简介

1. 什么是Numpy?

NumPy（Numerical Python）是 一个C语言编写的，在Python中用于**数值计算**的基础库，核心是`ndarray`（多维数组对象）

中文官网

2. Numpy有哪些功能?

- 高效处理大规模数值数据（比 Python 原生列表快 10-100 倍）
- 支持向量运算、矩阵操作(列表不支持)
- 读写磁盘，操作内存映射文件
- 是 Pandas、Matplotlib 等数据分析库的基础

3. 对比原生Python处理数据，效率真的高很多吗?

```python
import numpy as np
import time

# 列表处理1000万个数字的加法（慢，需循环）
time1 = time.time()
list1 = [i for i in range(10000000)]
list2 = [i*2 for i in range(10000000)]
result = [a + b for a, b in zip(list1, list2)]  # 需循环
time2 = time.time()
print(f"列表处理100万个数字的加法耗时：{time2-time1}")

# NumPy数组处理（快，直接运算）
arr1 = np.arange(10000000)
arr2 = np.arange(10000000) * 2
result = arr1 + arr2  # 无需循环，直接矢量化运算
time3 = time.time()
print(f"NumPy数组处理100万个数字的加法耗时：{time3-time2}")
```

安装

在终端（或 PyCharm/Jupyter 终端）输入：

```
pip install numpy
```

导入约定：

```
import numpy as np  # 简写为np，行业通用
```

核心：ndarray数组

学习目标：

1. 知道ndarray是什么
2. 掌握ndarray的几种创建方式
3. 掌握ndarray的常见属性

介绍

NumPy数组是一个**多维的数组对象**（矩阵），称为ndarray，具有**矢量算术运算能力**和**复杂的广播能力**，并具有**执行速度快**和**节省空间**的特点。注意：ndarray的下标从0开始，且数组里的所有元素必须是相同类型。

创建ndarray数组

下面是常见的创建ndarray数组的几种操作

使用列表创建

创建语法：`np.array(列表对象)`

```
import numpy as np

# 创建一个一维ndarray
arr1 = np.array([1, 2, 3])
```

```
print(arr1)
print(type(arr1))

# 创建一个二维ndarray
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2)
print(type(arr2))

# 创建一个三维ndarray
arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr3)
print(type(arr3))
```

使用函数创建特殊数组

```
np.zeros((2))                # 创建一个2行3列全0数组
np.ones((3, 2))              # 创建一个3行2列全1数组
np.empty((2,3))              # 创建一个2行3列，元素值内容随机且依赖于内存状态的数组
```

注意：默认创建的数组中的数据类型(dtype)都是：float64

arange创建一维数组

arange() 类似 python 的 range()，用于创建一个一维 ndarray 数组。

```
# 创建一个一维数组，起始值0，结束值10，步长2，元素类型:float32
# 左闭右开
arr4 = np.arange(0, 10, 2, dtype=np.float32)

# 数组
# [0., 2., 4., 6., 8.]
```

matrix创建二维数组

matrix 是 ndarray 的子类，只能生成 2 维的矩阵

```python
arr4 = np.matrix([[1, 2, 3], [4, 5, 6]])

# 注意，只能创建二维矩阵
# 会报错 ValueError: matrix must be 2-dimensional
# arr5 = np.matrix([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

创建随机数矩阵

```python
import numpy as np

# 1. 创建一个0-1的随机数矩阵，矩阵是2行3列
arr5 = np.random.random((2,3))
print(arr5)
print(type(arr5))

# 2. 创建一个0-10的随机整数矩阵，矩阵是2行3列
arr6 = np.random.randint(0, 10, (2,3))
print(arr6)
print(type(arr6))

# 3. 创建一个0-10的随机浮点数矩阵，矩阵是2行3列，是均匀分布
arr7 = np.random.uniform(0, 10, (2,3))
print(arr7)
print(type(arr7))
```

属性

```python
# 创建一个2行3列的矩阵、
arr8 = np.array([[1, 2, 3], [4, 5, 6]])

print(arr8.shape)    # 形状（行数，列数）→ (2, 3)
print(arr8.ndim)     # 维度 → 2（二维数组）
print(arr8.dtype)    # 数据类型 → int64（默认）
print(arr8.size)     # 总元素数 → 6
```

基本操作

形状改变（reshape)

