Python数据分析与展示

Github：https://github.com/IvenStarry
学习视频网站：
中国大学MOOC北京理工大学嵩天教授 https://www.icourse163.org/course/BIT-1001870002?tid=1472922453
菜鸟网 https://www.runoob.com/

NumPy

NumPy

1. 是一个开源的Python科学计算基础库
2. 有一个强大的N维数组对象ndarray
3. 广播功能函数
4. 整合C/C++/Fortran代码的工具
5. 线性代数、傅里叶变换、随机数生成等功能
6. 是SciPy、Pandas等数据处理或科学计算库的基础

数据的维度

维度是一组数据的组织形式

列表和数组区别

数据结构	特点
列表	数据类型可以不同
数组	数据类型相同

不同维度数据的组成

维度	概念
一维数据	由对等关系的有序或无序数据构成，采用线性方式组织
二维数据	由多个一维数据构成，是一维数据的组合形式（表格）
多维数据	由一维或二维数据在新维度上扩展得来
高维数据	仅利用最近的二元关系展示数据间的复杂结构（字典）

数据维度的Python表示

维度	在Python的表示
一维数据	列表和集合
二维数据	列表

维度	在Python的表示
多维数据	列表
高维数据	字典或数据表示格式（JSON、XML、YAML）

NumPy的数组对象ndarray

N维数组对象ndarray

1. 数组对象可以去掉元素间运算所需的循环，使一维向量更像单个数据
2. 设置专门的数组对象，提升运算速度（底层代码由C语言编写）
   采用相同的数据类型，助于节省运算和存储空间
3. ndarray是一个多维数组对象，由两部分构成：实际的数据，描述这些数据的元数据（数据维度、数据类型）
4. 一般要求所有元素类型相同，数组下标从0开始

np.array生成一个ndarray数组 输出成[]形式，元素用空格分割
轴(axis):保存数据的维度 **秩**:轴的数量

python仅支持**整数**、**浮点数**和**复数**3种类型
科学计算设计数据较多，对存储和性能有较高要求，对元素类型精确定义有助于NumPy合理使用存储空间并优化性能，也有助于程序员对程序规模合理评估

```
import numpy as np
# x = np.array(list/tuple, dtype='int32', ndmin) 可以指定数据类型
# 从列表
x = np.array([0, 1, 2, 3], dtype=np.float64)
print(x)
x = np.array([1, 2, 3], dtype=complex)
print(x)
# 从元组
x = np.array((0, 1, 2, 3))
print(x)
# 从列表和元组混合类型创建
x = np.array(([0, 1], [1, 2], [2, 3], [3, 4]))
print(x)
# ndmin 指定生成数组的最小维度
x = np.array([1, 2, 3, 4, 5], ndmin=2)
print(x)
```

数据类型

```
import numpy as np
# bool_ int_ uint8 float_ complex_

# todo dtype(object, align, copy) 转换为的数据类型对象 True填充字段使其类似C的结构体
复制dtype对象，若为false则是对内置数据类型对象的引用
dt = np.dtype(np.int32)
```

```
print(dt)
# int8 int16 int32 int64 四种数据类型可以使用字符串'i1' 'i2' 'i4' 'i8'代替
dt = np.dtype('i4')
print(dt)
# 字节顺序标注  <意味着小端法（低位字节存储在低位地址）  >意味着大端法（高位字节存放在低位地址）
dt = np.dtype('<i4')
print(dt)
# 创建结构化数据类型
dt = np.dtype([('age', np.int8)])
print(dt)
# 数据类型应用于ndarray对象
a = np.array([(10, ), (20, ), (30, )], dtype=dt)
print(a)
# 类型字段名用于存取实际的age列
print(a['age'])
# 定义一个结构化数据类型，将这个dtype应用于ndarray对象
student = np.dtype([('name', 'S20'), ('age', 'i1'), ('marks', 'f4')])
print(student)
a = np.array([('abc', 21, 50), ('xyz', 18, 75)], dtype=student)
print(a)

"""
内建类型符号
b bool
i int
u uint
f float
c 复数浮点型
m timedelta 时间间隔
M datetime 日期时间
O python对象
S, a (byte)字符串
U unicode 统一码 编码格式 数字到文字的映射
V 原始数据(void)
"""
```

数据属性

```
import numpy as np
a = np.array([[0, 1, 2, 3, 4], [9, 8, 7, 6, 5]])

"""
ndarray对象属性:
1. .ndim 轴的数量或维度的数量
2. .shape 对象的尺度，对于矩阵 n行m列
3. .size 对象元素的个数 相当于n*m
4. .dtype 元素类型
5. .itemsize 对象的每个元素的大小，以字节为单位
6. .flags 返回ndarray对象的内存信息 ： 包含了
    C_CONTIGUOUS : True       数据在一个单一C风格的连续段中
```

```
    F_CONTIGUOUS : False     数据是在一个单一的Fortan风格的连续段中
    OWNDATA : True           数组拥有它所使用的内存或从另外一个对象中借用它
    WRITEABLE : True         数据区域可以被写入 若False则只可读
    ALIGNED : True           数据和元素都适当对齐在硬件上
    WRITEBACKIFCOPY : False  这个数组是其他数组的一个副本。当这个数组被释放时，原数组将更新
"""
print(a.ndim)
print(a.shape)
print(a.size)
print(a.dtype)
print(a.itemsize)
print(a.flags)

# 可以由非同质对象构成(在numpy2.0.0版本中不支持 回退1.23.0版本才可以实现)
# 非同质ndarray对象无法发挥numpy优势 应避免书写
# x = np.array([[0, 1, 2, 3, 4], [5, 6, 7, 8]])
# print(x)
# print(x.shape) # (2, )
# print(x.size) # 2
# # 元素为对象类型
# print(x.dtype) # ('0')
# print(x.itemsize) # 4
```

创建数组

```
import numpy as np

# empty(shape, dtype=float, order='C') 生成未初始化数组 order 可选"C"或"F"代表行优先或列有限，在计算机内存中的存储元素的顺序
x = np.empty([3, 2], dtype=int)
print(x) # 数组元素随机值

# np.arange(n) 生成从0到n-1的adarray数组 返回整数型数据
x = np.arange(10)
print(x)

# np.ones(shape) 生成和shape大小一致的全1矩阵，shape是元组类型 数据类型默认浮点型
x = np.ones((2, 5))
print(x)
x = np.ones((2, 3, 4)) # 从外到内 最外层两个元素 每个元素三个维度 每个维度四个元素
print(x)

# np.zeros(shape) 生成和shape大小一致的全0矩阵，shape是元组类型 数据类型默认浮点型(可以指定数据类型)
x = np.zeros((3, 4), dtype=np.int_)
print(x)

# np.full(shape, val) 生成和shape大小一致的全val矩阵
x = np.full((3, 4), 10)
print(x)
```

```
# np.eye(n) 生成n*n的单位矩阵 对角线为1 其余为0 数据类型默认浮点型
x = np.eye(5)
print(x)

a = [[[1, 2], [2, 3]], [[3, 4], [4, 5]], [[5, 6], [6, 7]]]
# np.ones_like(a) 根据数组a的形状生成全1数组 subok 为True时，使用object的内部数据类型； 为False时，使用数组的数据类型
#创建矩阵
a=np.asmatrix([1,2,3,4])
#输出为矩阵类型
print(type(a))
#既要赋值一份副本，又要保持原类型
at=np.array(a,subok=True)
af=np.array(a) #默认为False
print('at.subok为True:',type(at))
print('af.subok为False:',type(af))
print(id(af),id(a))

# np.zeros_like(a,order="K") 根据数组a的形状生成全0数组 order默认K保留输入数组的存储顺序 可选C F
x = np.zeros_like(a)
print(x)

# np.full_like(a, val) 根据数组a的形状生成全val数组
x = np.full_like(a, 5)
print(x)
```

