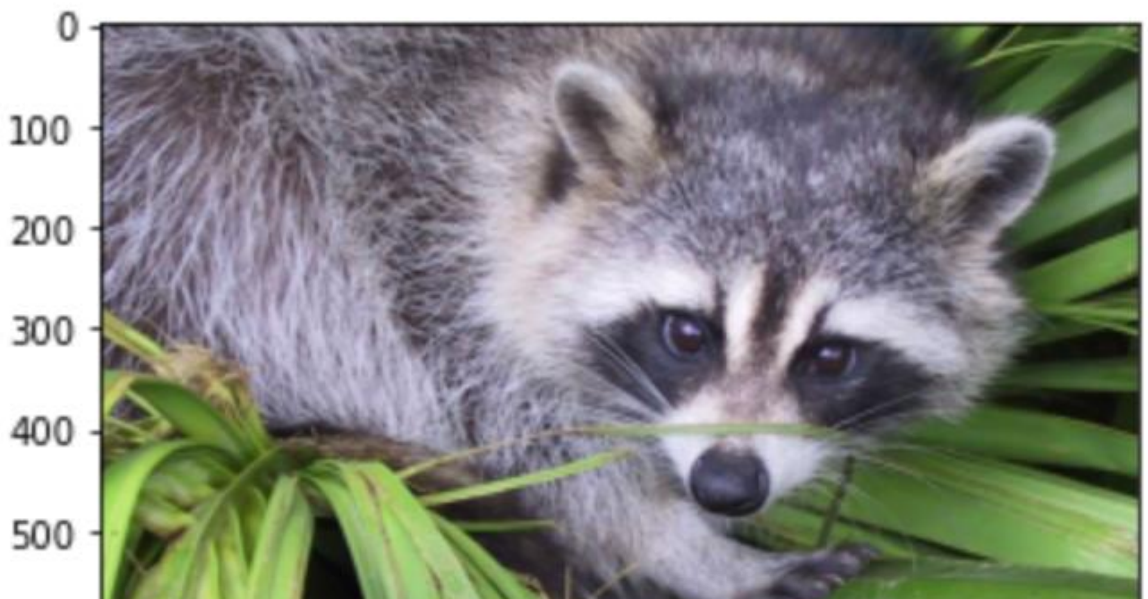
SVD图像降维的例子：Numpy Tutorials

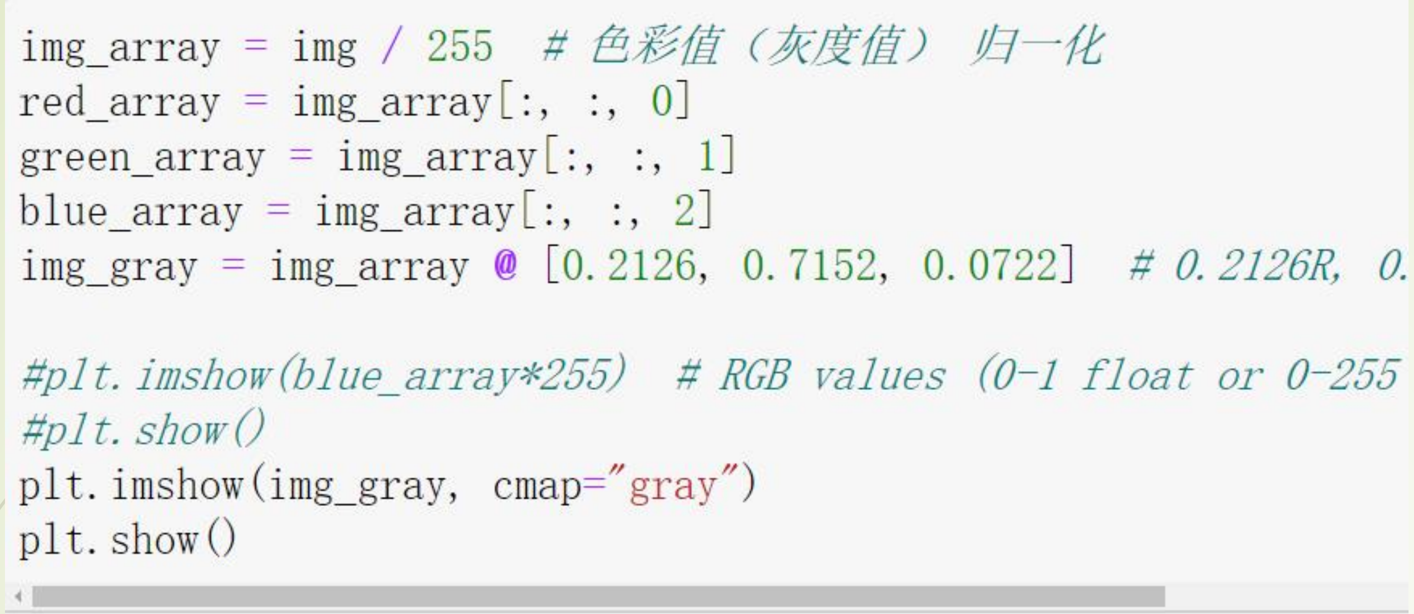


```
from scipy import misc

img = misc.face()
print(type(img), img.shape)
```

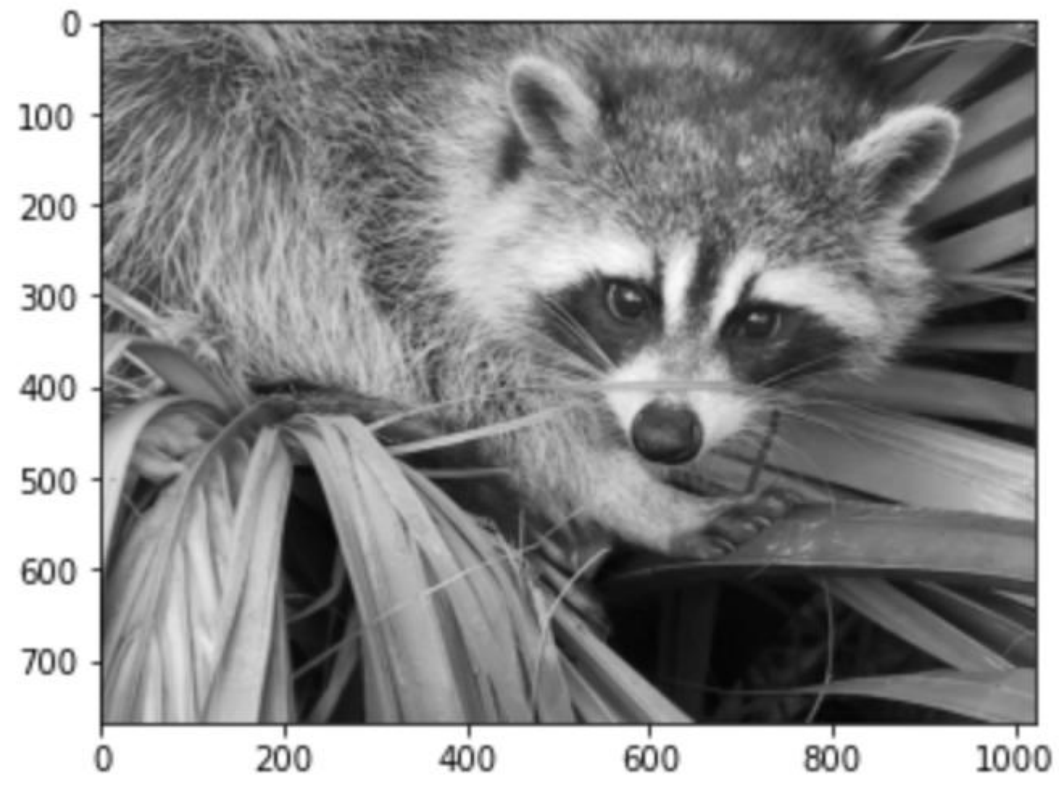
```
<class 'numpy.ndarray'> (768, 1024, 3)
```

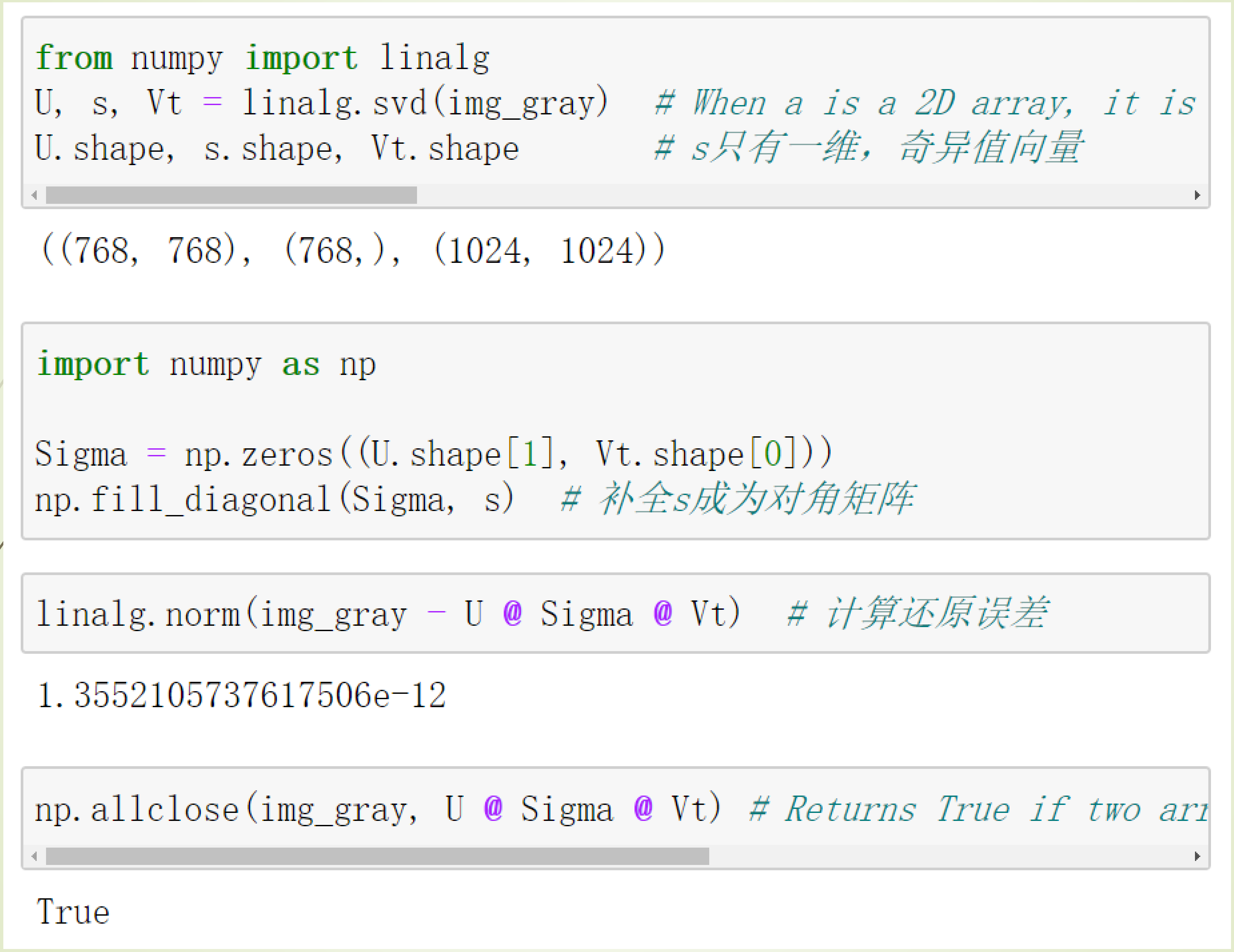
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.imshow(img)
plt.show()
```





```python
img_array = img / 255  # 色彩值（灰度值） 归一化
red_array = img_array[:, :, 0]
green_array = img_array[:, :, 1]
blue_array = img_array[:, :, 2]
img_gray = img_array @ [0.2126, 0.7152, 0.0722]  # 0.2126R, 0.