```
# reshape(), 改变ndarray的形状

# 一维变多维
arr9 = np.arange(0, 12)
arr10 = arr9.reshape(3,4)

# 说明: 不改变原有矩阵，返回改变之后的新矩阵
print(arr9)
print(arr10)

# 多维变一维
# 默认情况下'C'以行为主的顺序展开，'F'（Fortran风格）意味着以列的顺序展开
arr11 = arr10.reshape((12,),order='F')
print(arr11)
```

类型转换

```
# astype(), 转换ndarray的类型
arr12 = np.array([1, 2, 3, 4, 5, 6])
print(arr12.dtype)
print(arr12)
arr13 = arr12.astype(np.float32)          # 转为浮点型 → [1. 2. 3. 4. 5. 6.]
print(arr13.dtype)
print(arr13)
```

索引与切片

类似于列表的索引与切片，支持多维。

```
# 1. 列表的索引与切片
# list[start:end:step]
arr = [1, 2, 3, 4, 5, 6]
print(arr[0:4:2])             # 1,3

# 2. 一维ndarray的索引与切片， 和列表的索引与切片一样
arr14 = np.arange(0, 12)
print(arr14[0:4:2])           # [0,2]

# 3. 多维ndarray的索引与切片
```

```
arr15 = np.arange(0, 12).reshape(3,4)
print(arr15)

# 取一行(下标为1的一行, 也就是第二行)
print(arr15[1,:])

# 取一列(下标为1的一列, 也就是第二列)
print(arr15[:,1])

# 取不连续的多行
print(arr15[[0,2],:])

# 取从第2行开始的所有行
print(arr15[1:,:])
```

元素访问与修改

```
# 元素访问
arr14 = np.arange(0, 12).reshape(3,4)
print(arr14)
print(arr14[1,1])            # 访问到横坐标为1,纵坐标也为1的元素 --> 5
```

```
# 元素修改
arr15 = np.arange(0, 12).reshape(3,4)

# 修改单个值
arr15[1,1] = 100
print(arr15)

# 修改某一行的值
arr15[1,:] = 100
print(arr15)

# 修改某一列的值
arr15[:,1] = 100
print(arr15)

# 修改某一个块的值
arr15[0:2,0:2] = 100
print(arr15)

# 修改多个不相邻的点 (0,0) (1,1) (2,3)
arr15[[0,1,2],[0,1,3]] = 99
print(arr15)

```

```
# 根据条件修改
arr15[arr15>5] = 0
print(arr15)

# 三目运算
result = np.where(arr15>0,True,False)
print(result)
```

矢量运算(Numpy的核心优势)

Numpy的ndarray，对比Python中原生的list，最重要的优势，其实就是在于可以直接进行矢量运算。

最大的特点是：无需循环，直接对整个数组进行计算

元素的运算

由于numpy的广播机机制在运算过程中，加减乘除的值被广播到所有的元素上面。

```
arr16 = np.arange(0, 12).reshape(3,4)
print(arr16)
print('-' * 20)
print(arr16 + 10)
print('-'*20)
print(arr16 - 5)
print('-'*20)
print(arr16 * 2)
print('-'*20)
print(arr16 / 2)
```

矩阵的运算（数组）

1．同种形状的矩阵运算，运算结果是按位运算

```
# 1. 同种形状的矩阵运算，运算结果是按位运算
arr17 = np.arange(10, 22).reshape(3,4)
print(arr17)
arr18 = np.arange(1, 13).reshape(3,4)
```

```
print(arr18)

# 加法
print(arr17 + arr18)
# 减法
print(arr17 - arr18)
# 乘法
print(arr17 * arr18)
# 除法
print(arr17 / arr18)
```

2. 不同形状的矩阵，不能计算

```
arr19 = np.arange(0, 12).reshape(3,4)
arr20 = np.arange(0, 4).reshape(2,2)

# 报错: ValueError: operands could not be broadcast together with shapes (3,4) (2,2)
# print(arr19 + arr20)
```

3. 行数或者是列数相同的一维数组和多维数组之间可以进行计算

```
# 列数相同（行形状相同）
arr21 = np.arange(0, 12).reshape(3,4)        # 3行4列
print(arr21)
arr22 = np.arange(0, 4)                      # 1行4列
print(arr22)