从已有的数组创建数组

```
import numpy as np

# asarray(a, dtype, order) 类似array 参数比array少俩
x = [1, 2, 3]
a = np.asarray(x)
print(a)

# 元组转换ndarray
x = (1, 2, 3)
a = np.asarray(x)
print(a)

# 元组列表转换ndarray 高版本无法生成非同质数组
# // x = [(1, 2 , 3), (4, 5)]
# // a = np.asarray(x)
# // print(a)

# frombuffer(buffer, dtype, count=-1, offset) 实现动态数组 接受buffer输入参数 以流的形式读取转化成ndarray对象
# offset读取起始位置 默认0   b" "前缀表示：后面字符串是bytes 类型
s = b'Hello World!'
```

```
a = np.frombuffer(s, dtype='S1')
print(a)

# fromiter(iterable, dtype, count=-1) 从迭代对象中建立ndarray对象，返回一维数组
list = range(5)
it = iter(list)
x = np.fromiter(it, dtype=float)
print(x)
```

从数值范围创建数组

```
import numpy as np

# np.arange(start, stop, step, dtype)不包含stop
x = np.arange(10)
print(x)
x = np.arange(10, dtype=float)
print(x)
x = np.arange(10, 20, 2)
print(x)

# np.linspace(start, end, num, endpoint=True, retstep=False, dtype)
# num数组元素个数 4个元素有三个间隔 间隔为(10-1)/3=3
a = np.linspace(1, 10, 4)
print(a)
# 若设置endpoint为False 则不以end这个数结尾 但仍生成4个数 5个元素（包括10）有四个间隔
间隔变为(10-1)/4=2.25
b = np.linspace(1, 10, 4, endpoint=False)
print(b)
# 设置间距 retstep为True时，生成的数组显示间距
a = np.linspace(1, 10, 10, retstep=True)
print(a)
b = np.linspace(1, 10, 10).reshape([10,1])
print(b)

# np.logspace(s, s, num, endpoint, base=10.0, dtype) base log的对数
a = np.logspace(1.0, 2.0, 10)
print(a)
```

切片和索引

```
'''
索引：获取数组中特定位置的元素
切片：获取数组元素子集的过程
'''
import numpy as np

# 一维数组
a = np.array([11, 22, 33, 44, 55])
```

```
print(a[2])
s = slice(1, 4, 2) # 索引2 到索引4停止
print(a[s])
print(a[1:4:2]) # 同python列表 start end(not include) step

# 多维数组
a = np.arange(24).reshape((2, 3, 4))
print(a)
print(a[1, 2, 3])
print(a[0, 1, 2])
print(a[-1, -2, -3])
print(a[:, 1, -3]) # 第一个维度全选 第二个维度取索引为1的元素 第三个维度选索引为-3的元素
print(a[:, 1:3, :])
print(a[:, :, ::2])
print(a[..., ::2]) # 有...代表全选前面所有维度 ： 只能全选一个维度 ...可以全选所有维度

print('----------------------------')
print(a[0, ...])
print(a[0, ..., :])
print(a[0, :, :])
```

高级索引

```
import numpy as np

# 整数数组索引 使用一个数组访问另一个数组元素
x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0, 1, 2], [0, 1, 0]] # 第一个维度的索引[0, 1, 2] 第二个维度的索引[0, 1, 0]
print(y)

x = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])
print(f'数组是:\n{x}')
rows = [[0, 0], [3, 3]]
cols = [[0, 2], [0, 2]]
print(f'四个角的元素为:\n{x[rows, cols]}')

a = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
b = a[1:3, 1:3]
c = a[1:3, [1,2]]
d = a[...,1:]
print(b)
print(c)
print(d)

# 布尔索引 通过布尔运算获取符合指定条件的元素的数组
x = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])
print(x[x > 5])
a = np.array([np.nan, 1, 2, np.nan, 3, 4, 5]) # nan 非数字元素
print(a[~np.isnan(a)]) # isnan检测数组的非数字元素 ~取补运算符
a = np.array([1, 2+6j, 2, 5J, 5])
```

```
print(a[np.iscomplex(a)]) # iscomplex检测数组的复数元素

# 花式索引 根据索引数组的值作为目标数组的某个轴的下标来取值
x = np.arange(9)
print(x)
y = x[[0, 6]]
print(y)
print(y[0])
print(y[1])

# 二维数组
x = np.arange(32).reshape((8,4))
print(x)
print(x[[4, 2, 1, 7]])
print(x[[-4, -2, -1, -7]])
"""
np.ix_ 输入两个数组，产生笛卡尔积的映射关系
e.g. A=(a,b) B=(0,1,2)
A * B = {(a,0), (a,1), (a,2), (b,0), (b,1), (b,2)}
B * A = {(0,a), (1,a), (2,a), (0,b), (1,b), (2,b)}
"""
x = np.arange(32).reshape((8,-1))
print(x[np.ix_([1,5,7,2],[0,3,1,2])])
```

广播

```
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([10, 20, 30, 40])
c = a * b
print(c)

# 广播Broadcast 对不同形状的数组进行数值计算的方式
a = np.array([[0, 0, 0], [10, 10, 10], [20, 20, 20], [30, 30, 30]])
b = np.array([0, 1, 2])
print(b.shape)
print(a + b) # 将b延伸至与a维度大小相同

# np.tile(a, reps) reps：对应的英文单词为repeats，list表示，reps表示对A的各个axis进行重复的次数
bb = np.tile(b, (4, 1))
print(bb)
print(a + bb)
"""
广播触发机制 两个数组a b
两个矩阵在一个维度上数据宽度相同 但在另一个维度上数据宽度不同
并且形状小的矩阵 在数据宽度不同的的这一维度只有一个元素
e.g.a.shape=(4,3)而b.shape=(1,3)，两个矩阵axis=1的数据宽度是相同的，但是axis=0的数据宽
```

```
度不一样，
并且b.shape[0]=1，这就是广播机制的触发条件，numpy会把b沿axis=0的方向复制4份，即形状变成(4, 3)，与a的一致，接下来就是对应位相加即可
"""
```