#plt.imshow(blue_array*255)  # RGB values (0-1 float or 0-255
#plt.show()
plt.imshow(img_gray, cmap="gray")
plt.show()
```





```python
from numpy import linalg
U, s, Vt = linalg.svd(img_gray)  # When a is a 2D array, it is
U.shape, s.shape, Vt.shape       # s只有一维，奇异值向量
```

```
((768, 768), (768,), (1024, 1024))
```

```python
import numpy as np

Sigma = np.zeros((U.shape[1], Vt.shape[0]))
np.fill_diagonal(Sigma, s)  # 补全s成为对角矩阵
```

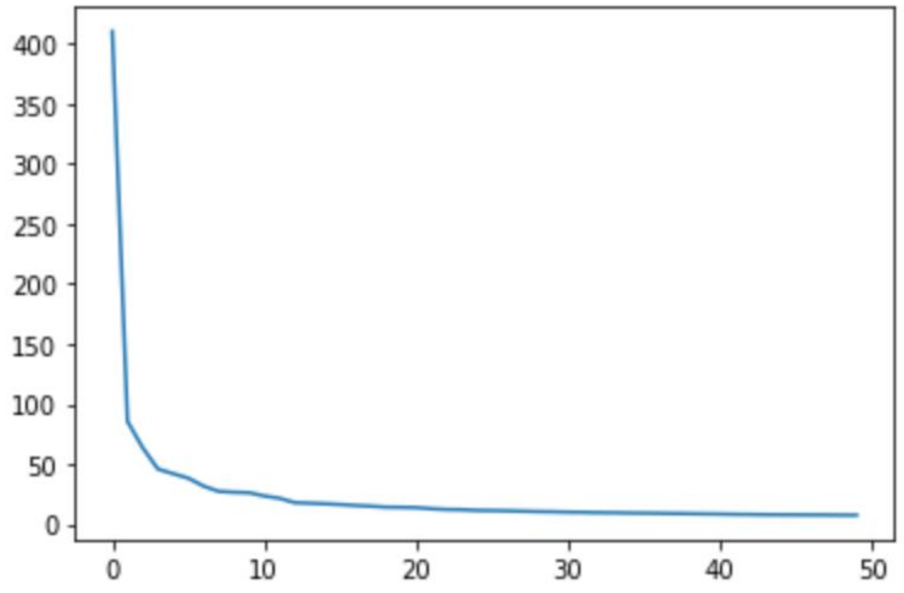
```python
linalg.norm(img_gray - U @ Sigma @ Vt)  # 计算还原误差
```

```
1.355210573761750 6e-12
```

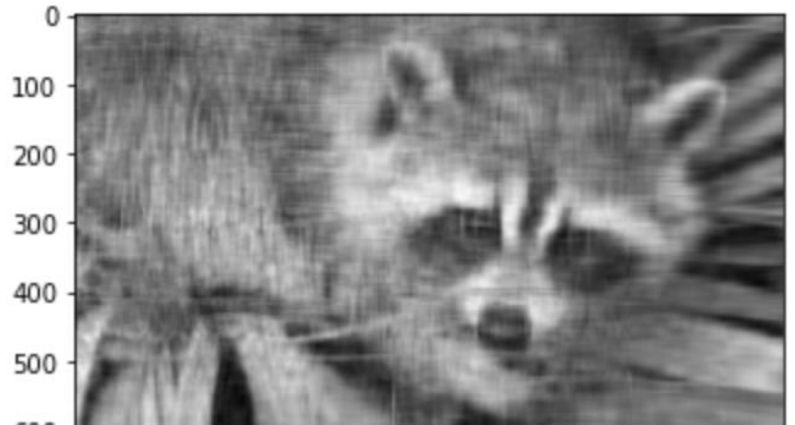
```python
np.allclose(img_gray, U @ Sigma @ Vt) # Returns True if two arr
```

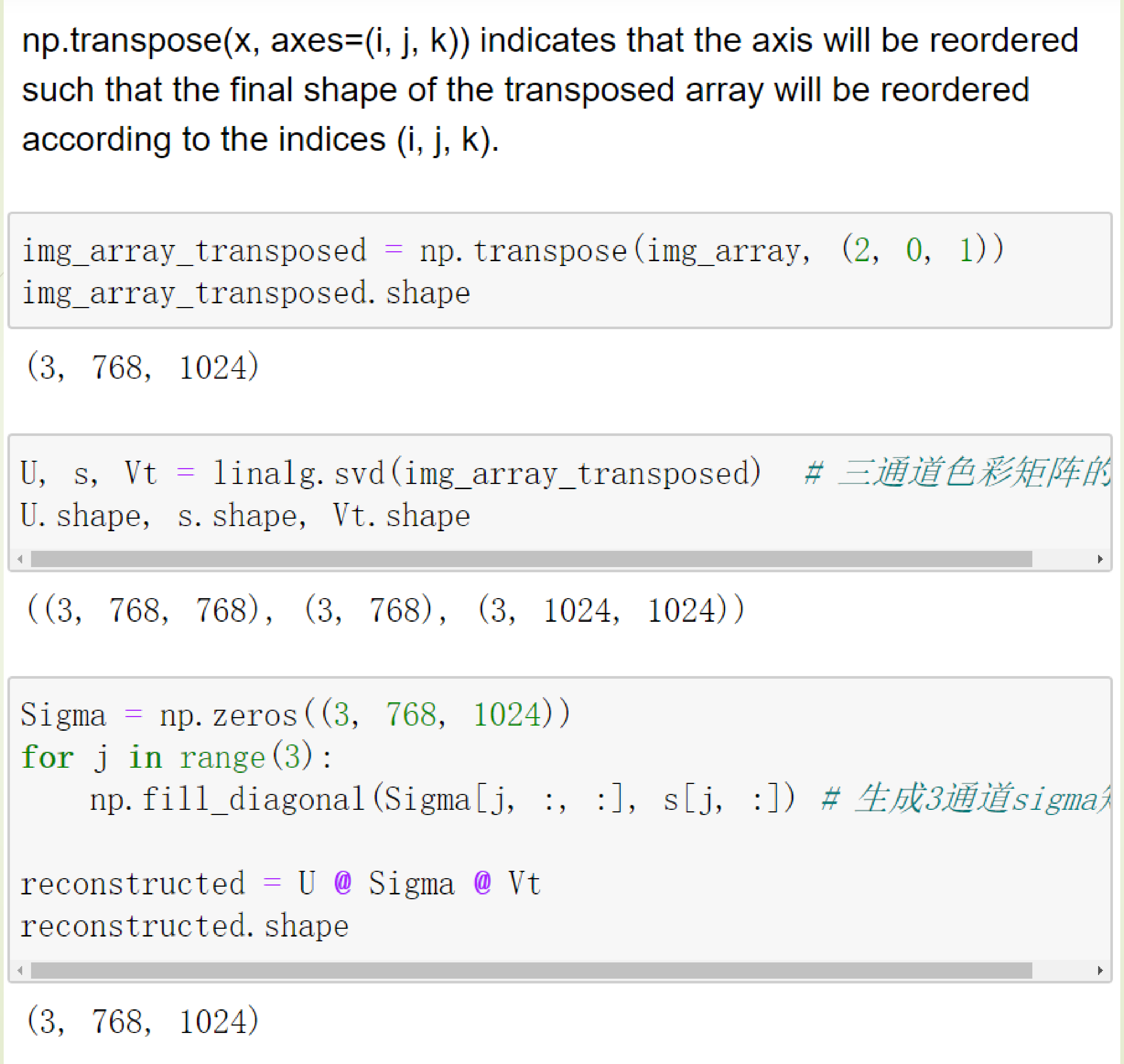
```
True
```

```
plt.plot(s[:50])  # 观察一下前50个奇异值的取值分布情况（能量占
plt.show()
```



```
k = 20
approx = U @ Sigma[:, :k] @ Vt[:k, :]
plt.imshow(approx, cmap="gray")
plt.show()
```





np.transpose(x, axes=(i, j, k)) indicates that the axis will be reordered such that the final shape of the transposed array will be reordered according to the indices (i, j, k).

```
img_array_transposed = np.transpose(img_array, (2, 0, 1))
img_array_transposed.shape
```

```
(3, 768, 1024)
```

```
U, s, Vt = linalg.svd(img_array_transposed)  # 三通道色彩矩阵的
U.shape, s.shape, Vt.shape
```

```
((3, 768, 768), (3, 768), (3, 1024, 1024))
```

```
Sigma = np.zeros((3, 768, 1024))
for j in range(3):
    np.fill_diagonal(Sigma[j, :, :], s[j, :]) # 生成3通道sigma

reconstructed = U @ Sigma @ Vt
reconstructed.shape
```

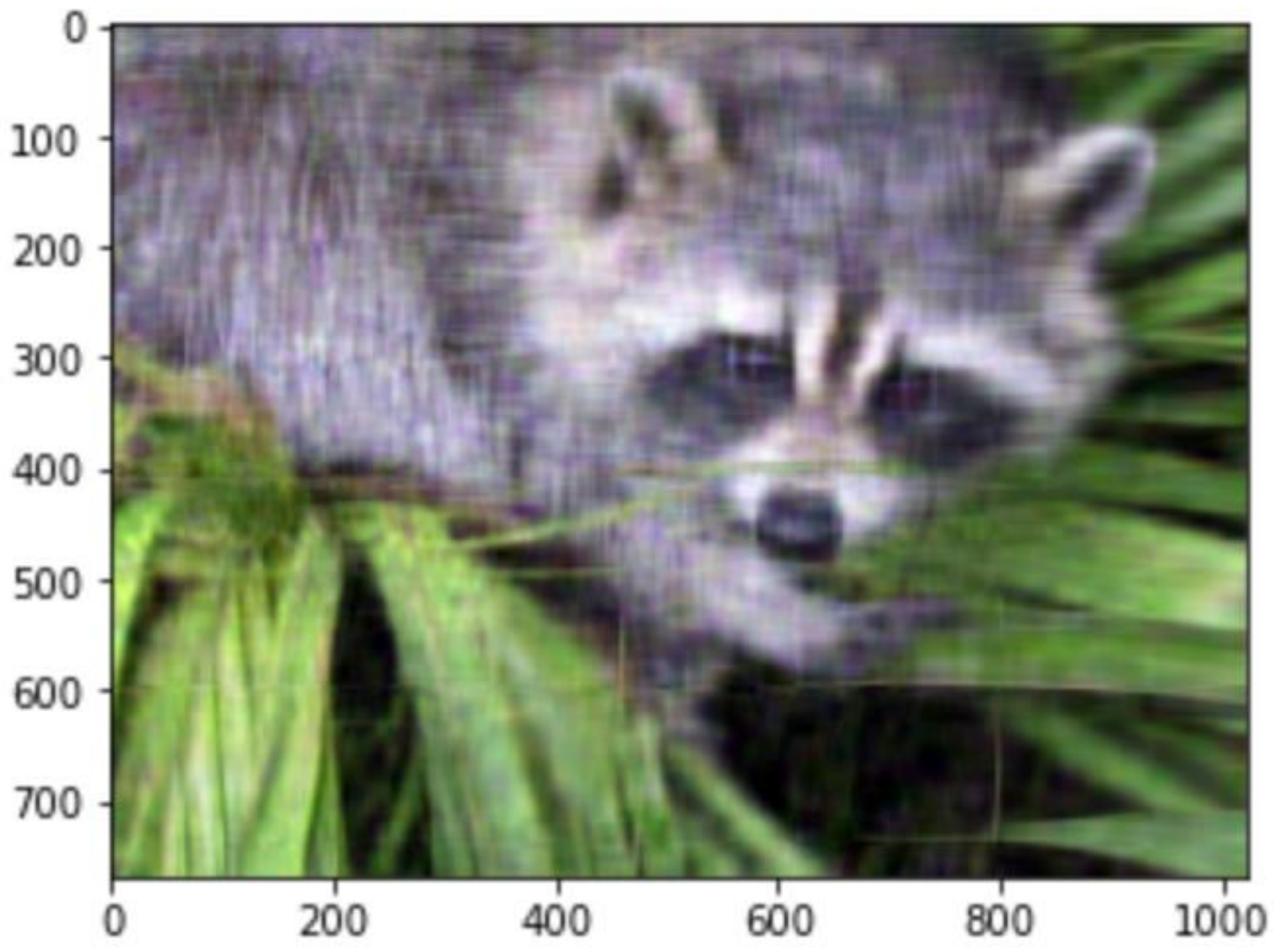
```
(3, 768, 1024)
```

```
reconstructed.min(), reconstructed.max()
```

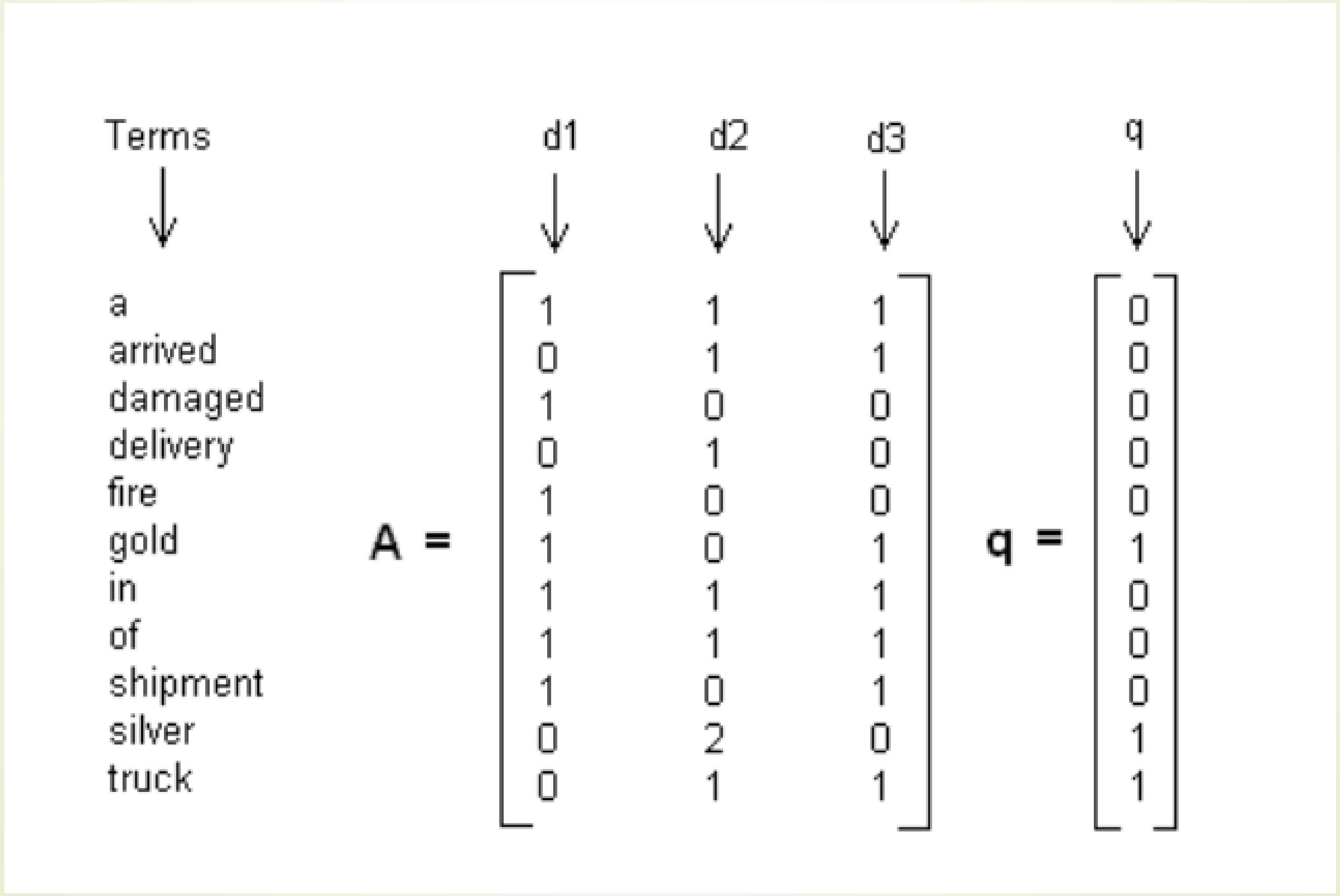
```
(-6.3056656250670695e-15, 1.000000000000004)
```

