Python科学计算基础之Numpy

1. Numpy简介

NumPy（Numerical Python） 是一个开源的 **Python** 科学计算库，用于快速处理任意维度的数组。

NumPy 支持常见的数组和矩阵操作。对于同样的数值计算任务，使用 NumPy 比直接使用 **Python** 要简洁、快速得多。

NumPy 使用 ndarray 对象来处理多维数组，该对象是一个快速而灵活的大数据容器，是多维数组的一种表示方法。为了提升B格，也可以用『张量』称呼它。用array函数就能够创建一个ndarray，比如。

```
a = np.array(1)
b = np.array([1,2])
c = np.array([[1.0,2.0,3.0],[3.0,4.0,5.0]])
```

这里，a实际上还是一个数，或者叫它『0阶张量』。b是个向量，或者叫它『1阶张量』。c是个矩阵，或者叫它『2阶张量』。ndarray的深度，或者说张量的阶数可以用其ndim属性得到，每一阶的长度可以用其shape属性得到。ndarray的维度方向的索引称为『轴』(axis)。

2. 安装和导入

numpy的安装命令：pip install numpy

在代码中使用numpy，习惯上这样引入。

```
import numpy as np
```

直接import numpy或者as成别的名称也可以，但习惯上还是用上面的方法，即用“np”，这样你的程序能和大都数人的程序保持一致。此外，不要试图 from numpy import * ，这会引起名称冲突，也不好调试。

3. 基本概念

3.1 创建ndarray数组

In [1]:

```
import numpy as np
```

创建ndarray数组的方式有很多种，这里介绍我使用的较多的几种：

方式 1: 基于list或tuple

In [2]:

```
# 一维数组

# 基于list
arr1 = np.array([1,2,3,4])
print(arr1)

# 基于tuple
arr_tuple = np.array((1,2,3,4))
print(arr_tuple)

# 二维数组 (2*3)
arr2 = np.array([[1,2,4], [3,4,5]])
arr2
```

```
[1 2 3 4]
[1 2 3 4]
```

Out[2]:

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

请注意：

- 一维数组用print输出的时候为 [1 2 3 4]，跟python的列表是有些差异的，没有“,”
- 在创建二维数组时，在每个子list外面还有一个"[]"，形式为“**[**[list1], [list2]**]**”

方式 2: 基于np.arange

In [3]:

```
# 一维数组
arr1 = np.arange(5)
print(arr1)

# 二维数组
arr2 = np.array([np.arange(3), np.arange(3)])
arr2
```

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

Out[3]:

```
array([[0, 1, 2],
       [0, 1, 2]])
```

方式 3: 基于arange以及reshape创建多维数组

In [4]:

```
# 创建三维数组
arr = np.arange(24).reshape(2,3,4)
arr
```

Out[4]:

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

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

- 请注意：arange的长度与ndarray的维度的乘积要相等，即 24 = 2X3X4

3.2 Numpy的数值类型

Numpy的数值类型如下：

每一种数据类型都有相应的数据转换函数，参考示例如下：

In [5]:

```
np.int8(12.334)
```

Out[5]:

```
12
```

In [6]:

```
np.float64(12)
```

Out[6]:

```
12.0
```

In [7]:

```
np.float(True)
```

Out[7]:

```
1.0
```

In [8]:

```
bool(1)
```

Out[8]:

```
True
```

在创建ndarray数组时，可以指定数值类型：

In [9]:

```
a = np.arange(5, dtype=float)
a
```

Out[9]:

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

- 请注意，复数不能转换成为整数类型或者浮点数，比如下面的代码会运行出错

In [10]:

```
# float(42 + 1j)
```

ndarray 的类型

名称	描述
np.bool	用一个字节存储的布尔类型（True或False）
np.int8	一个字节大小，-128 至 127
np.int16	整数，-32768 至 32767
np.int32	整数，-2^{31} 至 $2^{32} -1$
np.int64	整数，-2^{63} 至 $2^{63} - 1$
np.uint8	无符号整数，0 至 255
np.uint16	无符号整数，0 至 65535
np.uint32	无符号整数，0 至 $2^{32} - 1$
np.uint64	无符号整数，0 至 $2^{64} - 1$
np.float16	半精度浮点数：16位，正负号1位，指数5位，精度10位
np.float32	单精度浮点数：32位，正负号1位，指数8位，精度23位
np.float64	双精度浮点数：64位，正负号1位，指数11位，精度52位
np.complex64	复数，分别用两个32位浮点数表示实部和虚部
np.complex128	复数，分别用两个64位浮点数表示实部和虚部
np.object_	python对象
np.string_	字符串
np.unicode_	unicode类型