# 依然是广播的特性
print(arr22-arr21)

# 行数相同（列形状相同）
arr23 = np.arange(0, 12).reshape(3,4)
print(arr23)
arr24 = np.arange(0, 3).reshape(3,1)
print(arr24)
# 依然是广播的特性
print(arr23-arr24)
```

常用统计函数

学习目标：

1 掌握轴的概念

2 学会使用numpy中常用的统计函数

- sum()
- mean()
- max()
- min()
- ...

数组中的轴

在 Numpy 中，“轴（axis）”本质上是**数组维度的编号**，用于描述操作（如求和、求均值、拼接等）沿哪个维度进行。轴的数量等于数组的维度（`ndim`），且编号从 `0` 开始依次递增。

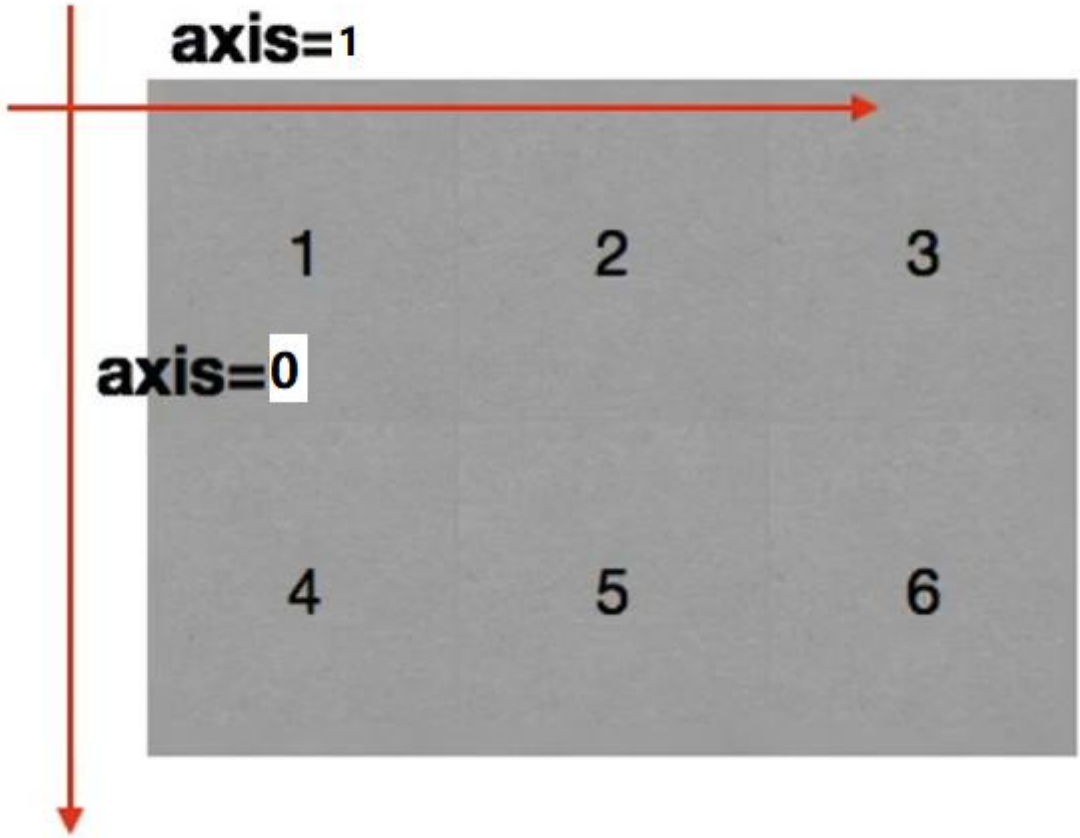
那么，如何区分不同的轴呢?

可以通过数组的“形状（shape）”结合具体场景理解，不同维度的数组对应不同的轴含义：

- 一维数组可看作“一条直线”，只有`1` **个轴（axis=0）**，所有元素都沿这个轴排列

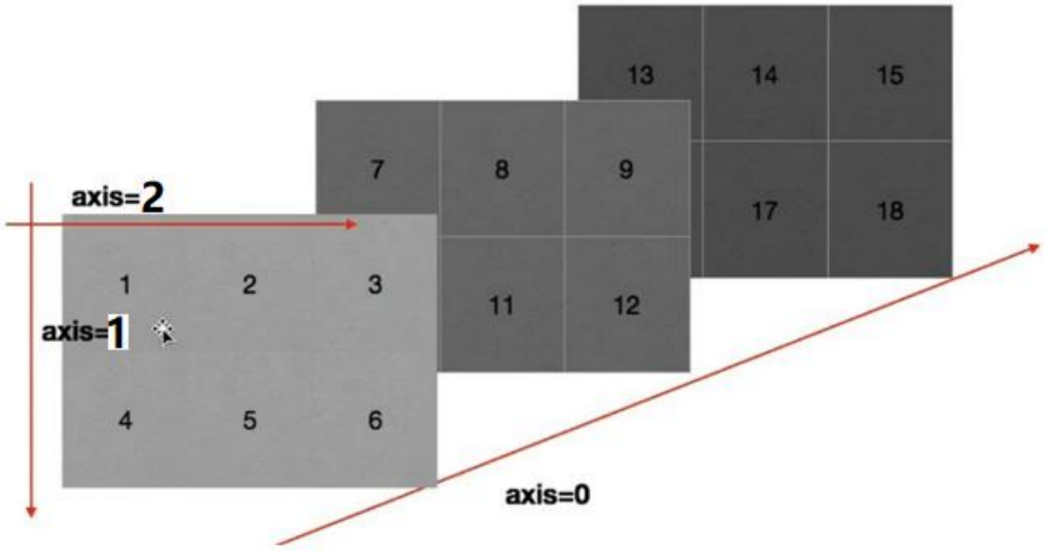
  例如：`arr = np.array([1, 2, 3, 4])`，形状为`(4,)`，axis=`0`。
- 二维数组可看作“一张表格”，有`2` **个轴**：
  - `axis=0`：沿“垂直方向”（跨行），操作时会将“不同行的同一列元素”进行计算；
  - `axis=1`：沿“水平方向”（跨列），操作时会将“同一行的不同列元素”进行计算。

二维数组的轴



- 三维数组可看作“多个二维数组堆叠”（类似“立方体”），有**3 个轴**：
  - `axis=0`：沿“堆叠方向”（跨二维数组），操作时会将“不同二维数组的对应位置元素”计算
  - `axis=1`：沿每个二维数组的“垂直方向”（跨行）；
  - `axis=2`：沿每个二维数组的“水平方向”（跨列）。

三维数组的轴



总结：

1 轴的数量 = 数组维度（`ndim`），编号从 0 开始；

2 **操作沿某个轴进行时，会“压缩该轴”（结果形状会去掉该轴的维度）**

3 理解技巧：想象“沿着轴的方向移动”，收集沿途元素进行计算（如求和、求均值等）

4 实用结论：对哪条轴进行统计操作，哪条轴就会消失

常用统计函数

Numpy中的统计函数有很多，这里列举一些常用的。

- 求和
- 求平均值
- 求最大值
- 求最小值
- 求前缀和
- 求最小值索引
- 求标准差
- 求方差
- 求极值

求和