```
reconstructed = np.clip(reconstructed, 0, 1)          # 裁剪掉负
plt.imshow(np.transpose(reconstructed, (1, 2, 0)))    # 恢复原形
plt.show()
```

```
approx_img = U @ Sigma[..., :k] @ Vt[..., :k, :]
plt.imshow(np.clip(np.transpose(approx_img, (1, 2, 0)),0,1))
plt.show()
```



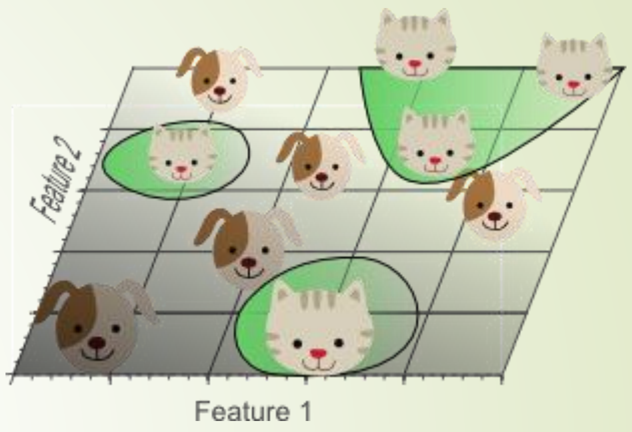
Latent Semantic Indexing (LSI)

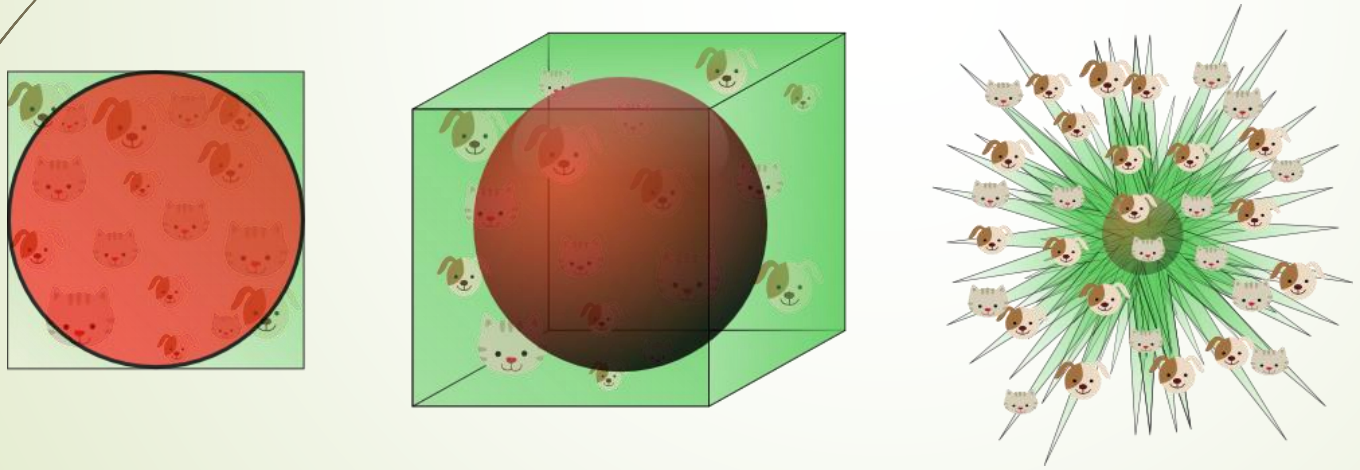


$$
\begin{array}{l}
\text{Terms} \\
\downarrow \\
\text{a} \\
\text{arrived} \\
\text{damaged} \\
\text{delivery} \\
\text{fire} \\
\text{gold} \\
\text{in} \\
\text{of} \\
\text{shipment} \\
\text{silver} \\
\text{truck}
\end{array}
\qquad
\mathbf{A} =
\begin{array}{ccc}
d1 & d2 & d3 \\
\downarrow & \downarrow & \downarrow \\
1 & 1 & 1 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 0 & 1 \\
0 & 2 & 0 \\
0 & 1 & 1
\end{array}
\qquad
\mathbf{q} =
\begin{array}{c}
q \\
\downarrow \\
0 \\
0 \\
0 \\
0 \\
0 \\
1 \\
0 \\
0 \\
0 \\
1 \\
1
\end{array}
$$

维度灾难与PCA

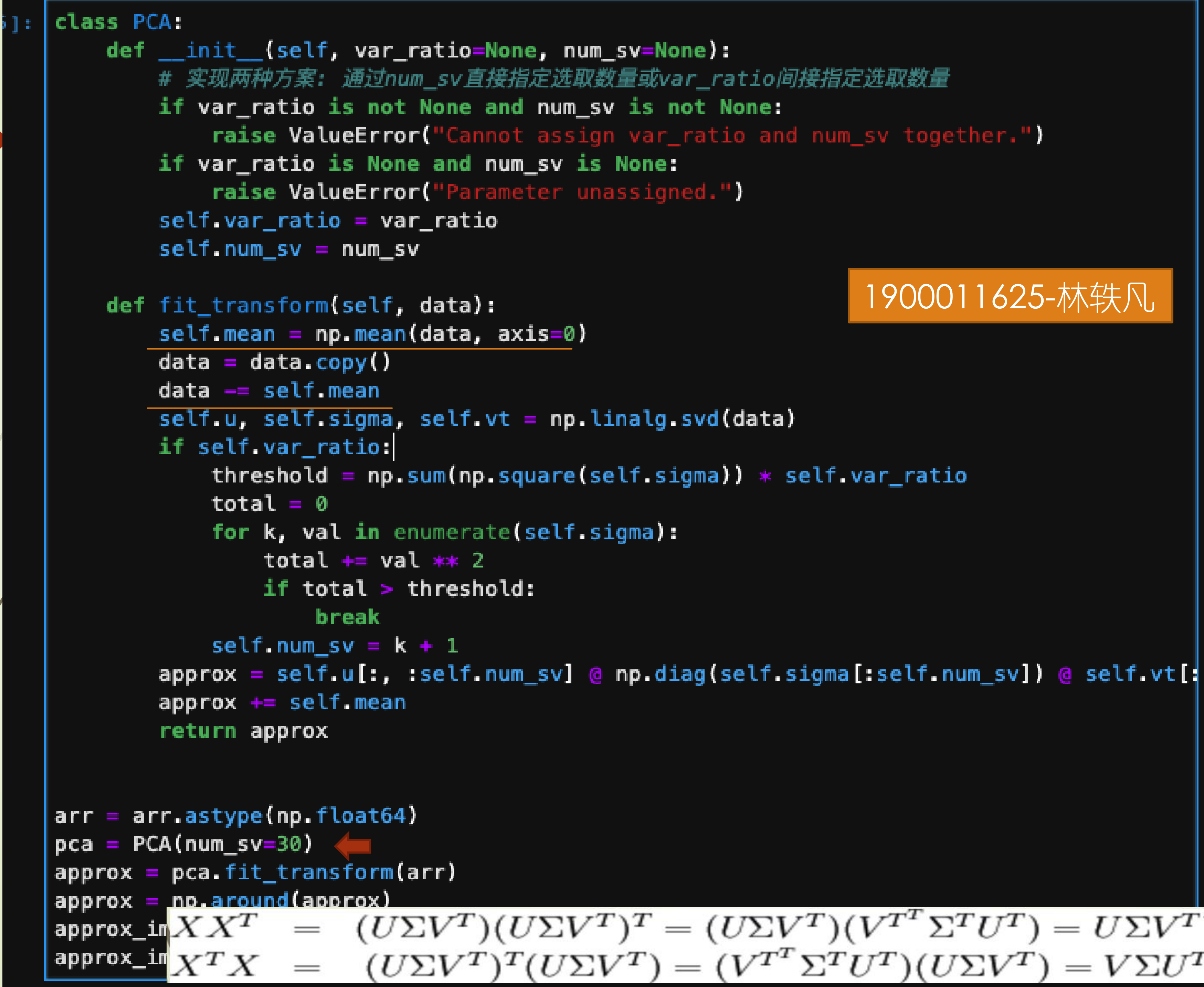
1. 高维度下，数据样本稀疏，难以做到密采样，易过拟合
2. 在高维空间，特征间的某些距离测量逐渐失效





PCA与SVD

1900011625-林轶凡

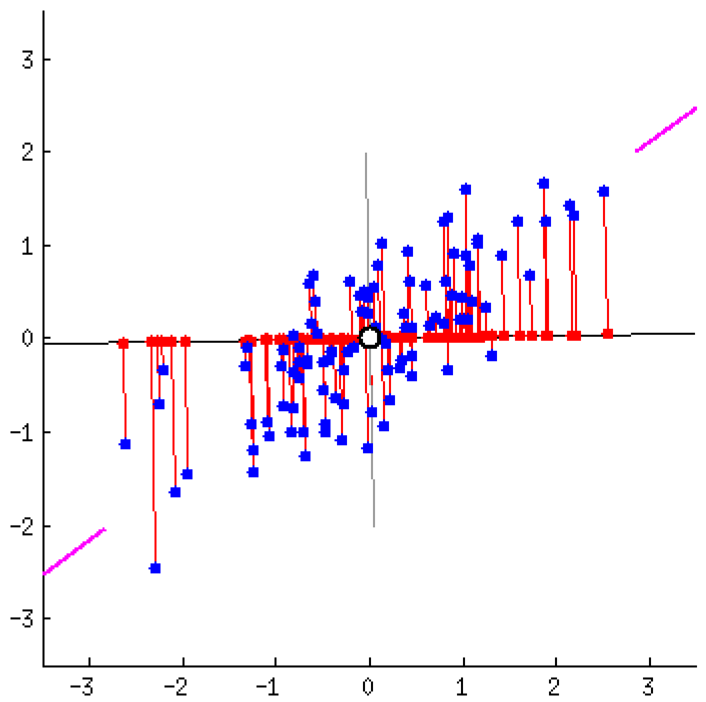


```
class PCA:
    def __init__(self, var_ratio=None, num_sv=None):
        # 实现两种方案: 通过num_sv直接指定选取数量或var_ratio间接指定选取数量
        if var_ratio is not None and num_sv is not None:
            raise ValueError("Cannot assign var_ratio and num_sv together.")
        if var_ratio is None and num_sv is None:
            raise ValueError("Parameter unassigned.")
        self.var_ratio = var_ratio
        self.num_sv = num_sv

    def fit_transform(self, data):
        self.mean = np.mean(data, axis=0)
        data = data.copy()
        data -= self.mean
        self.u, self.sigma, self.vt = np.linalg.svd(data)
        if self.var_ratio:
            threshold = np.sum(np.square(self.sigma)) * self.var_ratio
            total = 0
            for k, val in enumerate(self.sigma):
                total += val ** 2
                if total > threshold:
                    break
            self.num_sv = k + 1
        approx = self.u[:, :self.num_sv] @ np.diag(self.sigma[:self.num_sv]) @ self.vt[:
        approx += self.mean
        return approx