注意：创建数组的时候指定类型

3.3 ndarray数组的属性

- **dtype属性**，ndarray数组的数据类型，数据类型的种类，前面已描述。

In [11]:

```
np.arange(4, dtype=float)
```

Out[11]:

```
array([0., 1., 2., 3.])
```

In [12]:

```
# 'D'表示复数类型
np.arange(4, dtype='D')
```

Out[12]:

```
array([0.+0.j, 1.+0.j, 2.+0.j, 3.+0.j])
```

In [13]:

```
np.array([1.22,3.45,6.779], dtype='int8')
```

Out[13]:

```
array([1, 3, 6], dtype=int8)
```

- **ndim属性**，数组维度的数量

In [14]:

```
a = np.array([[1,2,3], [7,8,9]])
a.ndim
```

Out[14]:

```
2
```

- **shape属性**，数组对象的尺度，对于矩阵，即n行m列,shape是一个元组（tuple）

In [15]:

```
a.shape
```

Out[15]:

```
(2, 3)
```

- **size属性**用来保存元素的数量，相当于shape中nXm的值

In [16]:

```
a.size
```

Out[16]:

```
6
```

- **itemsize**属性返回数组中各个元素所占用的字节数大小。

In [17]:

```
a.itemsize
```

Out[17]:

```
8
```

- **nbytes属性**，如果想知道整个数组所需的字节数量，可以使用nbytes属性。其值等于数组的size属性值乘以itemsize属性值。

In [18]:

```
a.nbytes
```

Out[18]:

```
48
```

In [19]:

```
a.size*a.itemsize
```

Out[19]:

```
48
```

- **T属性**，数组转置

In [20]:

```
b = np.arange(24).reshape(4,6)
b
```

Out[20]:

```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])
```

In [21]:

```
b.T
```

Out[21]:

```
array([[ 0,  6, 12, 18],
       [ 1,  7, 13, 19],
       [ 2,  8, 14, 20],
       [ 3,  9, 15, 21],
       [ 4, 10, 16, 22],
       [ 5, 11, 17, 23]])
```

- **复数的实部和虚部属性，real和imag属性**

In [22]:

```
d = np.array([1.2+2j, 2+3j])
d
```

Out[22]:

```
array([1.2+2.j, 2. +3.j])
```

real属性返回数组的实部

In [23]:

```
d.real
```

Out[23]:

```
array([1.2, 2. ])
```

imag属性返回数组的虚部

In [24]:

```
d.imag
```

Out[24]:

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

- **flat属性**，返回一个numpy.flatiter对象，即可迭代的对象。

In [25]:

```
e = np.arange(6).reshape(2,3)
e
```

Out[25]:

```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [26]:

```
f = e.flat
f
```

Out[26]:

```
<numpy.flatiter at 0x7fe2131b8400>
```

In [27]:

```
for item in f:
    print(item)
```

```
0
1
2
3
4
5
```

可通过位置进行索引，如下：

In [28]:

```
f[2]
```

Out[28]:

```
2
```

In [29]:

```
f[[1,4]]
```

Out[29]:

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

也可以进行赋值

In [30]:

```
e.flat=7
e
```

Out[30]:

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

In [31]:

```
e.flat[[1,4]]=1
e
```

Out[31]:

```
array([[7, 1, 7],
       [7, 1, 7]])
```

下图是对ndarray各种属性的一个小结

3.4 ndarray数组的切片和索引

- **一维数组**

一维数组的切片和索引与python的list索引类似。

In [32]:

```
a =  np.arange(7)
a
```

Out[32]:

```
array([0, 1, 2, 3, 4, 5, 6])
```

In [33]:

```
a[1:4]
```

Out[33]:

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

In [34]:

```
# 每间隔2个取一个数
a[ : 6: 2]
```

Out[34]:

```
array([0, 2, 4])
```

- **二维数组的切片和索引**，如下所示：

In [35]:

```
b = np.arange(12).reshape(3,4)
b
```

Out[35]:

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

In [36]:

```
b[0:3,0:2]
```

Out[36]:

```
array([[0, 1],
       [4, 5],
       [8, 9]])
```

4. 数学计算

4.1 形状转换

- **reshape()和resize()**

In [37]:

```
b.reshape(4,3)
```

Out[37]:

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

In [38]:

```
b
```

Out[38]:

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

In [39]:

```
b.resize(4,3)
b
```

Out[39]:

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

函数resize（）的作用跟reshape（）类似，但是会改变所作用的数组，相当于有inplace=True的效果

- **ravel()和flatten()**，将多维数组转换成一维数组，如下：

In [40]:

```
b.ravel()
```

Out[40]:

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

In [41]:

```
b.flatten()
```

Out[41]:

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

In [42]:

```
b
```

Out[42]:

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

两者的区别在于返回拷贝（copy）还是返回视图（view），flatten()返回一份拷贝，需要分配新的内存空间，对拷贝所做的修改不会影响原始矩阵，而ravel()返回的是视图（view），会影响原始矩阵。

参考如下代码：

In [43]:

```
# flatten()返回的是拷贝，不影响原始数组
# 即数组"b"没有发生变化
b.flatten()[2]=20
b
```

Out[43]:

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

In [44]:

```
# ravel()返回的是视图，会影响原始数组
# 即数组"b"会发生变化
b.ravel()[2]=20
b
```

Out[44]:

```
array([[ 0,  1, 20],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

- **用tuple指定数组的形状**，如下：

In [45]:

```
b.shape=(2,6)
b
```

Out[45]:

```
array([[ 0,  1, 20,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

- **转置**

前面描述了数组转置的属性（T），也可以通过transpose()函数来实现

In [46]:

```
b.transpose()
```

Out[46]:

```
array([[ 0,  6],
       [ 1,  7],
       [20,  8],
       [ 3,  9],
       [ 4, 10],
       [ 5, 11]])
```

4.2 堆叠数组

In [47]:

```
b
```

Out[47]:

```
array([[ 0,  1, 20,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

In [48]:

```
c = b*2
c
```

Out[48]:

```
array([[ 0,  2, 40,  6,  8, 10],
       [12, 14, 16, 18, 20, 22]])
```

- **水平叠加**

hstack()

In [49]:

```
np.hstack((b,c))
```

Out[49]:

```
array([[ 0,  1, 20,  3,  4,  5,  0,  2, 40,  6,  8, 10],
       [ 6,  7,  8,  9, 10, 11, 12, 14, 16, 18, 20, 22]])
```

column_stack()函数以列方式对数组进行叠加，功能类似hstack（）

In [50]:

```
np.column_stack((b,c))
```

Out[50]:

```
array([[ 0,  1, 20,  3,  4,  5,  0,  2, 40,  6,  8, 10],
       [ 6,  7,  8,  9, 10, 11, 12, 14, 16, 18, 20, 22]])
```

- **垂直叠加**

vstack()

In [51]:

```
np.vstack((b,c))
```

Out[51]:

```
array([[ 0,  1, 20,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [ 0,  2, 40,  6,  8, 10],
       [12, 14, 16, 18, 20, 22]])
```

row_stack()函数以行方式对数组进行叠加，功能类似vstack（）

In [52]:

```
np.row_stack((b,c))
```

Out[52]:

```
array([[ 0,  1, 20,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [ 0,  2, 40,  6,  8, 10],
       [12, 14, 16, 18, 20, 22]])
```

- **concatenate()方法，通过设置axis的值来设置叠加方向**

axis=1时，沿水平方向叠加

axis=0时，沿垂直方向叠加

In [53]:

```
np.concatenate((b,c),axis=1)
```

Out[53]:

```
array([[ 0,  1, 20,  3,  4,  5,  0,  2, 40,  6,  8, 10],
       [ 6,  7,  8,  9, 10, 11, 12, 14, 16, 18, 20, 22]])
```

In [54]:

```
np.concatenate((b,c),axis=0)
```

Out[54]:

```
array([[ 0,  1, 20,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [ 0,  2, 40,  6,  8, 10],
       [12, 14, 16, 18, 20, 22]])
```

由于针对数组的轴为0或1的方向经常会混淆，通过示意图，或许可以更好的理解。

关于数组的轴方向示意图，以及叠加的示意图，如下：

深度叠加

这个有点烧脑，举个例子如下，自己可以体会下：

In [55]:

```
arr_dstack = np.dstack((b,c))
print(arr_dstack.shape)
arr_dstack
```

```
(2, 6, 2)
```