迭代数组

```
import numpy as np
# 迭代器np.nditer 可以完成对数组元素的访问

a = np.arange(6).reshape(2, 3)
print(a)
print("迭代输出数组：")
for x in np.nditer(a):
    print(x, end=' ')
print('\n----------------------')
for x in np.nditer(a.T):
    print(x, end=' ')
print('\n----------------------')
for x in np.nditer(a.T.copy(order='C')):
    print(x, end=' ')
# a和a.T的遍历顺序一样，是因为选择的顺序和数组a的内存布局一样（行序优先C） 但a.T.copy(order='C')的遍历结果不一样，因为指定了a.T的行序优先

# 在copy中控制遍历排序
print('\n----------------------')
print("以C风格顺序排序")
b = a.T.copy(order='C')
for x in np.nditer(b):
    print(x, end=' ')
print('\n----------------------')
print("以F风格顺序排序")
b = a.T.copy(order='F')
for x in np.nditer(b):
    print(x, end=' ')

# 在nditer中控制遍历排序
print('\n----------------------')
print("以C风格顺序排序")
for x in np.nditer(a.T, order='C'):
    print(x, end=' ')
print('\n----------------------')
print("以F风格顺序排序")
for x in np.nditer(a.T, order='F'):
    print(x, end=' ')

# nditer对象默认将遍历的数组视为只读对象，设置参数op_flags可以在遍历同时，对数组进行修改，指定readwrite或writeonly模式
print('修改之前的数组：')
print(a)
for x in np.nditer(a, op_flags=['readwrite']):
```

```
    # * 使用x = 2 + x是无法完成对数组a的修改操作的。因为直接赋值操作将会创建一个新的数组，而不会修改原始的数组 a。
    # * 要修改原始数组 a 中的值，需要使用 x[...] = 2 + x 这种形式的赋值语句，以确保对a进行原地修改。
    x[...] = 2 + x
print('修改之后的数组：')
print(a)

# flags参数： c_index 跟随C顺序的索引 f_index 跟随Fortan顺序的索引 multi_index 每次迭代可以跟踪一种索引类型 external_loop 给出的值是具有多个值的一维数组而不是零维数组
for x in np.nditer(a, flags = ["external_loop"], order = 'F'):
    print(x, end=' ')

# 广播迭代 a(3,4) b(1,4)
a = np.arange(0, 60, 5).reshape(3, 4)
b = np.array([1, 2, 3, 4], dtype=int)
print('\on')
print(a)
print(b)
for x, y in np.nditer([a, b]):
    print(f"{x},{y}", end=' ')
```

数组操作

```
import numpy as np

# todo 修改数组形状
# reshape(a, newshape, order='C') 不改变数据的条件下修改形状
a = np.arange(8)
b = a.reshape(4,2)
print(a)
print(b)

# flat 数组元素迭代器
a = np.arange(9).reshape(3 ,3)
print('------------------------------')
print(a)
for row in a:
    print(row)
for element in a.flat:
    print(element)

# numpy.ndarray.flatten(order='C') 数组降维，返回折叠后的一维数组，原数组不变
a = np.arange(8).reshape(2, 4)
print('------------------------------')
print(a)
print(a.flatten())
print(a.flatten(order='F'))

# ravel(a, order='C') 展平的数组元素 返回数组视图 修改影响原数组 C按行 F按列 A原顺序 K元素在内存的出现顺序
```

```
a = np.arange(8).reshape(2, 4)
print('----------------------------')
print(a)
print(a.ravel())
print(a.ravel(order='F'))

# todo 翻转数组
# transpose(a, axes) 对换数组维度
a = np.arange(12).reshape(3, 4)
print('----------------------------')
print(a)
print(np.transpose(a))
print(a.T)

# rollaxis(a, axis, start) 函数向后滚动特定的轴到一个特定位置 axis 向后滚动的轴 axis滚动到start轴前面 其他轴相对位置不变
a = np.arange(8).reshape(2, 2, 2)
print('----------------------------')
print(a)
print(np.where(a==6))
print(a[1, 1, 0])
# 将轴2滚动到轴0
b = np.rollaxis(a, 2, 0)
print(b)
print(np.where(b==6))
# 将轴2滚动到轴1
c = np.rollaxis(a, 2, 1)
print(c)
print(np.where(c==6))
'''
>>> a = np.ones((3,4,5,6))
>>> np.rollaxis(a, 3, 1).shape
(3, 6, 4, 5)
>>> np.rollaxis(a, 2).shape
(5, 3, 4, 6)
>>> np.rollaxis(a, 1, 4).shape
(3, 5, 6, 4)

三维数组array(轴0，轴1，轴2),将轴2滚动到轴0位置，其余轴顺序不变，即new_array(轴2，轴0，轴1)
元素[1,1,0]     [0,1,1]
'''


print('----------------------------')
# .swapaxes(ax1, ax2) 将数组的n个维度的2个维度调换 返回新数组 类似转置
a = np.array([[1, 2, 3, 4, 5], [2, 3, 4, 5, 6]])
print(a)
b = a.swapaxes(0, 1)
print(b)

# todo 修改数组维度
print('----------------------------')
# broadcast 模仿广播的对象 返回一个对象 该对象封装了将一个数组广播到另一个数组的结果
x = np.array([[1], [2], [3]])
```

```
y = np.array([4, 5, 6])
print(x.shape, y.shape)
b = np.broadcast(x, y) # 对y广播x
r, c = b.iters # 自带迭代器属性
print(next(r), next(c))
print(next(r), next(c))
print(f'广播对象的形状{b.shape}')
b = np.broadcast(x, y)
c = np.empty(b.shape)
print(c.shape)
c.flat = [u + v for (u, v) in b]
print(c)
print(x + y)

print('------------------------------')
# broadcast_to(a, shape, subok) 将数组广播到新形状 在原始数组返回只读视图 若新形状不符合广播规则, 则error
a = np.arange(4).reshape(1, 4)
print(a)
print(np.broadcast_to(a, (4,4)))

print('------------------------------')
# expand_dims(a, axis) 在指定位置插入新的轴扩展数组形状
a = np.array([[1, 2], [3, 4]])
print(a)
print(np.expand_dims(a, axis=0))
print(np.expand_dims(a, axis=0).shape, a.shape)
print(np.expand_dims(a, axis=1))
print(a.ndim, np.expand_dims(x, axis=1).ndim)
print(np.expand_dims(a, axis=1).shape, a.shape)

print('------------------------------')
# squeeze(a, axis) 从给定数组的形状中删除一维的条目
a = np.arange(9).reshape(3, 1, 3)
print(a)
print(np.squeeze(a))
print(a.shape, np.squeeze(a).shape)

print('------------------------------')
# todo 连接数组
# np.concatenate((a1, a2, ...), axis=0) 两个或多个数组合并成一个新的数组
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
x = np.concatenate((a, b))
print(x)
x = np.concatenate((a, b), 1)
print(x)

print('------------------------------')
# stack(a, axis) 沿着新的轴堆叠数组序列
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
print(np.stack((a, b), 0))
print(np.stack((a, b), 1))
```

```
print(np.stack((a, b), 2))

print('------------------------------')
# hstack((a1, a2, ...)) 水平堆叠
print(np.hstack((a, b)))
print('------------------------------')
# vstack((a1, a2, ...)) 垂直堆叠
print(np.vstack((a, b)))

print('------------------------------')
# todo 分割数组
# split(a, indices_or_sections, axis) 沿特定的轴将数组分割成子数组
# indices_or_sections:若为整数，用该数平均切分；若为数组，为沿轴切分的位置（左开右闭）
# axis：沿着哪个方向切分 默认0 横向切分 为1纵向切分
a = np.arange(9)
print(a)
print(np.split(a, 3))
print(np.split(a, [4, 7]))
a = np.arange(16).reshape(4, 4)
print(a)
print(np.split(a, 2))
print(np.split(a, 2, 1))

print('------------------------------')
# hsplit(a, num) 水平分割数组 num:要返回的相同形状的数组数量
a = np.arange(16).reshape(4, 4)
print(a)
print(np.hsplit(a, 2))

print('------------------------------')
# vsplit 垂直分割数组
print(a)
print(np.vsplit(a, 2))

print('------------------------------')
# todo 数组元素的添加与删除
# resize(a, shape) 修改原数组
a = np.arange(16).reshape(4, 4)
print(a)
print(np.resize(a, (2,8)))

print('------------------------------')
# append(a, values, axis=None) None 横向加成返回一维数组
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
print(np.append(a, [7, 8, 9]))
print(np.append(a, [[7, 8, 9]], 0))
print(np.append(a, [[7, 8, 9], [10, 11, 12]], 1))

print('------------------------------')
# insert(a, obj, values, axis) obj:在其之前插入值的索引 axis:未传入值则返回一维数组
a = np.array([[1, 2], [3, 4], [5, 6]])
print(a)
print(np.insert(a, 3, [11, 12]))
```

```
print(np.insert(a, 1, [11], axis=0)) # 轴0广播
print(np.insert(a, 1, 11, axis=1)) # 轴1广播

print('------------------------------')
# delete(a, obj, axis) obj 删除的数字或数组
a = np.arange(12).reshape(3, 4)
print(a)
print(np.delete(a, 5))
print(np.delete(a, 1, axis=1))
print(np.delete(a, np.s_[::2]))

print('------------------------------')
# unique(a, return_index, return_inverse, return_counts) 去除数组中的重复元素
# return_index True 返回新列表元素在旧列表中的位置（下标），并以列表形式存储
# return_inverse True 返回旧列表元素在新列表中的位置（下标），并以列表形式存储
# return_counts True 返回去重数组中的元素在原数组的出现次数
a = np.array([1, 1, 1, 2, 2, 3, 4, 5, 5, 5, 5, 6, 7, 7, 8, 9])
print(a)
print(np.unique(a, return_index=True))
print(np.unique(a, return_inverse=True))
print(np.unique(a, return_counts=True))


# .astype(new_type) 转化数据类型 创建新数组
print(a)
x = a.astype(np.float64)
print(x)

# .tolist() ndarray数组向列表转换
x = a.tolist()
print(type(x))
```