arr = arr.astype(np.float64)
pca = PCA(num_sv=30)
approx = pca.fit_transform(arr)
approx = np.around(approx)
approx_in
approx_in
```

$$X X^T = (U\Sigma V^T)(U\Sigma V^T)^T = (U\Sigma V^T)(V^{T^T}\Sigma^T U^T) = U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T$$
$$X^T X = (U\Sigma V^T)^T(U\Sigma V^T) = (V^{T^T}\Sigma^T U^T)(U\Sigma V^T) = V\Sigma U^T U\Sigma V^T = V\Sigma^T\Sigma V^T$$

PCA-主成分分析

1. 最近重构性
2. 最大可分性



PCA-主成分分析

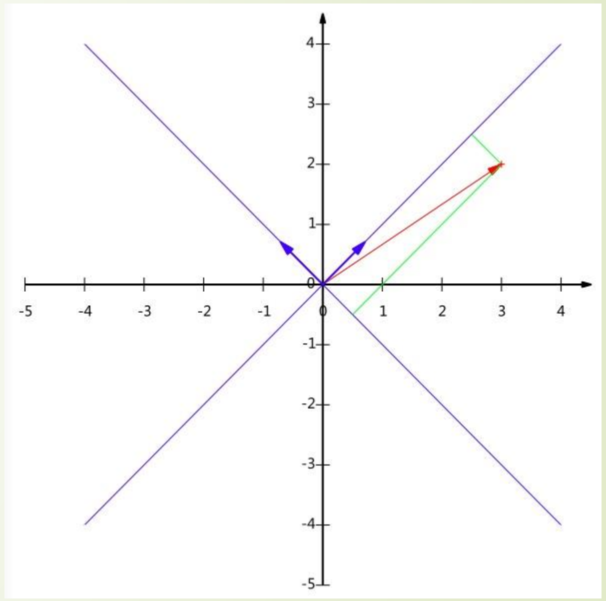
内积与投影

$$A \cdot B = |A||B|cos(\alpha)$$

$$A \cdot B = |A|cos(\alpha)$$

特征空间的基向量与基变换

$$\begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 5/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix}$$



$$\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{pmatrix} \begin{pmatrix} a_1 & a_2 & \cdots & a_M \end{pmatrix} = \begin{pmatrix} p_1a_1 & p_1a_2 & \cdots & p_1a_M \\ p_2a_1 & p_2a_2 & \cdots & p_2a_M \\ \vdots & \vdots & \ddots & \vdots \\ p_Ra_1 & p_Ra_2 & \cdots & p_Ra_M \end{pmatrix}$$

其中 p_i 是一个行向量，表示第 i 个基，a_j 是一个列向量，表示第 j 个原始数据记录

PCA-主成分分析

投影后数据尽量分散，数据的分散程度可以用方差衡量

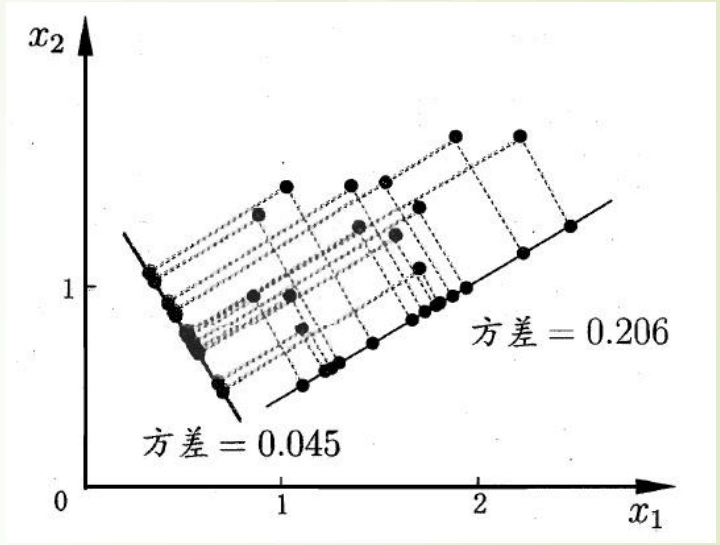
$$Var(a) = \frac{1}{m}\sum_{i=1}^{m}(a_i - \mu)^2$$

$$Var(a) = \frac{1}{m}\sum_{i=1}^{m}{a_i}^2$$

对于高维变换，我们希望每个字段尽可能表达更多的信息，不存在相关性，可以用两个字段的协方差衡量

$$Cov(a,b) = \frac{1}{m}\sum_{i=1}^{m}a_i b_i$$

协方差为0，表示两个属性字段不相关。



PCA-主成分分析

- 对原始样本进行（线性变换）基变换可以对原始样本给出不同的表示
- 基的维度小于数据的维度可以起到降维的效果
- 对基变换后新的样本求其方差，选取使其方差最大的基
- 新的基之间的协方差要为0

$$X = \begin{pmatrix} a_1 & a_2 & \dots & a_m \\ b_1 & b_2 & \dots & b_m \end{pmatrix}$$

协方差矩阵

$$\frac{1}{m} X X^\top = \begin{pmatrix} \frac{1}{m}\sum_{i=1}^{m} a_i^2 & \frac{1}{m}\sum_{i=1}^{m} a_i b_i \\ \frac{1}{m}\sum_{i=1}^{m} a_i b_i & \frac{1}{m}\sum_{i=1}^{m} b_i^2 \end{pmatrix}$$

目标：基变换后的新样本的协方差矩阵是对角化的

PCA-主成分分析

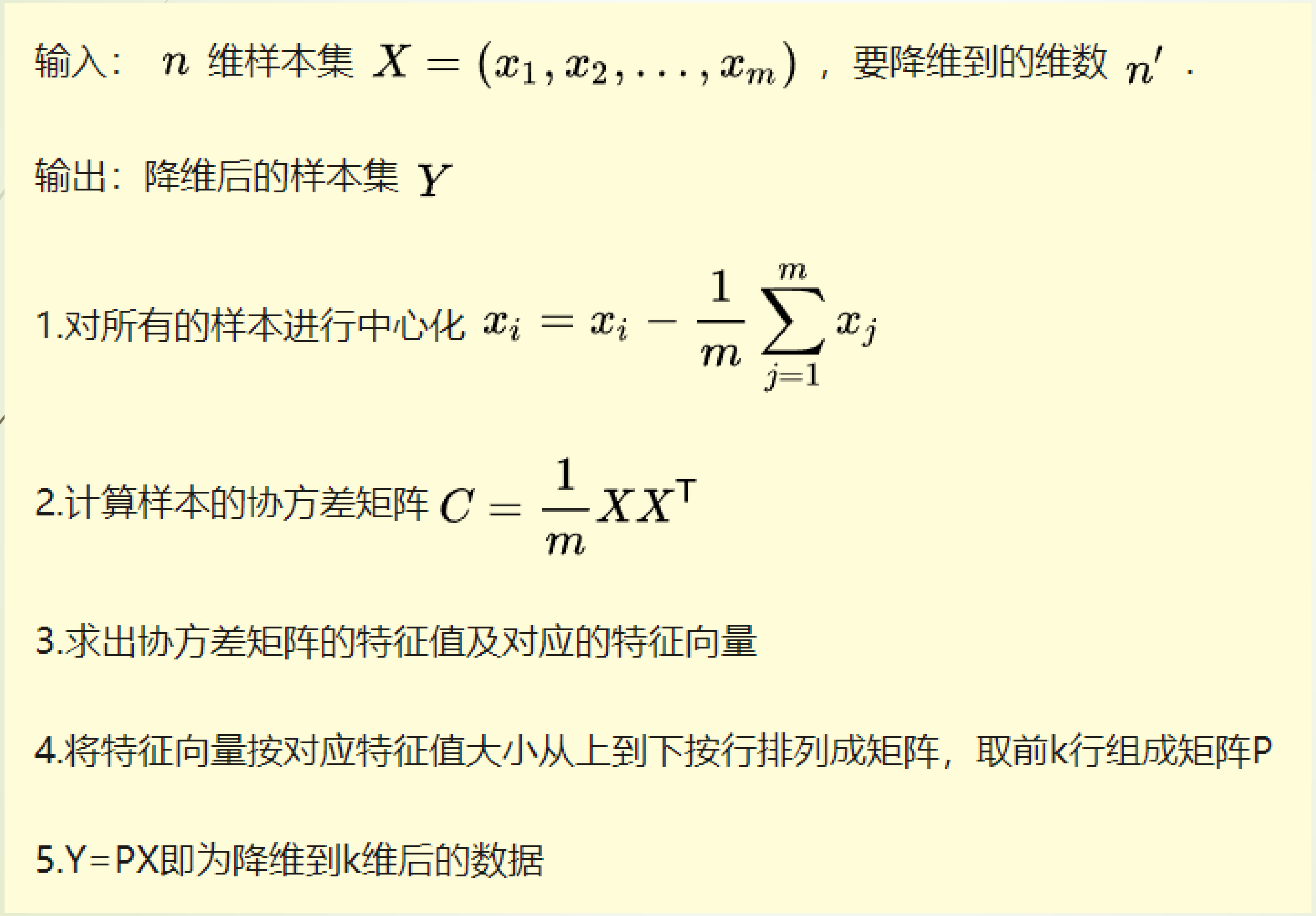
X：原始数据，每一列为一个数据
P：是一组基按行组成的矩阵
Y：是X对P做基变换的后的新数据
C：原始数据X的协方差矩阵
D：Y的协方差矩阵

我们找的P，正好是让原协方差矩阵对角化的P，将对角元素排列，对应的前K行就是要寻找的K个基，用P的前K行乘以X，就得到了新的数据表示，并将数据从N维降到了K维。

$$
\begin{aligned}
D &= \frac{1}{m}YY^\top \\
&= \frac{1}{m}(PX)(PX)^\top \\
&= \frac{1}{m}PXX^\top P^\top \\
&= P(\frac{1}{m}XX^\top)P^\top \\
&= PCP^\top \\
&= P\begin{pmatrix} \frac{1}{m}\sum_{i=1}^m a_i^2 & \frac{1}{m}\sum_{i=1}^m a_i b_i \\ \frac{1}{m}\sum_{i=1}^m a_i b_i & \frac{1}{m}\sum_{i=1}^m b_i^2 \end{pmatrix} P^\top
\end{aligned}
$$

于是，只需对协方差矩阵C进行特征分解，对求得的特征值进行排序，再对特征向量取前K列组成的矩阵乘以原始数据矩阵X，就得到了我们需要的降维后的数据矩阵Y。

PCA-主成分分析



输入： n 维样本集 $X=(x_1,x_2,\ldots,x_m)$ ，要降维到的维数 n' .

输出：降维后的样本集 Y

1.对所有的样本进行中心化 $x_i=x_i-\frac{1}{m}\sum_{j=1}^{m}x_j$

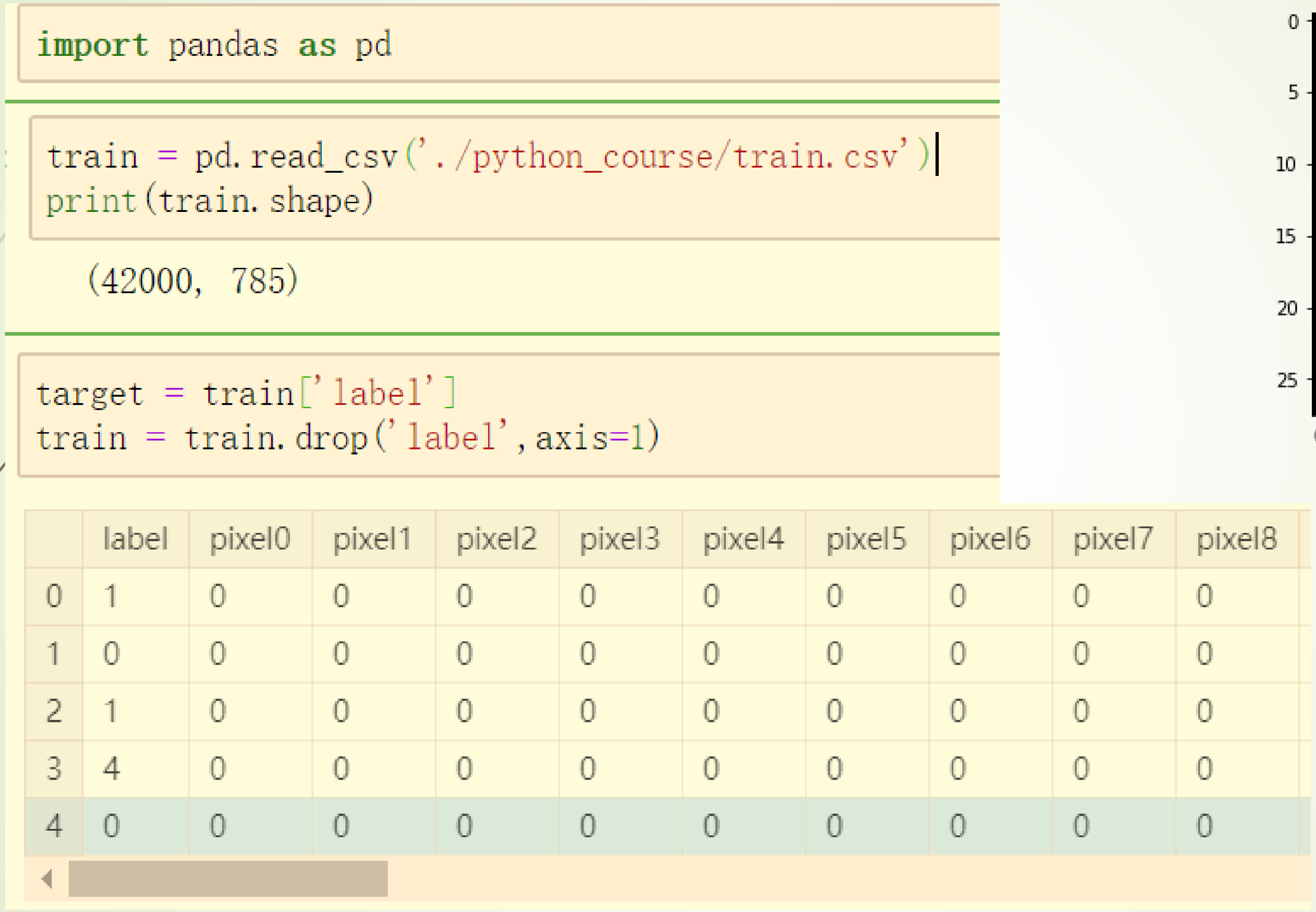
2.计算样本的协方差矩阵 $C=\frac{1}{m}XX^\mathsf{T}$

3.求出协方差矩阵的特征值及对应的特征向量

4.将特征向量按对应特征值大小从上到下按行排列成矩阵，取前k行组成矩阵P

5.Y=PX即为降维到k维后的数据

PCA-主成分分析



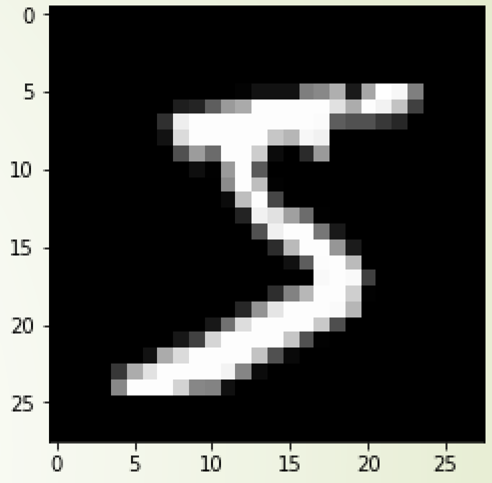
```
import pandas as pd
```