Out[55]:

```
array([[[ 0,  0],
        [ 1,  2],
        [20, 40],
        [ 3,  6],
        [ 4,  8],
        [ 5, 10]],

       [[ 6, 12],
        [ 7, 14],
        [ 8, 16],
        [ 9, 18],
        [10, 20],
        [11, 22]]])
```

叠加前，b和c均是shape为（2,6）的二维数组，叠加后，arr_dstack是shape为（2,6,2）的三维数组。

深度叠加的示意图如下：

In [ ]:

```

```

4.3 数组的拆分

跟数组的叠加类似，数组的拆分可以分为横向拆分、纵向拆分以及深度拆分。

涉及的函数为 hsplit()、vsplit()、dsplit() 以及split()

In [56]:

```
b
```

Out[56]:

```
array([[ 0,  1, 20,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

- **沿横向轴拆分（axis=1）**

In [57]:

```
np.hsplit(b, 2)
```

Out[57]:

```
[array([[ 0,  1, 20],
        [ 6,  7,  8]]), array([[ 3,  4,  5],
        [ 9, 10, 11]])]
```

In [58]:

```
np.split(b,2, axis=1)
```

Out[58]:

```
[array([[ 0,  1, 20],
        [ 6,  7,  8]]), array([[ 3,  4,  5],
        [ 9, 10, 11]])]
```

In [ ]:

```

```

- **沿纵向轴拆分（axis=0）**

In [59]:

```
np.vsplit(b, 2)
```

Out[59]:

```
[array([[ 0,  1, 20,  3,  4,  5]]), array([[ 6,  7,  8,  9, 10, 1
1]])]
```

In [60]:

```
np.split(b,2,axis=0)
```

Out[60]:

```
[array([[ 0,  1, 20,  3,  4,  5]]), array([[ 6,  7,  8,  9, 10, 1
1]])]
```

- **深度拆分**

In [61]:

```
arr_dstack
```

Out[61]:

```
array([[[ 0,  0],
        [ 1,  2],
        [20, 40],
        [ 3,  6],
        [ 4,  8],
        [ 5, 10]],

       [[ 6, 12],
        [ 7, 14],
        [ 8, 16],
        [ 9, 18],
        [10, 20],
        [11, 22]]])
```

In [62]:

```
np.dsplit(arr_dstack,2)
```

Out[62]:

```
[array([[[ 0],
         [ 1],
         [20],
         [ 3],
         [ 4],
         [ 5]],

        [[ 6],
         [ 7],
         [ 8],
         [ 9],
         [10],
         [11]]]), array([[[ 0],
         [ 2],
         [40],
         [ 6],
         [ 8],
         [10]],

        [[12],
         [14],
         [16],
         [18],
         [20],
         [22]]])]
```

拆分的结果是原来的三维数组拆分成为两个二维数组。

这个烧脑的拆分过程可以自行分析下~~

4.4 数组的类型转换

- **数组转换成list，使用tolist()**

In [63]:

```
b
```

Out[63]:

```
array([[ 0,  1, 20,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

In [64]:

```
b.tolist()
```

Out[64]:

```
[[0, 1, 20, 3, 4, 5], [6, 7, 8, 9, 10, 11]]
```

- **转换成指定类型，astype()函数**

In [65]:

```
b.astype(float)
```

Out[65]:

```
array([[ 0.,  1., 20.,  3.,  4.,  5.],
       [ 6.,  7.,  8.,  9., 10., 11.]])
```

In [ ]:

```

```

4.5 数组的广播

当数组跟一个标量进行数学运算时，标量需要根据数组的形状进行扩展，然后执行运算。

这个扩展的过程称为“广播（broadcasting）”

In [66]:

```
b
```

Out[66]:

```
array([[ 0,  1, 20,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

In [67]:

```
d = b + 2
d
```

Out[67]:

```
array([[ 2,  3, 22,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13]])
```

5. numpy常用统计函数

常用的函数如下：

请注意函数在使用时需要指定axis轴的方向，若不指定，默认统计整个数组。

- np.sum()，返回求和
- np.mean()，返回均值
- np.max()，返回最大值
- np.min()，返回最小值
- np.ptp()，数组沿指定轴返回最大值减去最小值，即（max-min）
- np.std()，返回标准偏差（standard deviation）
- np.var()，返回方差（variance）
- np.cumsum()，返回累加值
- np.cumprod()，返回累乘积值

In [68]:

```
b
```

Out[68]:

```
array([[ 0,  1, 20,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

In [69]:

```
np.max(b)
```

Out[69]:

```
20
```

In [70]:

```
# 沿axis=1轴方向统计
np.max(b,axis=1)
```

Out[70]:

```
array([20, 11])
```

In [71]:

```
# 沿axis=0轴方向统计
np.max(b,axis=0)
```

Out[71]:

```
array([ 6,  7, 20,  9, 10, 11])
```

In [72]:

```
np.min(b)
```

Out[72]:

```
0
```

- **np.ptp()**，返回整个数组的最大值减去最小值，如下：

In [73]:

```
np.ptp(b)
```

Out[73]:

```
20
```

In [74]:

```
# 沿axis=0轴方向
np.ptp(b, axis=0)
```

Out[74]:

```
array([ 6,  6, 12,  6,  6,  6])
```

In [75]:

```
# 沿axis=1轴方向
np.ptp(b, axis=1)
```

Out[75]:

```
array([20,  5])
```

- **np.cumsum()**，沿指定轴方向进行累加

In [76]:

```
b.resize(4,3)
b
```

Out[76]:

```
array([[ 0,  1, 20],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [77]:

```
np.cumsum(b, axis=1)
```

Out[77]:

```
array([[ 0,  1, 21],
       [ 3,  7, 12],
       [ 6, 13, 21],
       [ 9, 19, 30]])
```

In [78]:

```
np.cumsum(b, axis=0)
```

Out[78]:

```
array([[ 0,  1, 20],
       [ 3,  5, 25],
       [ 9, 12, 33],
       [18, 22, 44]])
```

- **np.cumprod()**，沿指定轴方向进行累乘积 （Return the cumulative product of the elements along the given axis）

In [79]:

```
np.cumprod(b,axis=1)
```

Out[79]:

```
array([[  0,   0,   0],
       [  3,  12,  60],
       [  6,  42, 336],
       [  9,  90, 990]])
```

In [80]:

```
np.cumprod(b,axis=0)
```

Out[80]:

```
array([[   0,    1,   20],
       [   0,    4,  100],
       [   0,   28,  800],
       [   0,  280, 8800]])
```

7. 数据处理案例

采用读取花萼数据为例，这是一个机器学习的典型案例。

In [81]:

```
import numpy as np
# 读取文件
iris_sepal_length = np.loadtxt("iris_sepal_length.csv",delimiter=",")
iris_sepal_length
```

Out[81]:

```
array([5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.6, 5. , 4.4, 4.9, 5.4, 4.8,
4.8,
       4.3, 5.8, 5.7, 5.4, 5.1, 5.7, 5.1, 5.4, 5.1, 4.6, 5.1, 4.8,
5. ,
       5. , 5.2, 5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9,
4.4,
       5.1, 5. , 4.5, 4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. , 7. ,
6.4,
       6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9, 6. , 6.1,
5.6,
       6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6, 6.8,
6.7,
       6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5,
5.5,
       6.1, 5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7, 6.3, 5.8, 7.1,
6.3,
       6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4, 6.8, 5.7, 5.8, 6.4,
6.5,
       7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2, 6.2, 6.1, 6.4,
7.2,
       7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9, 6.7, 6.9,
5.8,
       6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 5.9])
```

In [82]:

```
print("花萼长度表为:",iris_sepal_length)
```

```
花萼长度表为: [5.1 4.9 4.7 4.6 5.  5.4 4.6 5.  4.4 4.9 5.4 4.8 4.8 4.3
5.8 5.7 5.4 5.1
 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.  5.  5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9
5.
 5.5 4.9 4.4 5.1 5.  4.5 4.4 5.  5.1 4.8 5.1 4.6 5.3 5.  7.  6.4 6.9
5.5
 6.5 5.7 6.3 4.9 6.6 5.2 5.  5.9 6.  6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9
6.1
 6.3 6.1 6.4 6.6 6.8 6.7 6.  5.7 5.5 5.5 5.8 6.  5.4 6.  6.7 6.3 5.6
5.5
 5.5 6.1 5.8 5.  5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9
7.3
 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.  6.9 5.6 7.7 6.3 6.7
7.2
 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.  6.9 6.7 6.9 5.8
6.8
 6.7 6.7 6.3 6.5 6.2 5.9]
```

In [83]:

```
# 对数据进行排序
iris_sepal_length.sort()
print("排序后的花萼长度表示为:",iris_sepal_length)
```