```
arr25 = np.arange(1, 13).reshape(3, 4)
print(arr25)

# 求所有数据的和
print(arr25.sum())

# 按轴求和
# axis = 0, 表示按列求和
print(arr25.sum(axis=0))

# axis=1, 表示按行求和
print(arr25.sum(axis=1))
```

求平均值

```
arr25 = np.arange(1, 13).reshape(3, 4)
print(arr25)
# 1. 求所有数的平均值
print(arr25.mean())

# 2. 按轴求平均值
# axis = 0, 按列求平均值
print(arr25.mean(axis=0))
```

求最大值

```
arr25 = np.arange(1, 13).reshape(3, 4)
print(arr25)

# 1. 求所有数的最大值
print(arr25.max())

# 2. 按轴求最大值
# axis = 0, 按列求最大值
print(arr25.max(axis=0))
```

求最小值

```
arr25 = np.arange(1, 13).reshape(3, 4)
print(arr25)

# 1. 求所有数的最小值
print(arr25.min())

# 2. 按轴求最小值
# axis = 1, 按行求最小值
print(arr25.min(axis=1))
```

求前缀和

```
arr25 = np.arange(1, 13).reshape(3, 4)
print(arr25)

# 1. 求所有数的前缀和
print(arr25.cumsum())

# 2. 按轴求前缀和
# axis = 1, 按行求前缀和
print(arr25.cumsum(axis=1))
```

求最小值索引

```
# arr25 = np.arange(1, 13).reshape(3, 4)
arr25 = np.random.randint(0, 100, (3, 4))
print(arr25)