```
train = pd.read_csv('./python_course/train.csv')
print(train.shape)
```

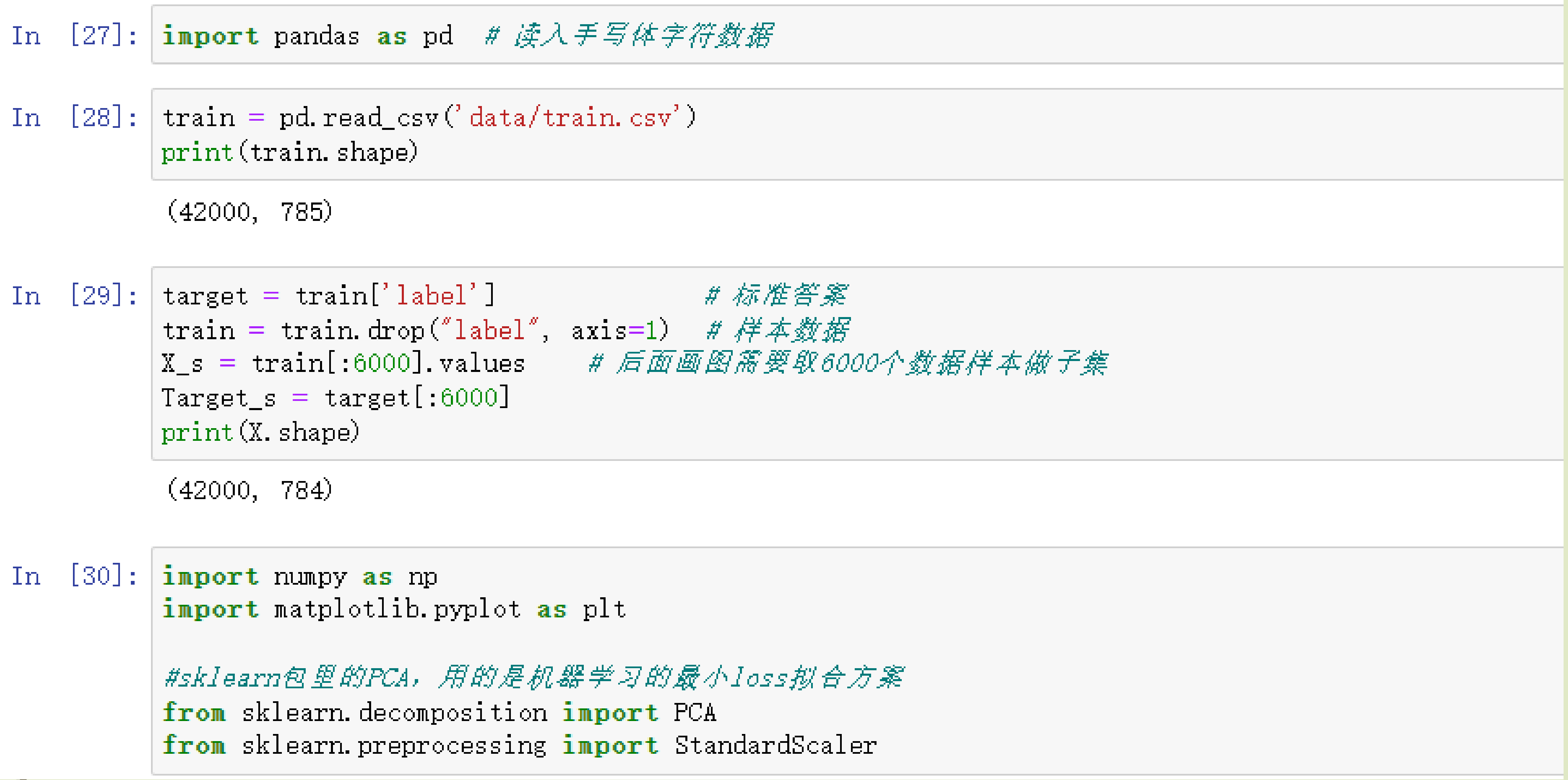
```
(42000, 785)
```

```
target = train['label']
train = train.drop('label',axis=1)
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8
0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0



PCA数据降维与聚类算法-可视化



```
In [27]: import pandas as pd  # 读入手写体字符数据
```

```
In [28]: train = pd.read_csv('data/train.csv')
         print(train.shape)
```

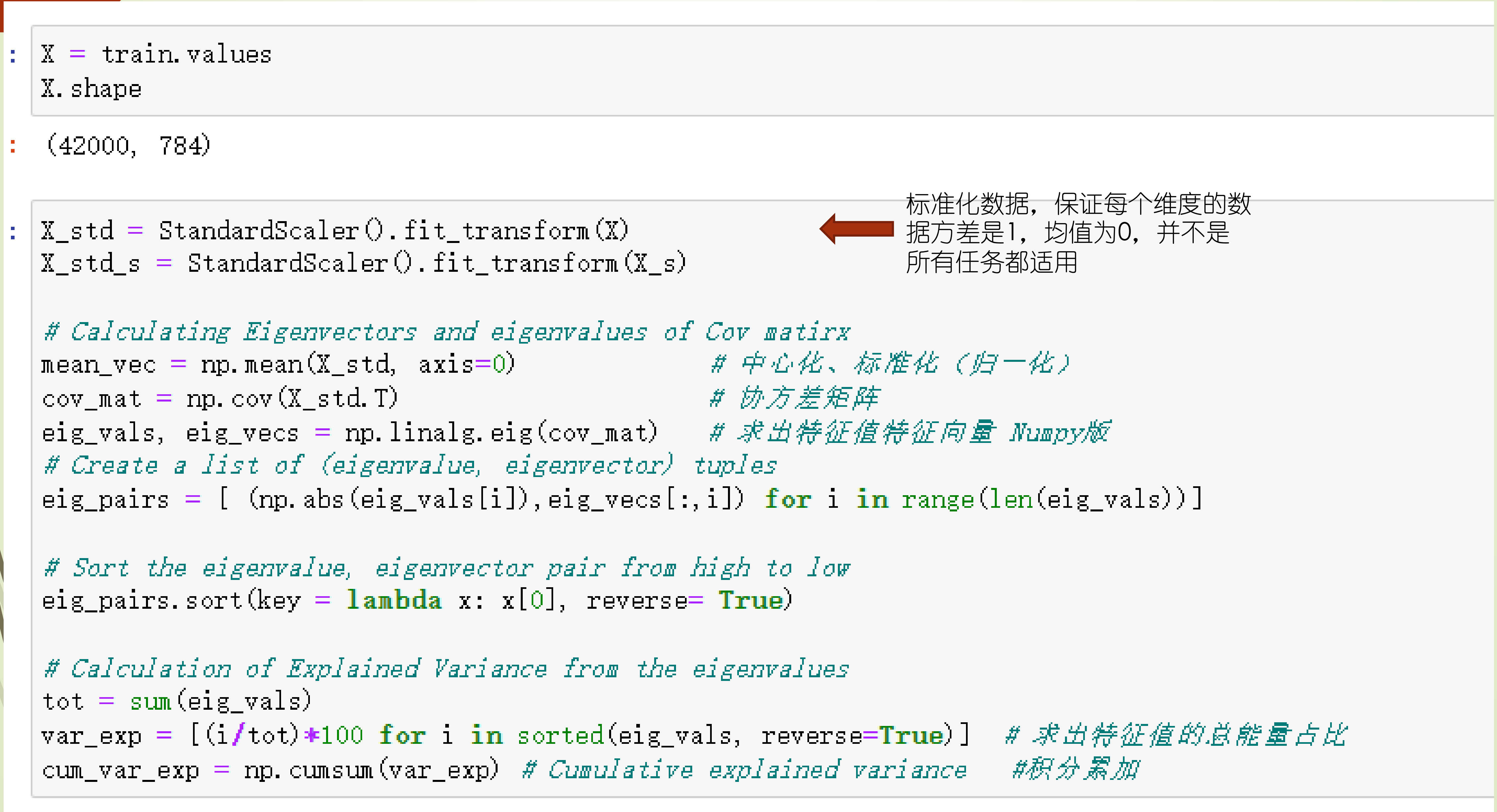
```
(42000, 785)
```

```
In [29]: target = train['label']                    # 标准答案
         train = train.drop("label", axis=1)  # 样本数据
         X_s = train[:6000].values     # 后面画图需要取6000个数据样本做子集
         Target_s = target[:6000]
         print(X.shape)
```

```
(42000, 784)
```

```
In [30]: import numpy as np
         import matplotlib.pyplot as plt

         #sklearn包里的PCA，用的是机器学习的最小loss拟合方案
         from sklearn.decomposition import PCA
         from sklearn.preprocessing import StandardScaler
```

用Numpy实现PCA



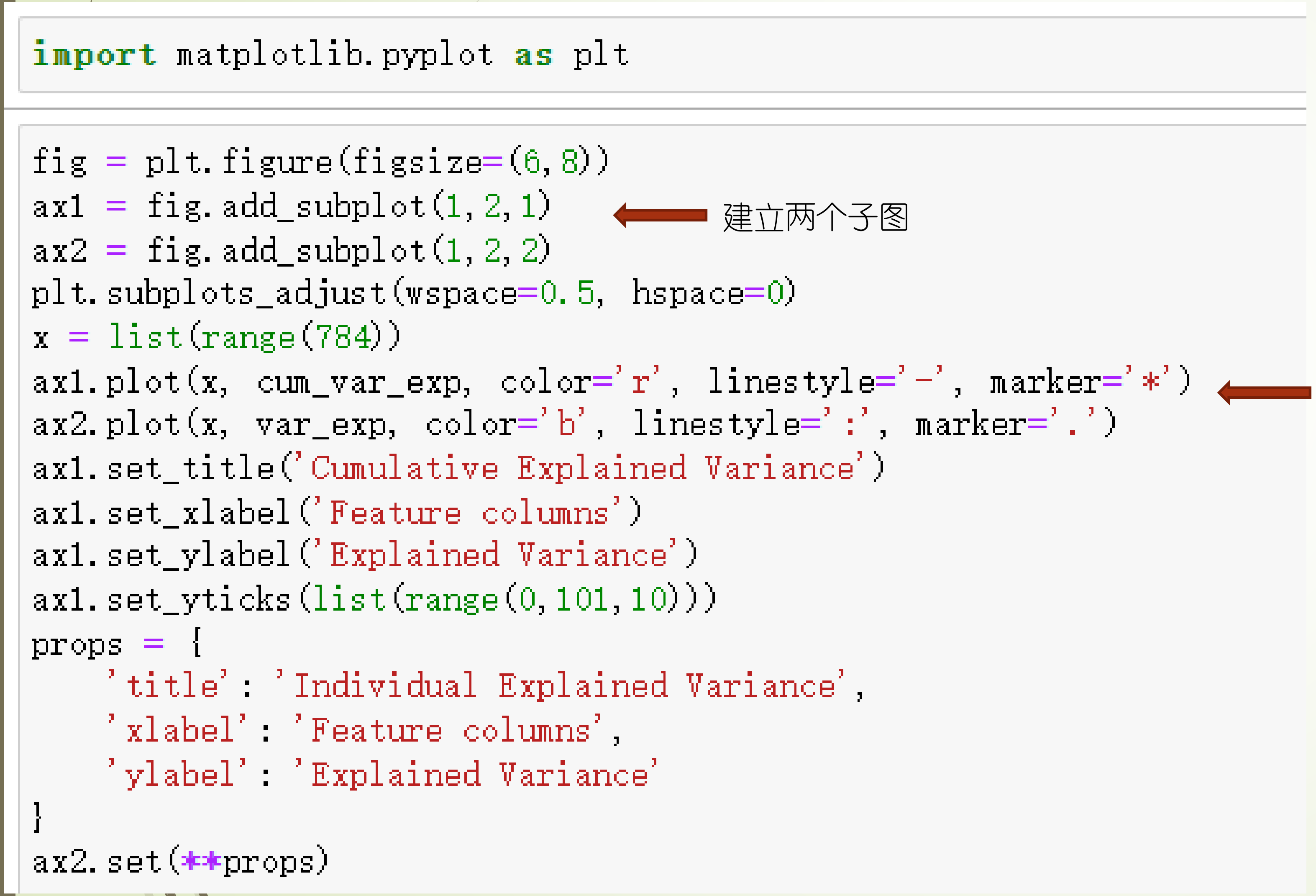
```
X = train.values
X.shape
```

```
(42000, 784)
```

```
X_std = StandardScaler().fit_transform(X)
X_std_s = StandardScaler().fit_transform(X_s)

# Calculating Eigenvectors and eigenvalues of Cov matirx
mean_vec = np.mean(X_std, axis=0)                # 中心化、标准化（归一化）
cov_mat = np.cov(X_std.T)                        # 协方差矩阵
eig_vals, eig_vecs = np.linalg.eig(cov_mat)     # 求出特征值特征向量 Numpy版
# Create a list of (eigenvalue, eigenvector) tuples
eig_pairs = [ (np.abs(eig_vals[i]),eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the eigenvalue, eigenvector pair from high to low
eig_pairs.sort(key = lambda x: x[0], reverse= True)

# Calculation of Explained Variance from the eigenvalues
tot = sum(eig_vals)
var_exp = [(i/tot)*100 for i in sorted(eig_vals, reverse=True)]  # 求出特征值的总能量占比
cum_var_exp = np.cumsum(var_exp) # Cumulative explained variance   #积分累加
```