```
排序后的花萼长度表示为: [4.3 4.4 4.4 4.4 4.5 4.6 4.6 4.6 4.6 4.7 4.7 4.8
4.8 4.8 4.8 4.8 4.9 4.9
 4.9 4.9 4.9 4.9 5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.1 5.1 5.1
5.1
 5.1 5.1 5.1 5.1 5.1 5.2 5.2 5.2 5.2 5.3 5.4 5.4 5.4 5.4 5.4 5.4 5.5
5.5
 5.5 5.5 5.5 5.5 5.5 5.6 5.6 5.6 5.6 5.6 5.6 5.7 5.7 5.7 5.7 5.7 5.7
5.7
 5.7 5.8 5.8 5.8 5.8 5.8 5.8 5.8 5.9 5.9 5.9 6.  6.  6.  6.  6.  6.
6.1
 6.1 6.1 6.1 6.1 6.1 6.2 6.2 6.2 6.2 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3
6.3
 6.4 6.4 6.4 6.4 6.4 6.4 6.4 6.5 6.5 6.5 6.5 6.5 6.6 6.6 6.7 6.7 6.7
6.7
 6.7 6.7 6.7 6.7 6.8 6.8 6.8 6.9 6.9 6.9 6.9 7.  7.1 7.2 7.2 7.2 7.3
7.4
 7.6 7.7 7.7 7.7 7.7 7.9]
```

In [84]:

```
# 去除重复值
print("去除重复的花萼长度表为:",np.unique(iris_sepal_length))
```

```
去除重复的花萼长度表为: [4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.  5.1 5.2 5.3 5.4
5.5 5.6 5.7 5.8 5.9 6.
 6.1 6.2 6.3 6.4 6.5 6.6 6.7 6.8 6.9 7.  7.1 7.2 7.3 7.4 7.6 7.7 7.
9]
```

In [85]:

```
# 计算数组的总和
print("花萼表的总和为:",np.sum(iris_sepal_length))
```

```
花萼表的总和为: 876.5
```

In [86]:

```
# 计算所有元素的累计和
print('花萼长度表的累计和:',np.cumsum(iris_sepal_length))
```

```
花萼长度表的累计和: [  4.3   8.7  13.1  17.5  22.   26.6  31.2  35.8  4
0.4  45.1  49.8  54.6
  59.4  64.2  69.   73.8  78.7  83.6  88.5  93.4  98.3 103.2 108.2 1
13.2
 118.2 123.2 128.2 133.2 138.2 143.2 148.2 153.2 158.3 163.4 168.5 1
73.6
 178.7 183.8 188.9 194.  199.1 204.3 209.5 214.7 219.9 225.2 230.6 2
36.
 241.4 246.8 252.2 257.6 263.1 268.6 274.1 279.6 285.1 290.6 296.1 3
01.7
 307.3 312.9 318.5 324.1 329.7 335.4 341.1 346.8 352.5 358.2 363.9 3
69.6
 375.3 381.1 386.9 392.7 398.5 404.3 410.1 415.9 421.8 427.7 433.6 4
39.6
 445.6 451.6 457.6 463.6 469.6 475.7 481.8 487.9 494.  500.1 506.2 5
12.4
 518.6 524.8 531.  537.3 543.6 549.9 556.2 562.5 568.8 575.1 581.4 5
87.7
 594.1 600.5 606.9 613.3 619.7 626.1 632.5 639.  645.5 652.  658.5 6
65.
 671.6 678.2 684.9 691.6 698.3 705.  711.7 718.4 725.1 731.8 738.6 7
45.4
 752.2 759.1 766.  772.9 779.8 786.8 793.9 801.1 808.3 815.5 822.8 8
30.2
 837.8 845.5 853.2 860.9 868.6 876.5]
```

In [87]:

```
print("花萼长度表的均值为:",np.mean(iris_sepal_length))
```

```
花萼长度表的均值为: 5.843333333333334
```

In [88]:

```
# 计算数组标准差
print("花萼长度表的标准差:",np.std(iris_sepal_length))
```

```
花萼长度表的标准差: 0.8253012917851409
```

In [89]:

```
# 计算数组方差
print("花萼长度表的方差:",np.var(iris_sepal_length))
```

```
花萼长度表的方差: 0.6811222222222223
```

In [90]:

```
# 计算最小值
print("花萼长度表的最小值为:",np.min(iris_sepal_length))
```

```
花萼长度表的最小值为: 4.3
```

In [91]:

```
# 计算最大值
print("花萼长度表的最大值为:",np.max(iris_sepal_length))
```

花萼长度表的最大值为： 7.9

In [ ]:

```

```