# 1. 求所有数的最小值索引
print(arr25.argmin())
# 2. 按轴求最小值索引
# axis = 1, 按行求最小值索引
print(arr25.argmin(axis=1))
```

求标准差

标准差是一组数据平均值分散程度的一种度量。

- 如果标准差较大，那么代表大部分数值和其平均值之间的差异较大

- 如果标准差较小，那么代表大部分数值和其平均值之间的差异较小

标准差越大，代表数据波动越大，越不稳定；标准差越小，代表数据波动小，越稳定。

```
arr25 = np.arange(1, 13).reshape(3, 4)
print(arr25)

# 1. 求所有数的标准差
print(arr25.std())

# 2. 按轴求标准差
# axis = 1, 按行求标准差
print(arr25.std(axis=1))
```

求极值

极值：就是最大值与最小值的差。

```
arr25 = np.arange(1, 13).reshape(3, 4)
print(arr25)

# 1. 求所有数的极值
print(np.ptp(arr25))

# 2. 按轴求极值
# axis = 1, 按行求极值
print(np.ptp(arr25, axis=1))
```

额外说明：Numpy中提供的统计使用的函数还有很多，具体可以参考官方文档

其他操作

数组的添加

`np.append(数组，待添加的元素，[指定轴])`

```
arr27 = np.array([[1,2,3],[4,5,6]])
print(arr27)

# 数组的添加
# 1. 默认情况下，展开为一维数组再添加
arr28 = np.append(arr27, np.array([7,8,9]))
print(arr28)

```

```
# 2. 指定轴添加
# axis = 0, 沿着0轴添加元素
arr29 = np.append(arr27, np.array([[7,8,9]]), axis=0)
print(arr29)

# axis=1, 沿着1轴添加元素
arr30 = np.append(arr27, np.array([[7],[8]]), axis=1)
print(arr30)
```

`np.insert(数组, 索引, 元素, [指定轴])`

```
arr28 = np.array([[1,2,3],[4,5,6]])
print(arr28)

# 数组的插入
# 1. 默认情况下，展开为一维数组再插入
# 2. 插入的元素必须和数组的元素形状相同

# 2表示位置2
# 7表示插入7
arr29 = np.insert(arr28, 2, 7)
print(arr29)

# 指定轴插入
# axis = 0, 沿着0轴插入元素
# 索引0，表示插入第一行
arr30 = np.insert(arr28, 0, np.array([[7,8,9]]), axis=0)
print(arr30)
```

数组的删除

`np.delete(数组, 下标, [指定轴])`

```python
arr28 = np.array([[1,2,3],[4,5,6]])
print(arr28)

# 1. 默认情况下，展开为一维数组再删除， 2表示删除索引为2的元素
arr31 = np.delete(arr28, 2)
print(arr31)
print('-'*20)

# 2. 指定轴删除
# axis = 0, 沿着0轴删除元素
# 0表示删除索引为0的元素， 也就是删除第一行
arr32 = np.delete(arr28, 0, axis=0)
print(arr32)
```

数组的去重

`np.unique(数组，[指定轴])`

```python
arr29 = np.array([1,2,3,4,5,5,5,6,7,8,9,9]).reshape(3,4)
print(arr29)

# 1. 直接去重
# 默认情况下，展开一维数组再去重，得到的也是一维数组
arr30 = np.unique(arr29)
print(arr30)

# 2. 按轴去重
# axis = 0, 沿着0轴去重
# 添加一行重复数据
arr31 = np.append(arr29, np.array([[1,2,3,4]]), axis=0)
print(arr31)

# 去重，去除重复数据
arr32 = np.unique(arr31, axis=0)
print(arr32)
```

拼接与分割

1. 数组的拼接

```python
# 数组的拼接
arr33 = np.array([[1,2,3],[4,5,6]])
arr34 = np.array([[7,8,9],[10,11,12]])

# 默认情况下，沿着轴0进行拼接
arr35 = np.concatenate((arr33, arr34))
print(arr35)
print('-'*20)

# 指定轴了之后，按轴1拼接
print(np.concatenate((arr33, arr34), axis=1))
```

2. 数组的堆叠（会升维，不用关注数据的变化，重点在于维度的变化）

```python
# 一维数组的堆叠
import numpy as np
a = np.array([1, 2, 3])  # shape: (3,)
b = np.array([4, 5, 6])  # shape: (3,)