标准化数据，保证每个维度的数据方差是1，均值为0，并不是所有任务都适用

PCA-主成分分析

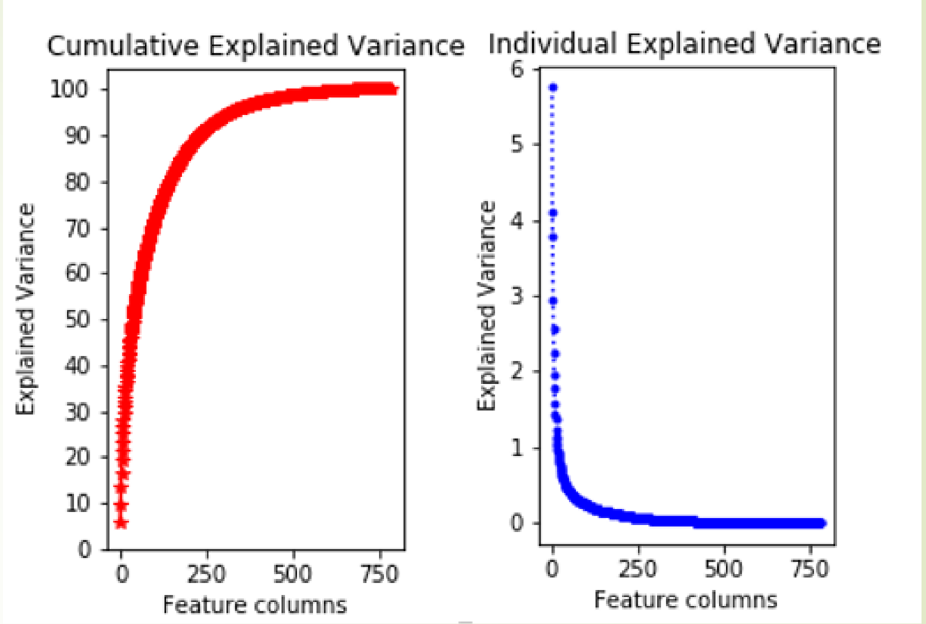


```
import matplotlib.pyplot as plt
```

```
fig = plt.figure(figsize=(6,8))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
plt.subplots_adjust(wspace=0.5, hspace=0)
x = list(range(784))
ax1.plot(x, cum_var_exp, color='r', linestyle='-', marker='*')
ax2.plot(x, var_exp, color='b', linestyle=':', marker='.')
ax1.set_title('Cumulative Explained Variance')
ax1.set_xlabel('Feature columns')
ax1.set_ylabel('Explained Variance')
ax1.set_yticks(list(range(0,101,10)))
props = {
    'title': 'Individual Explained Variance',
    'xlabel': 'Feature columns',
    'ylabel': 'Explained Variance'
}
ax2.set(**props)
```

建立两个子图

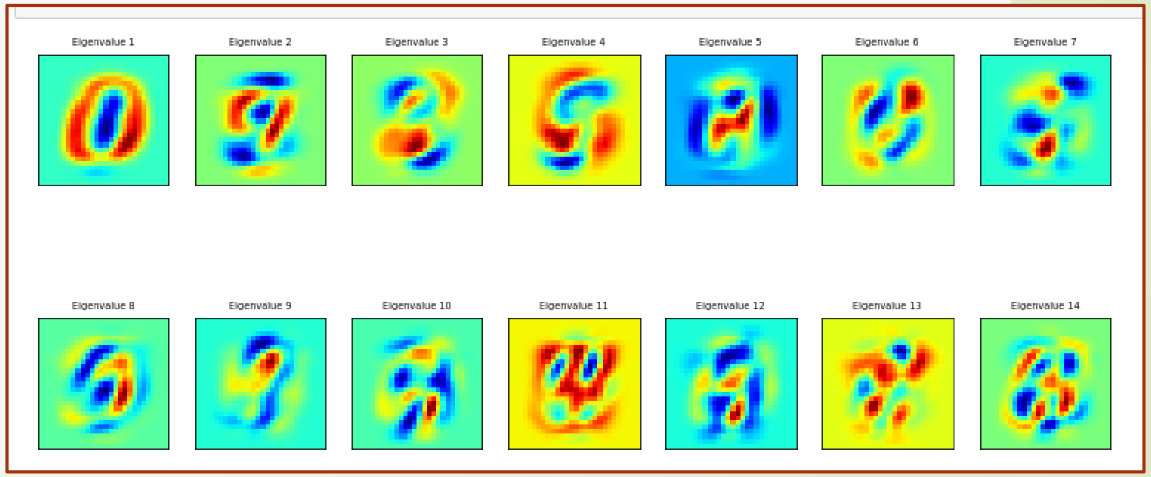
进行绘制，设置坐标轴，图名，刻度范围等



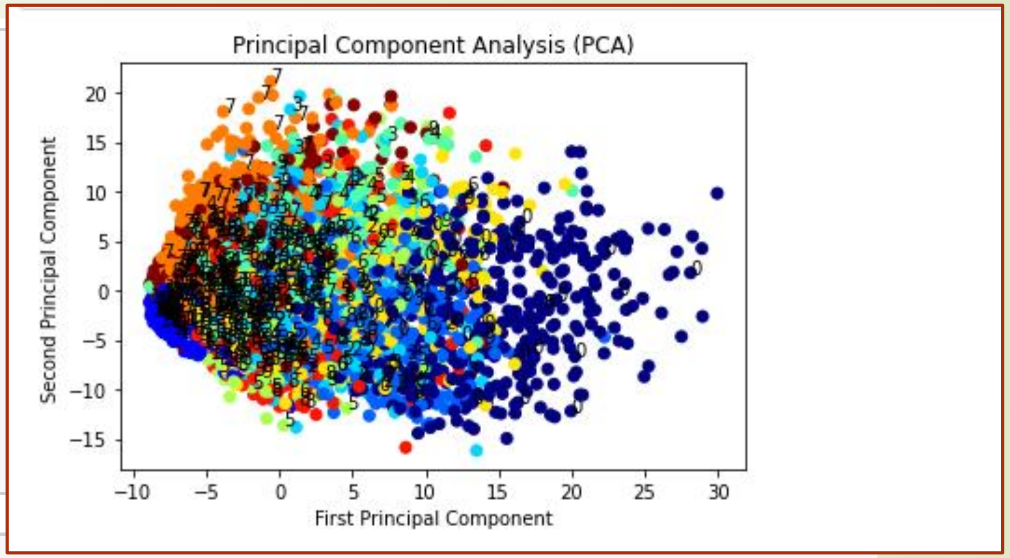
```python
# Invoke SKlearn's PCA method
n_components = 30
pca = PCA(n_components=n_components).fit(X)  # 拟合最小损失的30维PCA空间，SKlearn版

eigenvectors = pca.components_.reshape(n_components, 28, 28) # 把特征向量转成二维图片

# Extracting the PCA components ( eignevalues )
eigenvalues = pca.singular_values_
eigenvalues.shape
```

```
(30,)
```

```python
# Plot the first 2*7 eignenvectors
n_row = 2
n_col = 7

plt.figure(figsize=(13,6))   # 画框大小
for i in list(range(n_row * n_col)):
    offset = 0
    plt.subplot(n_row, n_col, i + 1) # 行列标定子图
    plt.imshow(eigenvectors[i].reshape(28,28), cmap='jet')
    title_text = 'Eigenvalue ' + str(i + 1)
    plt.title(title_text, size=6.5)
    plt.xticks(())
    plt.yticks(())
plt.show()
```

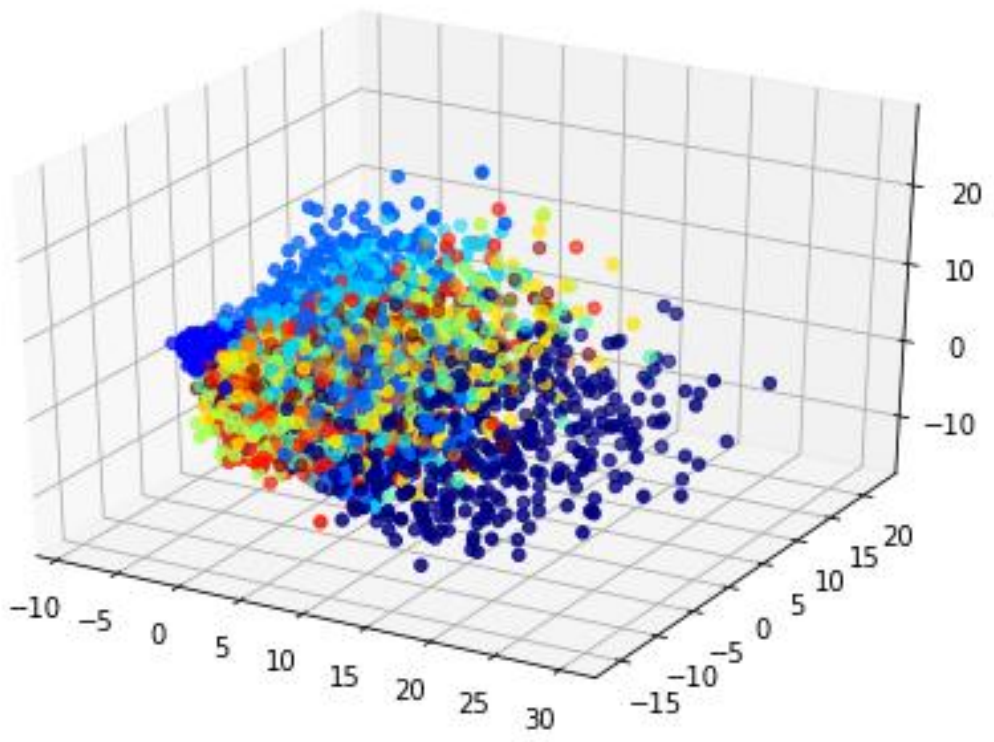


```
pca = PCA(n_components = 3)
pca.fit(X_std)     # 生成3维基底的特征空间

# 把原空间6000个子集的数据784维，投影到新3D特征空间中
# 这一步非常重要，PCA只是学到了一个基底
# 具体数据效果还是要把数据投影进去，可以思考Numpy下该如何实现？
X_3d = pca.transform(X_std_s)
# X_3d[1]
```



```
fig = plt.figure()
ax1 = fig.add_subplot(111)
#设置标题
ax1.set_title('Principal Component Analysis (PCA)')
#设置X轴标签
plt.xlabel('First Principal Component')
#设置Y轴标签
plt.ylabel('Second Principal Component')
# 这里只按前两维画散点，不同target颜色不同
ax1.scatter(X_3d[:,0],X_3d[:,1],c = Target, cmap='jet', marker = 'o')

for i in range(0,6000,10):     # 为了能看清楚，这里用了小样本的6000个数据
    ax1.annotate(str(Target[i]), (X_3d[i, 0], X_3d[i, 1])) # 根据实际标签显示不同数字
#设置图标
#显示所画的图
plt.show()
```

```
pca = PCA(n_components=3)
pca.fit(X_std)
X_3d = pca.transform(X_std)


from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = Axes3D(fig)    # 3维散点图

ax.scatter(X_3d[:, 0], X_3d[:, 1], X_3d[:, 2], c=Target,  cmap='jet', marker = 'o')
plt.show()
```



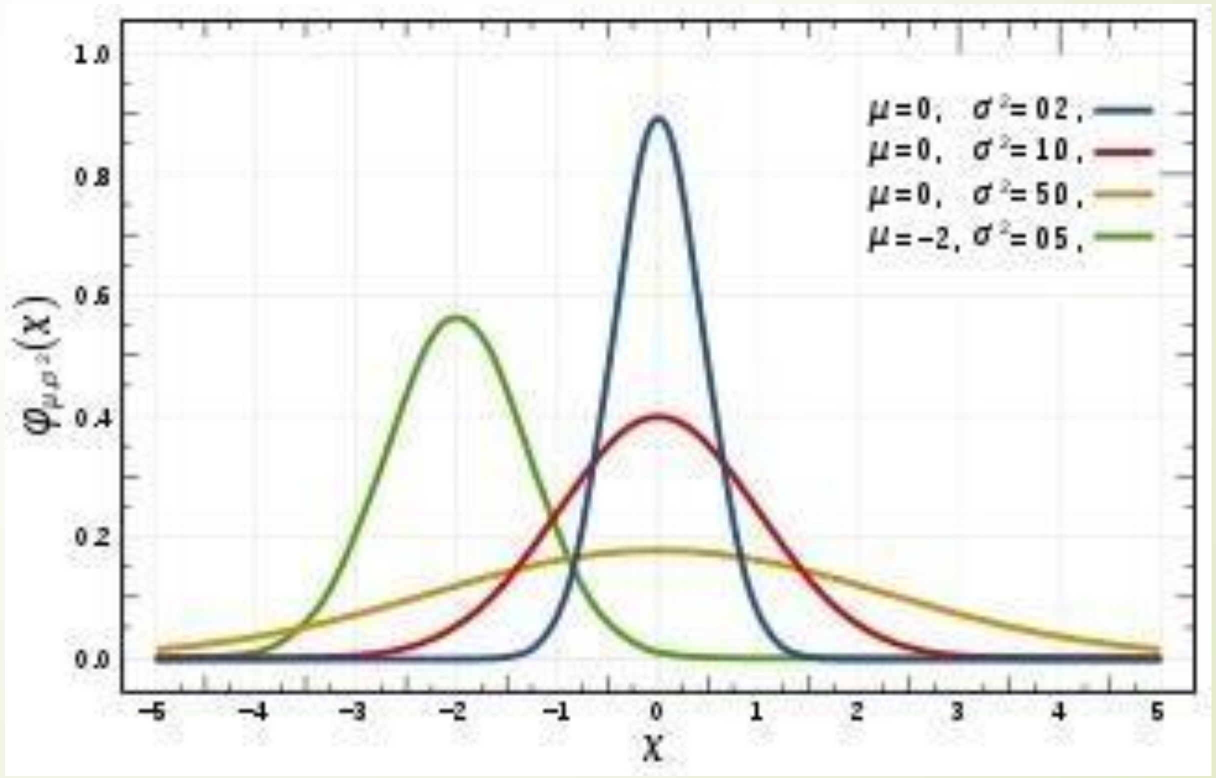
PCA的物理直观：

- PCA的目标：降维的同时保留大部分信息
  - 直观地理解：数据投影之后的值在特征轴上尽可能分散
  - 同时我们希望各个维度尽可能不相关
    - 因此自然引入协方差矩阵：方差尽可能大（协方差矩阵的对角线），协方差为0
  - 常用于同类型样本数集的降维