位运算

```
import numpy as np

# 位运算
a1 = np.array([True, False, True], dtype=bool)
a2 = np.array([False, True, False], dtype=bool)

result_and = np.bitwise_and(a1, a2) # 按位与
result_or = np.bitwise_or(a1, a2) # 按位或
result_xor = np.bitwise_xor(a1, a2) # 按位异或
result_not = np.bitwise_not(a1) # 按位取反

print('and:', result_and)
print('or:', result_or)
print('xor:', result_xor)
print('not', result_not)

# invert 按位取反 1 00000001 2 00000010    补码取反 -1 11111110 转原码 10000010  补
```

```
码取反 11111101 转原码 10000011
print('Invert:', np.invert(np.array([1, 2], dtype=np.int8)))

# left_shift 左移位运算
print(bin(5),bin(np.left_shift(5, 2)))
print('Left shift:', np.left_shift(5, 2)) # 00101 -> 10100

# right_shift 右移位运算
print(bin(10),bin(np.left_shift(10, 1)))
print('Right shift', np.right_shift(10, 1)) # 01010 -> 00101

'''
操作符运算
& 与运算
| 或运算
^ 异或运算
~ 取反运算
<< 左移运算
>> 右移运算
'''
```

字符串函数

```
import numpy as np

# char.add() 两个数组的字符串连接
print(np.char.add('hello', 'world'))
print(np.char.add(['hello', 'hi'], ['world', 'nihao~']))

# char.multiply(a, num) 执行多重连接 num重复次数
print(np.char.multiply('Iven', 5))

# char.center(str, width, fillchar) 将字符串居中，指定字符在左侧和右侧进行填充 width:
填充后整体长度 fillchar:填充字符
print(np.char.center('Iven', 10, fillchar='*'))

# char.capitalize() 将字符串的第一个字母转换大写
print(np.char.capitalize('rosennn'))

# char.title() 对数组的每个单词的第一个字母转为大写
print(np.char.title('rosenn enjoys surfing'))

# char.lower() 对数组的每个元素转换小写，对每个元素调用str.lower
print(np.char.lower('IVEN'))

# char.upper() 对数组的每个元素转换大写，对每个元素调用str.upper
print(np.char.upper(['iven', 'rosenn']))

# char.split(str, sep) 指定分隔符对字符串进行分割，返回数组 默认分隔符是空格
print(np.char.split('i like coding'))
print(np.char.split('www.github.com'), sep='.')
```

```
# char.splitlines() 以换行符作为分隔符来分割字符串，返回数组  \r\n都可以作为换行符
print(np.char.splitlines('Iven\nlikes it'))
print(np.char.splitlines('Iven\rlikes it'))

# char.strip() 移除开头或结尾的特定字符
print(np.char.strip('abbbbacc', 'a'))

# char.join() 通过指定分隔符来连接数组中的元素或字符串
print(np.char.join([':', '-'], ['Iven', 'Starry']))

# char.replace(str, old, new) 使用新字符串替换字符串的所有子字符串
print(np.char.replace('i like coffee', 'ff', 'fffff'))

# char.encode() 对数组中每个元素都调用str.encode函数进行编码,默认编码UTF-8
print(np.char.encode('Iven','cp500')) # cp500是编码类型
print(np.char.encode('Iven','ascii')) # ascii是编码类型
print(np.char.encode('Iven'))         # 默认是uft-8编码

# char.decode() 对编码元素进行str.decode()解码
# char.encode() 对数组中每个元素都调用str.encode函数,默认编码UTF-8
a = np.char.encode('Iven','cp500')
b = np.char.encode('Iven','ascii')
c = np.char.encode('Iven')
print(np.char.decode(a,'cp500')) # cp500是编码类型
print(np.char.decode(b,'ascii')) # ascii是编码类型
print(np.char.decode(c))         # 默认是uft-8编码
```

数学函数

```
import numpy as np

# sin cos  tan arccos arcsin arctan
a = np.array([0, 30, 45, 60, 90])
print(np.sin(a * np.pi / 180))
print(np.cos(a * np.pi / 180))
print(np.tan(a * np.pi / 180))
# np.degrees(将弧度转换为角度)
print(np.degrees(np.arcsin(np.sin(a * np.pi / 180))))
print(np.degrees(np.arccos(np.cos(a * np.pi / 180))))
print(np.degrees(np.arctan(np.tan(a * np.pi / 180))))

# around(a, decimals) 返回四舍五入值 decimals:舍入的位数 默认0  如果为负数，整数将四舍五入到小数点左侧的位置
a = np.array([1.0, 5.55, 123, 0.567, 25.532])
print(a)
print(np.around(a))
print(np.around(a, decimals=1))
```

```
print(np.around(a, decimals=-1))
print(np.around(a, decimals=-2))

# floor 返回小于或等于表达式的最大整数 向下取整
# ceil  返回大于或等于表达式的最小整数 向上取整
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])
print(np.floor(a))
print(np.ceil(a))
```