# axis = 0，沿着0轴进行堆叠
np.stack([a, b], axis=0)
# 结果:
# array([[1, 2, 3],
#        [4, 5, 6]])
# shape: (2, 3)

# axis = 1，沿着1轴进行堆叠
np.stack([a, b], axis=1)
# 结果:
# array([[1, 4],
#        [2, 5],
#        [3, 6]])
# shape: (3, 2)


# 二维数组的堆叠
a = np.array([[1, 2, 3], [4, 5, 6]])  # shape: (2, 3)
b = np.array([[7, 8, 9], [10, 11, 12]])  # shape: (2, 3)

# axis = 0，沿着0轴进行堆叠，（新轴在第0位，长度=2）
np.stack([a, b], axis=0)
# 结果:
# array([[[ 1,  2,  3],
#         [ 4,  5,  6]],
#        [[ 7,  8,  9],
#         [10, 11, 12]]])
# shape: (2, 2, 3)  （新轴在第0位，长度=2）

# axis = 1，沿着1轴进行堆叠，（新轴在第1位，长度=2）
```

```
np.stack([a, b], axis=1)
# 结果:
# array([[[ 1,  2,  3],
#         [ 7,  8,  9]],
#        [[ 4,  5,  6],
#         [10, 11, 12]]])
# shape: (2, 2, 3)  （新轴在第1位，长度=2）

# axis = 2，沿着2轴进行堆叠，（新轴在第2位，长度=2）
np.stack([a, b], axis=2)
# 结果:
# array([[[ 1,  7],
#         [ 2,  8],
#         [ 3,  9]],
#        [[ 4, 10],
#         [ 5, 11],
#         [ 6, 12]]])
# shape: (2, 3, 2)  （新轴在第2位，长度=2）
```

3. 数组的分割

```
arr36 = np.arange(1, 10).reshape(3, 3)
print(arr36)

# 1. 默认情况下，按轴0进行分割
# 3表示分割成3份
arr37 = np.split(arr36, 3)
print(arr37)
print(type(arr37))

# 2. 按轴1进行分割
arr38 = np.split(arr36, 3, axis=1)
print(arr38)
```

数组的转置与轴滚动

```
arr39 = np.arange(0, 12).reshape(3, 4)
print("原始数组:")
print(arr39)

# 1. 转置第一种方式
arr40 = arr39.T
print("转置第一种方式:")
print(arr40)

```

```python
# 2. 转置第二种方式
arr41 = np.transpose(arr39)
print("转置第二种方式:")
print(arr41)

# 3. 对换数组的轴
arr42 = np.transpose(arr39)
print("对换轴:")
print(arr42)

# 4. 轴滚动
# 表示把1轴滚动到0轴的位置
arr43 = np.rollaxis(arr39, 1, 0)
print("轴滚动:")
print(arr43)

# 轴滚动第二个案例
arr44 = np.ones((3, 4, 5, 6))
print(arr44.shape)
# 表示把3轴滚动到1轴的位置
arr45 = np.rollaxis(arr44, 3, 1)
print(arr45.shape)
```

读取文件数据

从文件中读取数据的API：`np.loadtxt()`

```python
# np.loadtxt(fname, dtype=float, delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0)
# fname: 文件名
# dtype: 数据类型
# delimiter: 分隔符，默认为None，表示使用空格进行分隔
# converters: 转换函数，默认为None，表示没有转换函数
# skiprows: 跳过行数，默认为0，表示没有跳过行
# usecols: 使用列数，默认为None，表示使用所有列
# unpack: 是否将数据进行解包，默认为False，表示不进行解包,也就是有多少条记录，就返回多少个数组
# ndmin: 最小维度，默认为0，表示没有最小维度
```

现在这里有一个英国和美国各自youtube 1000多个视频的的csv，表中列如下(也就是数据的特征)

- [点击,喜欢,不喜欢,评论数量]
- ["views","likes","dislikes","comment_total"]

运用知识，读取文件，并且绘制直方图

```
import numpy as np
from matplotlib import pyplot as plt

us_file_path = './youtube_video_data/US_video_data_numbers.csv'
uk_file_path = './youtube_video_data/GB_video_data_numbers.csv'

t_us = np.loadtxt(us_file_path, delimiter=',', dtype=np.int64)
t_uk = np.loadtxt(uk_file_path, delimiter=',', dtype=np.int64)