如何理解方差越大，该特征的信息量越大？

- 从量化表达误差的角度看

一个特征分布的方差越大，其信息量即区分度也就越高



特征分解与主成分分析的应用场景

- 都会要求产生一组正交基
- 主成分分析：把样本投影到新的特征空间
  - 特征工程（检索、聚类）、数据降噪、数据压缩
- 特征分解（SDV）：样本集被分解为 参数*特征 的线性表达
  - 文档话题模型（主题分析）
  - 隐含语义挖掘（词义泛化、协同过滤）：降维后乘回矩阵X’
  - 检索、聚类（也需要把样本投到新的特征空间）
  - 特征分析与压缩（GIF、jpg）：可以放松正交性约束

k-means聚类算法简介

- 输入:类的数目k，包含n个文本的特征向量。
- 输出: k个类，使平方误差准则最小。
- 步骤:
- 1)任意选择k个对象作为初始的类中心;
- 2) repeat;
- 3)根据类中对象的平均值，将每个对象(重新)赋给最类似的类;
- 4)更新类的平均值;
- 5) until不再发生变化。

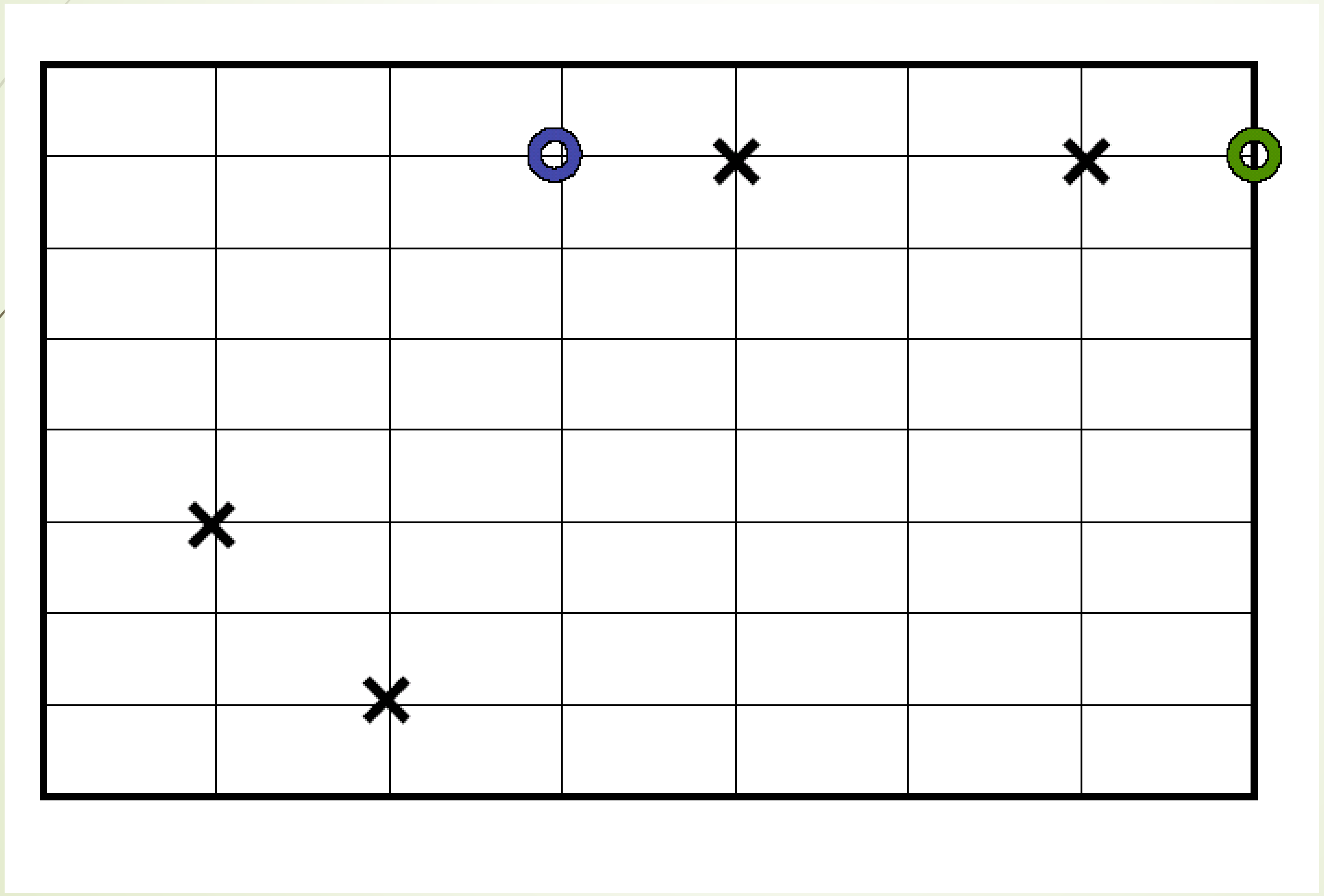
K-均值举例

- 将n个向量分到k个类别中去
  - $x_1=(2,1)$ $x_2=(1,3)$ $x_3=(6,7)$ $x_4=(4,7)$
- 选择k个初始中心
  - $f_1=(4,3)$ $f_2=(5,5)$
- 计算两项距离
- 计算均值

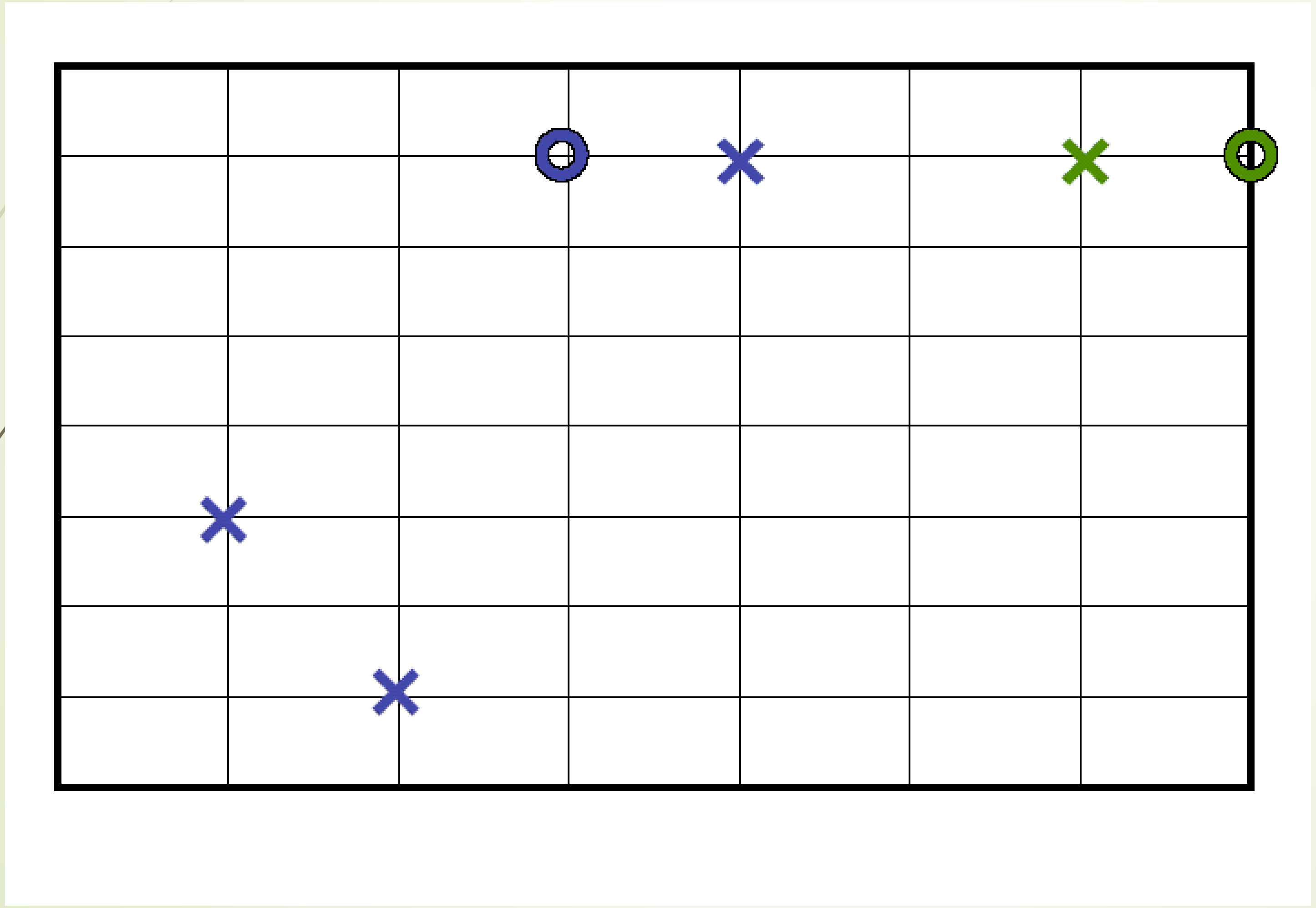
$$d(x_i,x_j) = \sqrt{\sum_{k=1}^{n}(x_{i_k} - x_{j_k})^2}$$

$$\mu(x_1,\ldots,x_n) = \left(\frac{\sum_{i=1}^{n} x_{i_1}}{n},\ldots,\frac{\sum_{i=1}^{n} x_{i_m}}{n}\right)$$