算术函数

```
import numpy as np

# 加减乘除 add subtract multiply divide
a = np.arange(9, dtype=np.float64).reshape(3, 3)
b = np.array([10, 10, 10])
print(np.add(a, b))
print(np.subtract(a, b))
print(np.multiply(a, b))
print(np.divide(a, b))

# reciprocal() 返回参数各元素的倒数
a = np.array([0.25, 1.33, 1, 100])
print(a)
print(np.reciprocal(a))

# power() 将第一个输入数组中的元素作为底数，计算它与第二个输入数组中相应元素的幂 x^n
a = np.array([10, 100, 1000])
print(a)
print(np.power(a, 2))
b = np.array([1, 2, 3])
print(np.power(a, b))

# np.mod np.remainder 计算输入数组中相应元素的相除后的余数
a = np.array([10, 20, 30])
b = np.array([3, 5, 7])
print(np.mod(a, b))
print(np.remainder(a, b))


a = np.arange(24).reshape((2, 3, 4))
print(f"平方运算:{np.square(a)}")
print(f"开方运算: {np.sqrt(a)}")
print(f"整数小数分离{np.modf(np.sqrt(a))}") # np.modf()将整数和小数部分分成两个部分

b = np.sqrt(a)
print(a)
print(b)
print(np.maximum(a, b)) # 输出结果浮点数
print(a > b)
```

统计函数

```
import numpy as np

print('--------------amin amax----------------')
# amin amax 沿指定轴找最大最小值
'''
amin amax(a, axis=None, out=None, keepdims=<no value>, initial=<no value>, where=
<no value>)
axis：在哪个轴上计算最大最小值
out：指定结果存储位置
keepdims：True将保持结果数组的维度数目与输入数组相同 False 去除计算后维度为1的轴
initial：指定一个初始值，然后在数组的元素上计算最大最小值
where：布尔数组 指定只考虑只满足条件的数组
'''
a = np.array([[3, 7, 5],[8, 4, 3], [2, 4, 9]])
print(np.amin(a))
print(np.amin(a, 1))
print(np.amin(a, 0))
print(np.amax(a))
print(np.amax(a, 1))
print(np.amax(a, 0))

print('--------------ptp----------------')
# ptp(a) a中元素最大值和最小值之差 参数选择同上
print(np.ptp(a))
print(np.ptp(a, axis=1))
print(np.ptp(a, axis=0))

print('--------------percentile----------------')
# percentile(a, q, axis) 表示小于这个值的观察值的百分比 q计算的百分位数0-100 该值=(最
大值-最小值)*p + 最小值
print(np.percentile(a, 50))
print(np.percentile(a, 50, axis=1))
print(np.percentile(a, 50, axis=0))
print(np.percentile(a, 50, axis=0, keepdims=True))

print('--------------median----------------')
# median(a) 元素中位数
print(np.median(a))
print(np.median(a, axis=0))
print(np.median(a, axis=1))

print('--------------mean----------------')
# mean(a, axis=None) 计算数组a的axis轴所有元素期望 默认全部轴
print(np.mean(a))
print(np.mean(a, axis=0))
print(np.mean(a, axis=1))

print('--------------average----------------')
# average(a, axis=None, weights=None, returned=False) 计算数组a的axis轴所有元素加权
平均值 默认全部轴  若为True同时返回加权平均值和权重综合
```

```
print(np.average(a, axis=0, weights=[10, 5, 1]))
print(np.average(a, axis=0, weights=[10, 5, 1], returned=True))

print('--------------std----------------')
# std(a, axis=None) 计算数组a的axis轴所有元素标准差 默认全部轴
# std = sqrt(mean((x - x.mean()) ** 2))
print(np.std(a))

print('--------------var----------------')
# var(a, axis=None) 计算数组a的axis轴所有元素方差 默认全部轴
print(np.var(a))

print('--------------sum----------------')
# sum(a, axis=None) 计算数组a的axis轴所有元素之和 默认全部轴
print(np.sum(a))

print('--------------min max----------------')
# min(a) max(a) 最小值最大值
print(np.min(a), np.max(a))
```

排序与条件筛选函数

```
import numpy as np

# sort(a, axis, kind, order) 返回输入数组的排序副本 kind 默认快速排序 order如果数组包含字段则是要排序的字段
a = np.array([[3, 7], [9, 1]])
print(a)
print(np.sort(a))
print(np.sort(a, axis=0))
# msort(a) 数组第一个轴排序 等于sort(a, axis=0) numpy2.0被删除
# // print(np.msort(a))

dt = np.dtype([('name', 'S10'), ('age', int)])
a = np.array([('Iven', 22), ('Rosenn', 21), ('bob', 23), ('starry', 19)], dtype=dt)
print(a)
print(np.sort(a, order='name')) # 注意大小写ascii码不同

# argsort 返回数组值从小到大的索引值
a = np.array([3, 1, 2])
print(a)
print(np.argsort(a))
print(a[np.argsort(a)]) # 重构原数组
for i in np.argsort(a) :
    print(a[i], end=' ') # 循环重构原数组
print('\n')

# lexsort 对多个序列进行排序 优先排靠后的一列 在这里即a 排完a相同再排b
```

```
a = np.array(['Iven', 'Rosenn', 'Iven', 'starry'])
b = ('s', 's', 'f', 'f')
ind = np.lexsort((b, a))
print(ind)
print([a[i] + ', ' + b[i] for i in ind])

# sort_complex 复数排序 现排实部后虚部 从小到大
a = np.array([1+2j, 1-2j, 1, 2+1j, 1+3j])
print(np.sort_complex(a))

# partition(a, kth[, axis, kind, order]) 指定一个数，对数组进行分区
a = np.array([232, 564, 278, 3, 2, 1, -1, -10, -30, -40])
print(np.partition(a, 4)) # 指定排序后的数组索引为3的数 比这个数小的排这个数前 比这个数大的排后面
print(np.partition(a, (1, 3))) # 小于1(-30)在前面，大于3(-1)的在后面，1和3之间的在中间，顺序无所谓

# argmin(a) argmax(a) 默认求a中元素最小值、最大值的降到一维后下标 指定axis求在该轴的下标索引
a = np.array([[3, 7], [9, 1]])
print(np.argmin(a), np.argmax(a))
print(np.argmin(a, axis=0), np.argmin(a, axis=1), np.argmax(a, axis=0), np.argmax(a, axis=1))

# nonzero() 返回输入数组中非零元素的索引
a = np.array([[30, 40 ,0], [0, 20, 10], [50, 0, 60]])
print(np.nonzero(a))

# where() 返回输入数组中满足给定条件的元素索引
a = np.arange(9.).reshape(3, 3)
print(a)
print(np.where(a > 3)) # 返回索引
print(a[np.where(a > 3)]) # 利用索引获取元素

# extract() 根据某个条件从数组中抽取元素，返回满条件的元素
condition = np.mod(a, 2) == 0
print(condition)
print(np.extract(condition, a))

# unravel_index(index, shape) 根据shape将一维下标index转多维下标
print(np.unravel_index(np.argmax(a), a.shape))
```