# print(t_us)
# print(t_us.shape)
# print('-'*30)
# print(t_uk)
# print(t_uk.shape)

# 取评论数据
t_us_comments=t_us[:,-1]

# 数据过滤(一开始注释掉)
# t_us_comments=t_us_comments[t_us_comments<=5000]

# 怎么知道分多少，打印最大和最小值
print(t_us_comments.max(),t_us_comments.min())

d=5000
# d=50

# 计算组数
bin_nums=(t_us_comments.max()-t_us_comments.min())//d

# 绘图
plt.figure(figsize=(20,8),dpi=80)
plt.hist(t_us_comments,bins=bin_nums)

plt.show()

```

数组中的特殊值

在上面的例子中， 我们发现有些数据可能是不合法的，那么就需要剔除。

在以后的工作中，我们获取到的数据中，可能也有一些是不合法的或者是一些特殊值。比如：

我们通过某种手段获取到了1kw个用户的基本信息，基本信息中包括性别，手机号，余额等等，但是实际情况下，这1kw个用户的所有手机号，余额一定有值吗？如果没有值，那么在矩阵中该如何表示呢？

所以，接下来我们介绍一下numpy中的特殊值

- nan：not a number，通常表示确实的数据
- inf：表示正无穷大
- -inf：表示负无穷大

```
import numpy as np

# 创建一个nan和inf #
a = np.nan
b = np.inf
c = -np.inf
print(a,type(a))
print(b,type(b))
print(b > c)

t = np.arange(24,dtype=float).reshape(4,6)

# 将三行四列的数改成nan
t[3,4] = np.nan
print(t)

# 可以使用np.count_nonzero() 来判断非零的个数
print(np.count_nonzero(t))

# 并 且  np.nan != np.nan     结果 是TRUE
# 所以我们可以使用这两个结合使用判断nan的个数
print(np.count_nonzero(t != t))

# 将nan替换为0
t[np.isnan(t)] = 0
print(t)
```

总结

1 Numpy是什么？

是一个数值计算的框架，核心是ndarray（多维数组），可以帮我们高效的处理数据

2 ndarray数组的创建

- 通过列表创建
- 通过API创建特殊的数组：
  - np.zeros()
  - np.ones()
  - np.empty()
- 随机数组
  - 0-1随机浮点数：np.random.random()
  - 指定范围内随机数：np.random.randint()
- 一维数组：np.arange()
- 二维数组：np.matrix()

3 核心属性

- `shape`：数组形状（行数、列数），如`(3,4)`表示 3 行 4 列。
- `ndim`：数组维度，如 2 表示二维数组。
- `dtype`：元素数据类型（默认`int64`/`float64`），可通过`astype()`转换。
- `size`：数组总元素个数，等于各维度大小的乘积。

4 基本操作

- 形状调整（reshape）
- 类型转换（astype）
- 索引与切片
- 元素访问与修改

5 矢量运算

- 元素运算(广播机制)
- 矩阵运算

6 统计函数

- 轴的理解
- 高频统计函数：sum()，mean()，max()，min()，ptp()，std()，cumsum()，argmin()

7 其他操作

- 数组的组合与修改
  - 增删：`np.append()`（添加元素）、`np.insert()`（插入元素）、`np.delete()`（删除元素）、`np.unique()`（去重）

- 拼接与分割：`np.concatenate()`（拼接数组，需指定轴）、`np.split()`（分割数组）、`np.stack()`（堆叠数组，会升维）
- 维度调整：`T`（转置）、`np.transpose()`（轴对换）、`np.rollaxis()`（轴滚动）。

- **文件读取**：`np.loadtxt()`读取文本 / CSV 文件，支持指定分隔符（`delimiter`）、跳过行数（`skiprows`）、选择列（`usecols`）等参数。
- 特殊值：
  - `np.nan`（非数字，缺失值）：特点是`np.nan != np.nan`，需用`np.isnan()`判断，常用`arr[np.isnan(arr)]=0`替换为 `0`
  - `np.inf` / `-np.inf`（正 / 负无穷）：表示超出浮点范围的数值，可通过比较运算（如`np.inf > -np.inf`）判断。