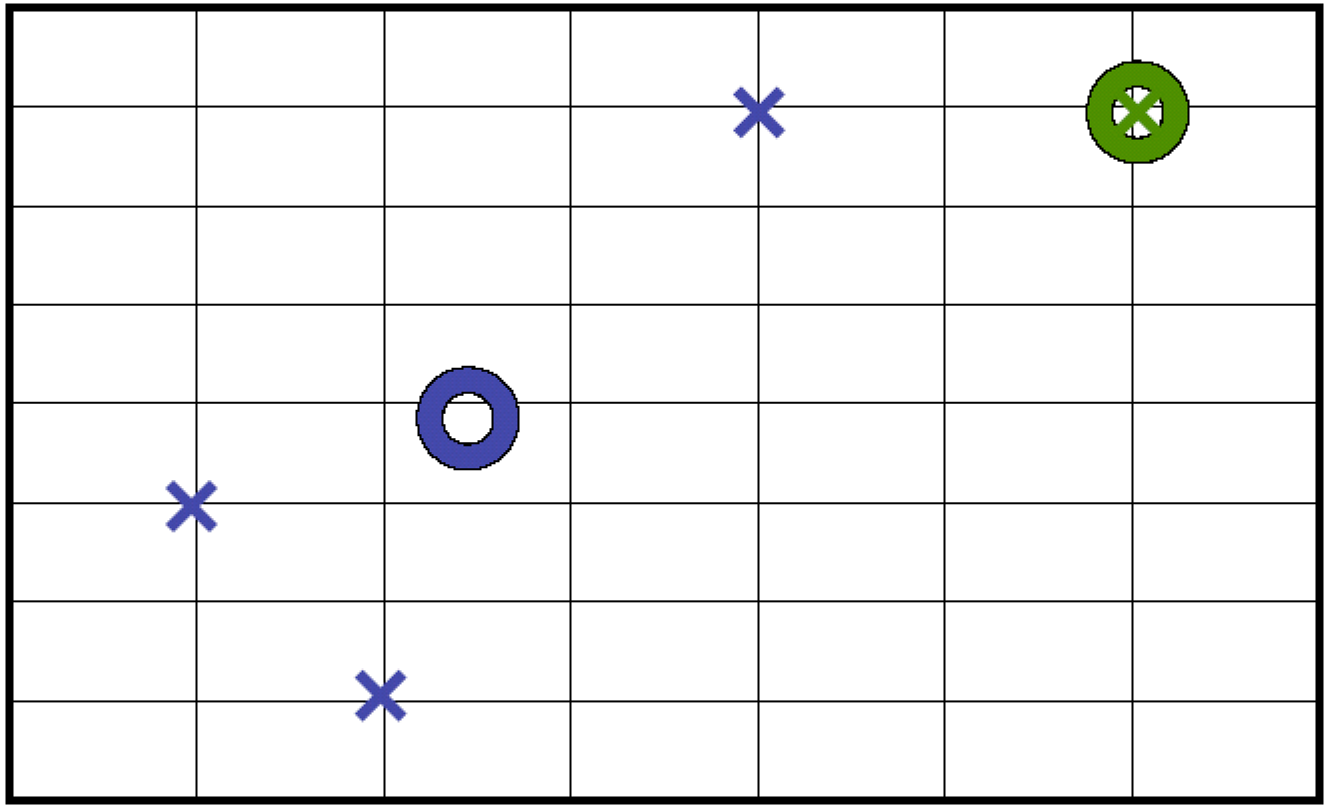
随机两个kernel



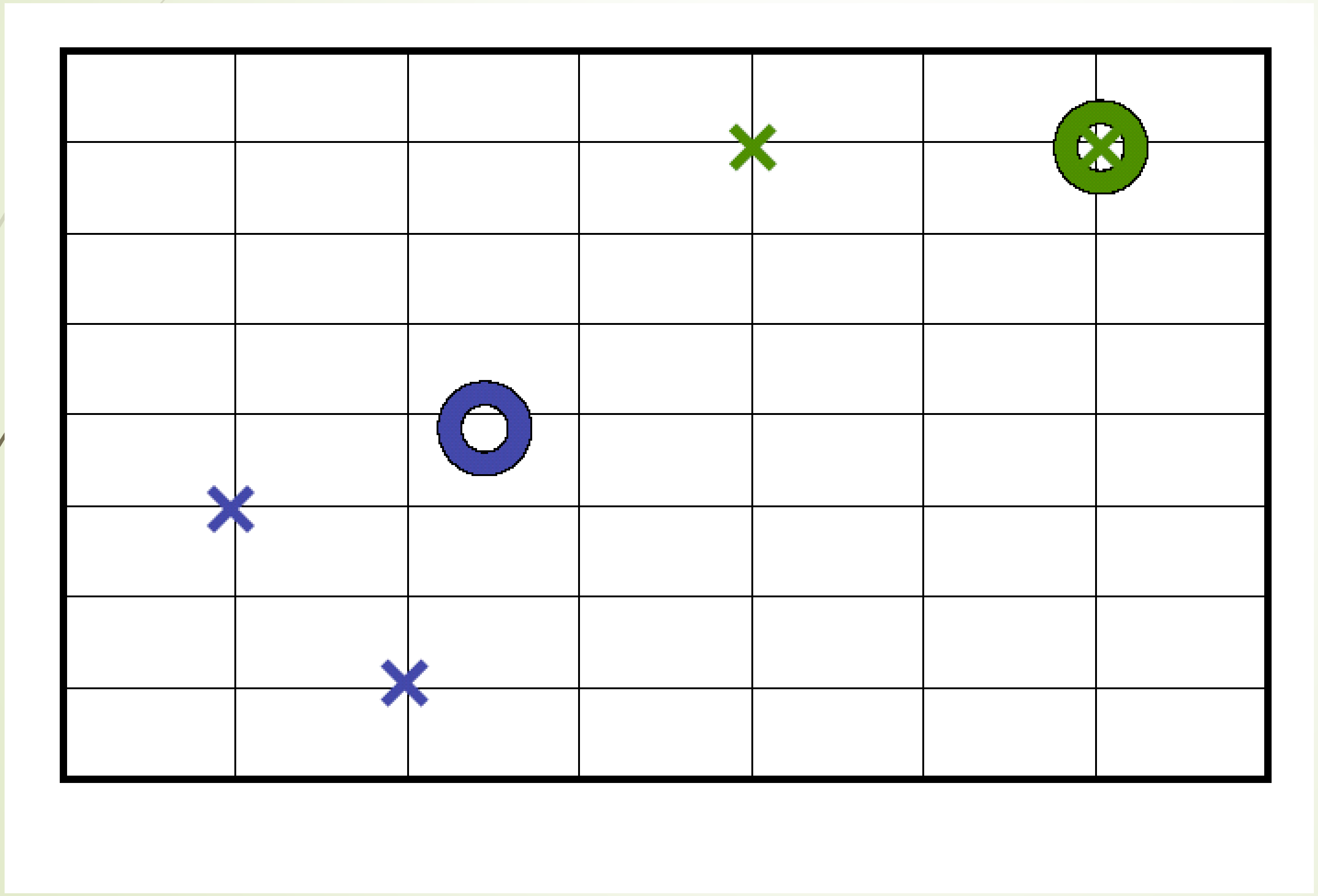
完成按距离的划分



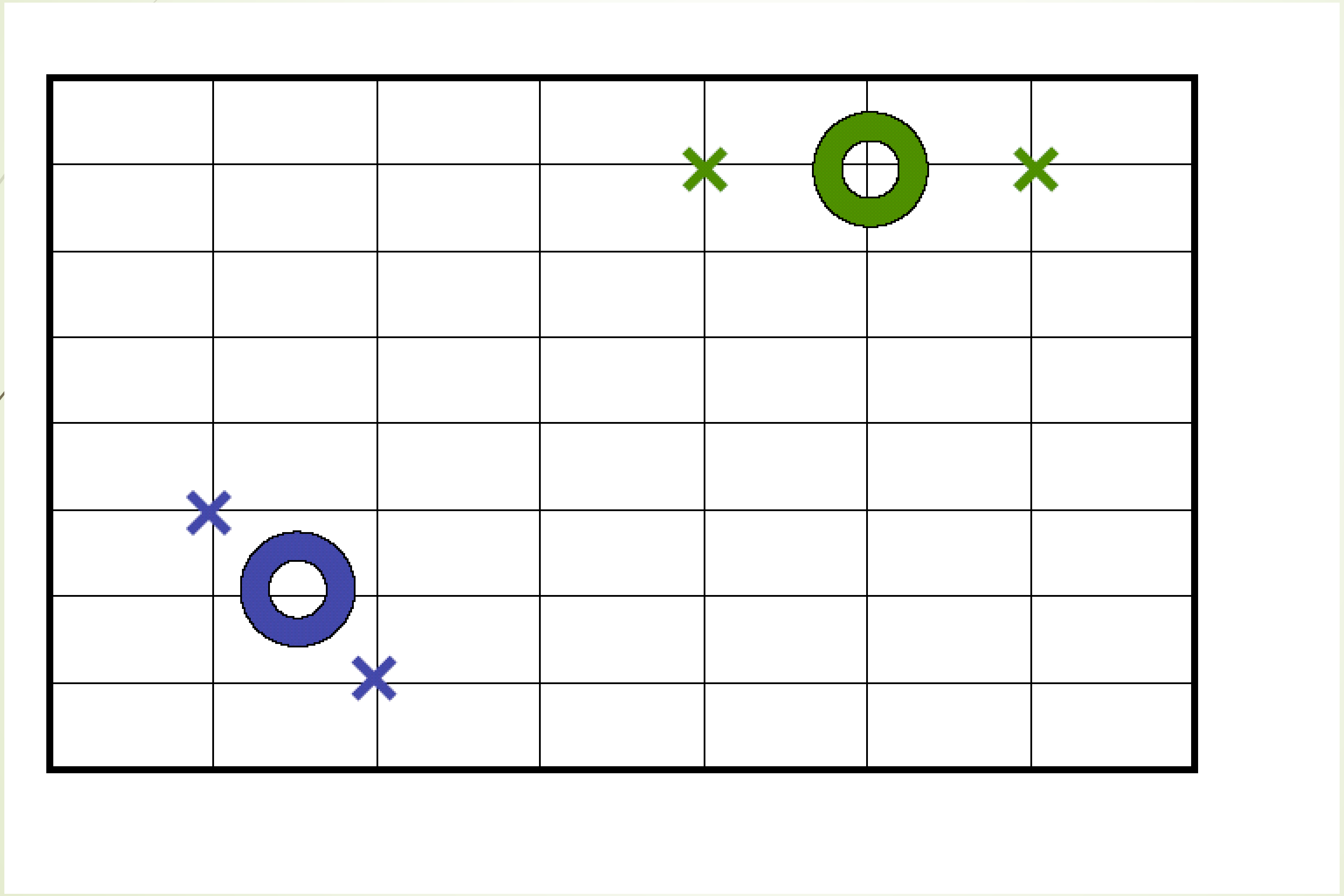
根据划分计算出每个类新的kernel



按新的kernel重新对样本进行划分



迭代到收斂

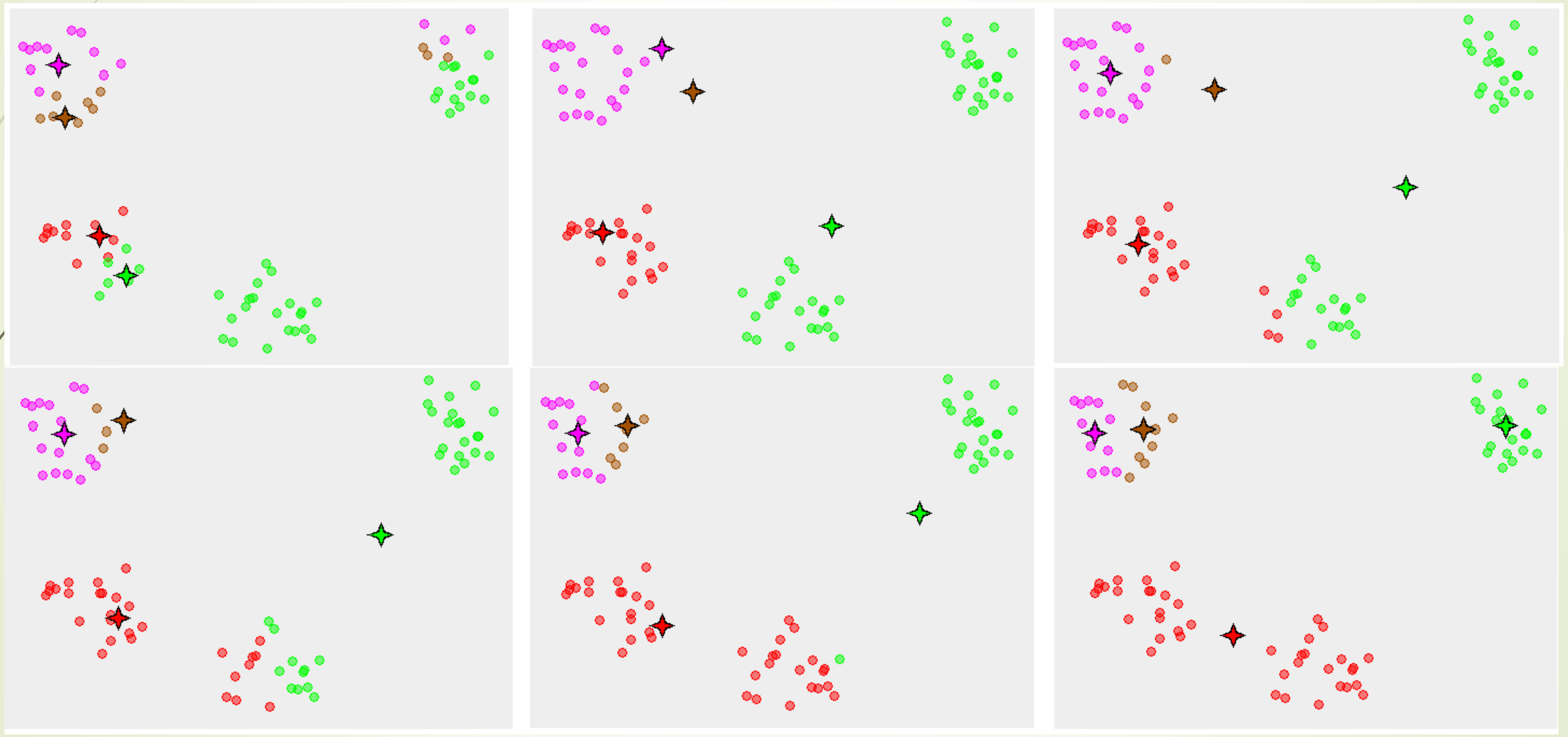


K-均值算法分析

- 算法复杂度为O (kln)，其中l为迭代次数， n为文档个数， k为类别个数
- K-均值算法最后一定是可以收敛的
- 该算法本质上是一种贪心算法。
  - 可以保证局部最优，但是很难保证全局最优。
- 需要预先指定k值和初始划分

K-means convergence to a local minimum

— From Wikipedia



K-means的物理意义：

- VQ下平方误差最小
- 能否采用其他的误差方案？

K-means 的改进

- 确定K
  - 对于不同的K都尝试聚类，取效果最好的
- 确定初始种子
  - 排除明显是“噪声”的文档向量
  - 尝试多种初始向量的组合，取效果最好的
  - 通过其他方法（如层次聚类）确定初始文档向量

聚类的质量评价

- 指标：纯度（Purity）和F值（F-measure）
- 标准答案：一般是人工分好类的文档集合

纯度

- 对于聚类后形成的任意类别r，聚类的纯度定义为
$$P(S_r) = \frac{1}{n_r}\max(n_r^i)$$
- 整个聚类结果的纯度定义为
$$\text{Purity} = \sum_{r=1}^{k}\frac{n_r}{n}P(S_r)$$
- n_r^i：属于预定义类i且被分配到第r个聚类的文档个数
- n_r：第r个聚类类别中的文档个数

F值

- F值：准确率（precision）和召回率（recall）的调和平均数
- $\mathrm{precision(i, r)} = n_r^i / n_r$
- $\mathrm{recall(i, r)} = n_r^i / n_i$
- n_r^i：属于预定义类i且被分配到第r个聚类的文档个数
- n_r：第r个聚类类别中的文档个数
- n_i：预定义类别i中的文档个数

F值

- 聚类r和类别i之间的f值计算如下：
- $f(i, r) = \frac{2 \times recall(i,r) \times precision(i,r)}{precision(i,r) + recall(i,r)}$

- 最终聚类结果的评价函数为
- $F = \sum_i \frac{n_i}{n} \max\{f(i, r)\}$，$n$是所有文档的个数

作业稍晚布置