随机数函数

```
import numpy as np
# rand() 随机数数组 浮点数 符合均匀分布 [0, 1)
a = np.random.rand(3, 4, 5)
print(a)

# randn() 标准正态分布
```

```
a = np.random.randn(3, 4, 5)
print(a)

# randint(low, high, shape) 随机整数数组 [low, high)
# seed() 使用同样的seed种子 生成的随机数数组相同
np.random.seed(10)
a = np.random.randint(100, 200, (3, 4))
print(a)
np.random.seed(10)
b = np.random.randint(100, 200, (3, 4))
print(b)

# shuffle 数组的第一轴进行随即变换 改变原数组
np.random.shuffle(a)
print(a)
np.random.shuffle(a)
print(a)

# permutation() 数组的第一轴进行随机变换 不改变原数组 生成新数组
b = np.random.permutation(a)
print(b)

# choice(a, size, replace, p) 从一维数组中以概率p抽取元素 形成size形状新数组 replace
是否重用新元素 默认True值
a = np.random.randint(10, 20, (8,))
print(a)
print(np.random.choice(a, (3,2)))
print(np.random.choice(a, (3,2), replace=False))
print(np.random.choice(a, (3,2), p=a/np.sum(a)))

# uniform(low, high, size) 均匀分布数组
print(np.random.uniform(0, 10, (3, 4)))
# normal(loc, scale, size) 正态分布数组 loc均值 scale标准差
print(np.random.normal(10, 5, (3, 4)))
# poisson(lam, size)       柏松分布数组 lam随机事件发生率
print(np.random.poisson(0.5, (3, 4)))
```

梯度函数

```
import numpy as np
'''
gradient(f) 计算数组f元素的梯度 当f多维 返回每一个维度梯度
一维数组：存在俩侧值 斜率=（右侧值-左侧值）/ 2
只存在一侧值 斜率=（本身-左侧值） 或者 （右侧值-本身）
二维数组：存在两个梯度值 横向一个 纵向一个 输出两个数组 第一个数组输出最外层维度梯度 第二
个数组输出第第二层维度梯度
'''

a = np.random.randint(0, 20, (5))
print(a)
print(np.gradient(a))
```

```
b = np.random.randint(0, 50, (3, 5))
print(b)
print(np.gradient(b))
```

字节交换

```
import numpy as np

# np.adarray.byteswap() 将ndarray每个元素的字节进行大小端转换
a = np.array([1, 245, 8755], dtype=np.int16)
print(a)
print(list(map(hex, a))) # hex 16进制编码
print(a.byteswap(inplace=True)) # 传入True原地交换
print(list(map(hex, a)))
'''
大端模式
1: 0000 0000 0000 0001
245: 0000 0000 1111 0101
8755: 0010 0010 0011 0011

小端模式
1: 0000 0001 0000 0000
245: (首1补码)1111 0101 0000 0000 (转原码表示)1000 1011 0000 0000
8755: 0011 0011 0010 0010
'''
```

副本和视图

类型	id是否相同	修改新数据，原数据是否变化
赋值	相同	变化
view 创建新的数组对象	不同	不变
切片	不同	变化
copy副本	不同	不变

```
import numpy as np

'''
副本：数据的完整的拷贝。对拷贝修改不影响原数据，物理内存不在一起
视图：数据的别称或引用，通过这个别称或者引用即可访问、操作，不产生拷贝，修改视图影响原数
据，物理内存在同一位置

副本发生在：
1.python序列的切片操作，调用deepcopy()函数
```

```
2.调用ndarray.copy()

视图发生在:
1.numpy切片操作返回原数据的视图
2.调用ndarray.view()
'''

print('----------------------------------------')
# 赋值不创建副本 id相同，修改赋值对象，也会修改原数据
a = np.arange(6)
print(a)
print(id(a)) # id() 返回对象的“标识值”。该值是一个整数，在此对象的生命周期中保证是唯一且恒定的,类似于指针
b = a
print(b)
print(id(b))
b.shape = 3,2
print(b)
print(a)

print('----------------------------------------')
# 视图或浅拷贝
# view 创建新的数组对象 id不同，修改新的数组对象，不修改原数据
a = np.arange(6).reshape(3, 2)
print(a)
b = a.view() # view() 创建一个新的数组对象。维度变化并不改变原始数据的维度
print(b)
print(id(a))
print(id(b))
b.shape = 2,3
print(b) # 修改b形状 并不修改a
print(a)

print('----------------------------------------')
# 切片仍是原数据的视图 id不同 视图指向原数据，修改切片原数据也修改
arr = np.arange(12)
print(a)
a = arr[3:]
b = arr[3:]
a[1] = 123
b[2] = 234
print(arr) # 切片创建视图 修改数据会影响到原始数组
print(id(a), id(b), id(arr[3:]))
# a, b 都是arr的一小部分视图。对视图的修改直接反映到原数据中，但ab的id不同，视图虽然指向原数据，但与赋值引用有区别

print('----------------------------------------')
# 副本或深拷贝 id不同 修改拷贝原数据不修改
a = np.array([[10, 10], [2, 3], [4, 5]])
print(a)
b = a.copy() # copy函数创建一个副本，副本修改不影响原始数据，物理内存不在同一位置
print(b)
print(id(a), id(b))
b[0, 0] = 1
```

```
print(a)
print(b)
```

矩阵库

```
import numpy.matlib
import numpy as np
# numpy包含矩阵库numpy.matlib，模块返回一个矩阵，而非ndarray对象

# 转置矩阵
a = np.arange(12).reshape(3, 4)
print(a)
print(a.T)
print(type(a), type(a.T))

# matlib.empty 返回一个新的矩阵
a = numpy.matlib.empty((2, 2))
print(a)
print(type(a))

# matlib.zeros 返回全0矩阵
a = numpy.matlib.zeros((2, 2))
print(a)

# matlib.ones 创建全1矩阵
a = numpy.matlib.ones((2, 2))
print(a)

# matlib.eye(row, col, k, dtype) 创建对角线为1其他位置为0的矩阵  k:对角线的索引
a = numpy.matlib.eye(3, 3, k=0, dtype=float)
print(a)
a = numpy.matlib.eye(3, 3, k=1, dtype=float)
print(a)

# matlib.identity() 返回给定大小的单位矩阵
a = numpy.matlib.identity(3, dtype=float)
print(a)

# matlib.rand 随机数矩阵
a = numpy.matlib.rand(3, 3)
print(a)

# 矩阵总是二维的，而ndarray是一个n维数组，二者可以互换
a = np.matrix('1, 2; 3, 4')
print(a, type(a))
b = np.asarray(a) # asarray: 矩阵转ndarray
print(b, type(b))
c = np.asmatrix(b) # asmatrix: adarray转矩阵
print(c, type(c))
```

线性代数

```
import numpy as np

# numpy提供线性代数函数库linalg
print('-----------------dot----------------------')
# dot() 矩阵点积
a = np.array([[1, 2], [3, 4]])
b = np.array([[11, 12], [13, 14]])
print(np.dot(a, b))
# [[1*11+2*13, 1*12+2*14],[3*11+4*13, 3*12+4*14]]

print('-----------------vdot----------------------')
# vdot 向量点积 将输入参数平摊为一维向量
print(np.vdot(a, b))
# 1*11 + 2*12 + 3*13 + 4*14 = 130

print('-----------------inner----------------------')
# inner 返回一维数组的向量内积 对于更高维度，返回最后一个轴上和的乘积
print(np.inner(np.array([1, 2, 3]), np.array([0, 1, 0])))
# 等价于 1*0+2*1+3*0
a = np.array([[1, 2], [3, 4]])
b = np.array([[11, 12], [13, 14]])
print(np.inner(a, b))
# 1*11+2*12, 1*13+2*14
# 3*11+4*12, 3*13+4*14

print('-----------------matmul----------------------')
# matmul 返回两个数组的矩阵乘积
# 二维数组就是矩阵乘法
a = [[1, 0], [0, 1]]
b = [[4, 1], [2, 2]]
print(np.matmul(a, b))
# 若一个参数的维度为1维度，则通过在其维度加1提升为矩阵，运算后去除
a = [[1, 0], [0, 1]]
b = [1, 2]
print(np.matmul(a, b))
print(np.matmul(b, a))
# 若任一参数维度大于2，则另一参数进行广播，输出多个矩阵
a = np.arange(8).reshape(2, 2, 2)
b = np.arange(4).reshape(2, 2)
print(np.matmul(a, b))

print('-----------------linalg.det----------------------')
# linalg.det() 计算输入矩阵的行列式
a = np.array([[1, 2], [3, 4]])
print(np.linalg.det(a))

print('-----------------linalg.solve、inv----------------------')
# linalg.solve() 给出矩阵的线性方程解 linalg.inv()计算逆矩阵
a = np.array([[1, 2], [3, 4]])
b = np.linalg.inv(a)
```

```
print(a, b)
print(np.dot(a, b)) # AB=E

# 计算AX=B的解 先求A的逆 X=A^(-1) B
a = np.array([[1, 1, 1], [0, 2, 5], [2, 5, -1]])
print(a)
a_inv = np.linalg.inv(a)
print(a_inv)
b = np.array([[6], [-4], [27]])
print(b)
print('解方程AX=B')
print(np.linalg.solve(a, b))
print('计算:X = A^(-1) B:')
print(np.dot(a_inv, b))
```

IO

```
'''
CSV文件：逗号分隔值文件
np.savetxt(frame, array, fmt='%.18e', delimiter=None)
frame:文件、字符串或产生器 可以是.gz .bz2的压缩文件
array:存入文件的数组
fmt:存入文件的格式 %d %.2f %.18e
delimiter:分割字符串。默认为任何空格
'''
# 整数保存
import numpy as np
a = np.arange(100).reshape(5, 20)
np.savetxt('related_data/test1.csv', a, fmt='%d', delimiter=',')
# 浮点数保存
a = np.arange(100).reshape(5, 20)
np.savetxt('related_data/test2.csv', a, fmt='%.1f', delimiter=',')

'''
np.loadtxt(frame, dtype=np.float, delimiter=None, unpack=False)
frame:文件、字符串或产生器 可以是.gz .bz2的压缩文件
dtype:数据类型可选
delimiter:分割字符串。默认为任何空格
unpack:若为True，读入属性将分别写入不同变量
'''
# 默认浮点型
b = np.loadtxt('related_data/test1.csv', delimiter=',')
print(b)
# 指定整数型
# * 新版numpy无int类型 使用int_
b = np.loadtxt('related_data/test1.csv', dtype=np.int_, delimiter=',')
print(b)

# csv只能有效存储一维和二维数组即load/save函数只能存取一维和二维数组

'''
```

```
a.tofile(frame, sep='', format='%s')
frame: 文件、字符串
sep: 数据分割字符串 如果为空串 写入文件为二进制
format: 写入数据格式
'''
import numpy as np

a= np.arange(100).reshape(5, 10, 2)
# 只是逐项输出数据 看不出维度信息
a.tofile("related_data/test1.dat", sep=',', format='%d')
# 存储为二进制文件 占用空间更小 如果可以知道显示字符的编码以及字节之间的关系就可以实现转化
a.tofile("related_data/test2.dat", format='%d')

'''
np.fromfile(frame, dtype=float, count=-1, sep='')
frame:文件、字符串
dtype:读取的数据类型
count:读入元素个数,-1表示读入全部文件
sep: 数据分割字符串 如果为空串 写入文件为二进制

读取时需要知道存入文件时数组的维度和元素类型
可以通过再写一个元数组文件存储维度和元素类型信息
'''
# 文本文件
b = np.fromfile('related_data/test1.dat', dtype=np.int_, sep=',')
print(b)
b = np.fromfile('related_data/test1.dat', dtype=np.int_, sep=',').reshape(5, 10, 2)
print(b)

# 二进制文件 无需指定分隔符
c = np.fromfile("related_data/test2.dat", dtype=np.int_).reshape(5, 10 ,2)
print(c)

'''
NumPy便捷文件存取 固定文件格式
正常文件: np.save(fname, *args)   数组保存在npy文件
压缩: np.savez(fname, *args, **kwds) 存储多个数组保存在npz扩展文件

fname: 文件名 以.npy为扩展名 压缩扩展名为.npz
args: 保存的数组
kwds" 用关键字参数为数组起名，非关键字自动起名arr_0 arr_1

np.load(fname)
fname: 文件名 以.npy为扩展名 压缩扩展名为.npz

可以直接还原数组维度和元素类型信息
因为在文件开头用显示的方式 将数组的源信息存储在了第一行
'''
np.save("related_data/test1.npy", a)
d = np.load("related_data/test1.npy")
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.arange(0, 1.0, 0.1)
```

```
c = np.sin(b)
np.savez('related_data/test.npz', a, b, sin_array=c)
r = np.load('related_data/test.npz')
print(r.files) # 查看数据名称
print(r['arr_0'])
print(r['arr_1'])
print(r['sin_array'])
```

图像的数组表示

图像是一个由像素组成的二维矩阵，每个元素是一个RGB值

```
from PIL import Image
import numpy as np

im = np.array(Image.open("related_data/dog.jpg"))
# 图像是一个三维数组，维度分别是高度、宽度和像素RGB值
print(im.shape, im.dtype)
```

图像的变换

```
from PIL import Image
import numpy as np

# 像素值翻转
a = np.array(Image.open("related_data/dog.jpg"))
print(a.shape, a.dtype)
b = [255, 255, 255] - a
# Image.fromarray 数组转图像Image对象
im = Image.fromarray(b.astype('uint8'))
im.save("related_data/dog_reverse_trans.jpg")

# .convert('L') 将彩色图片转换灰度值图片 灰度值翻转
a = np.array(Image.open("related_data/dog.jpg").convert('L'))
b = 255 - a
im = Image.fromarray(b.astype('uint8'))
im.save('related_data/dog_gray_trans.jpg')

# 区间变换
c = (100/255)*a + 150
im = Image.fromarray(c.astype('uint8'))
im.save('related_data/dog_interval_trans.jpg')

# 像素平方
d = 255 * (a/255) ** 2
im = Image.fromarray(d.astype('uint8'))
im.save('related_data/dog_square_trans.jpg')
```

实例_图像的手绘效果

```
# 手绘效果：黑白灰色、边界线条较重、相同或相近色彩趋于白色、略有光源效果
from PIL import Image
import numpy as np

a = np.array(Image.open('related_data/dog.jpg').convert('L')).astype('float')

# 调整图像明暗和添加虚拟深度
depth = 10. # 预设深度值为10 取值范围0-100
grad = np.gradient(a)
grad_x, grad_y = grad
grad_x = grad_x * depth / 100. # 根据深度调整x和y方向的梯度值 除以100进行归一化
grad_y = grad_y * depth / 100.
A = np.sqrt(grad_x**2 + grad_y**2 + 1.) # 构建x和y轴梯度的三维归一化单位坐标系
uni_x = grad_x / A # 单位法向量
uni_y = grad_y / A
uni_z = 1. / A

vec_el = np.pi / 2.2
vec_az = np.pi / 4.
dx = np.cos(vec_el) * np.cos(vec_az) # .cos(vec_el)单位光线在平面上的投影长度
dxdydz光源对xyz三方向的影响程度
dy = np.cos(vec_el) * np.sin(vec_az)
dz = np.sin(vec_el)

# 梯度和光源相互作用，将梯度转化为灰度
b = 255 * (dx * uni_x + dy * uni_y + dz * uni_z)
# 避免数据越界 将生成的灰度值裁剪至0-255区间
b: np.ndarray = b.clip(0, 255)

im = Image.fromarray(b.astype('uint8'))
im.save('related_data/dog_hand_painting.jpg')
```

Matplotlib

Matplotlib

1. 是python的绘图库，使数据图形化，提供多样化的输出格式
2. 可以绘制各种静态、动态、交互式的图表
3. 可以绘制线图、散点图、等高线图、条形图、柱状图、3D图形、甚至是图像动画等

基础图表函数

Pyplot

```
'''
Pyplot是Matplotlib的子库，提供了和matlab类似的绘图API
```

```
是常用的绘图模块，包含一系列绘图函数的相关函数，可以对图形进行修改
'''

import matplotlib.pyplot as plt
import numpy as np

# 利用两点绘制一条线
xpoints = np.array([0, 6])
ypoints = np.array([0, 100])
plt.plot(xpoints, ypoints)
plt.show()

'''
plot() 绘制线图和散点图
画单条线 plot([x], y. [fmt], *, data=None, **kwargs)
画多条线 plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
x,y: 点或线的节点 x为x轴数据，y为y轴数据，数据可以是列表或数组
fmt: 可选，定义基本格式（颜色、标记和线条样式等）
**kwargs: 可选。用在二维平面图上，设置指定属性，如标签，线的宽度等

标记字符: . 点标记  , 像素标记(极小点)  o 实心圈标记  v 倒三角标记  ^ 上三角标记  > 右三角标记  < 左三角标记等
颜色字符: b m:洋红色 g y r k:黑色 w c '#0080000'RGB颜色符串  多条曲线不指定颜色，会自动选择不同颜色
线型参数: - 实线  -- 破折线  -. 点划线  : 虚线
'''

# 绘制(1, 3)到(8, 10)的线
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints)
plt.show()

# 绘制两个坐标点，而不是一条线，传参o
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints, 'o')
plt.show()

# 绘制一条不规则线
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])
plt.plot(xpoints, ypoints)
plt.show()

# 如果不指定x轴的点。x会根据y的值来设置为0,1,2,3,..N-1
ypoints = np.array([3, 10])
plt.plot(ypoints)
plt.show()

# 若y值更多一点
ypoints = np.array([3, 8, 1, 10, 5, 7])
plt.plot(ypoints)
plt.show()
```

```
# 正弦余弦图 x,y对应正弦 x,z对应余弦
x = np.arange(0, 4 * np.pi, 0.1)
y = np.sin(x)
z = np.cos(x)
plt.plot(x, y, x, z)
plt.show()
```

绘图标记

坐标标记 线类型标记、颜色标记、标记大小与颜色

```
import matplotlib.pyplot as plt
import matplotlib.markers
import numpy as np

# 自定义标记 plot()方法的marker参数
ypoints = np.array([1, 3, 4, 5, 8, 9, 6, 1, 3, 4, 5, 2, 4])
plt.plot(ypoints, marker='o')
plt.show()

# marker定义的符号 具体见学习笔记
plt.plot(ypoints, marker='*')
plt.show()

# 定义下箭头
plt.plot([1, 2, 3], marker=matplotlib.markers.CARETDOWN)
plt.show()
plt.plot([1, 2, 3], marker=7) # 看表也可以用7
plt.show()

# fmt参数 定义了基本格式，如标记、线条样式和颜色
# fmt = '[marker][line][color]'
ypoints = np.array([6, 2, 13, 10])
plt.plot(ypoints, 'o:r') # o实心圆标记 :虚线 r红色
plt.show()

# 标记大小与颜色
'''
markersize:         ms      标记的大小
markerfacecolor:    mfc     定义标记内部的颜色
markeredgecolor:    mec     定义标记边框的颜色
'''
ypoints = np.array([6, 2, 13, 10])
plt.plot(ypoints, marker='o', ms=20, mfc='w', mec='r')
plt.show()
plt.plot(ypoints, marker='o', ms=10, mfc='#4CAF50', mec='#4CAF50') # 自定义颜色
plt.show()
```

绘图线

线的类型

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

# 线的类型
# linestyle 简写 ls
ypoints = np.array([6, 2, 13, 10])
plt.plot(ypoints, linestyle='dotted')
plt.show()
plt.plot(ypoints, ls='-.') # 简写
plt.show()

# 线的颜色
# color 简写 c 同绘图标记颜色
plt.plot(ypoints, color='r')
plt.show()
plt.plot(ypoints, c='#8FBC8F') # 简写
plt.show()
plt.plot(ypoints, c='SeaGreen') # 十六进制颜色名
plt.show()

# 线的宽度
# linewidth 简写 lw  值可以是浮点数
plt.plot(ypoints, linewidth='12.5')
plt.show()

# 多条线
# plot()可以包含多对xy值 绘制多条线
y1 = np.array([3, 7, 5, 9])
y2 = np.array([6, 2, 13, 10])
plt.plot(y1) # 未传入x 默认 0123
plt.plot(y2)
plt.show()
x1 = np.array([0, 1, 2, 3])
x2 = np.array([1, 2, 3, 4]) # 自定义坐标
plt.plot(x1, y1, x2, y2)
plt.show()
```

文本显示

```
import matplotlib.pyplot as plt
import matplotlib.font_manager
import numpy as np

# 设置xy轴的标签 xlabel() ylabel()
x = np.array([1, 2, 3, 4])
y = np.array([1, 4, 9, 16])
plt.plot(x, y)
plt.xlabel('x-label')
```

```
plt.ylabel('y-label')
plt.show()

# 标签 title()
plt.plot(x, y)
plt.xlabel('x-label')
plt.ylabel('y-label')
plt.title('matplotlib test')
plt.show()

# 图形中文显示 fontproperties 可以使用系统的字体 这里使用思源黑体
zhfont1 =
matplotlib.font_manager.FontProperties(fname='related_data/SourceHanSansCN-
Bold.otf')
x = np.arange(1, 11)
y = 2 * x + 5
plt.plot(x, y)
plt.xlabel('x轴', fontproperties=zhfont1)
plt.ylabel('y轴', fontproperties=zhfont1)
plt.title('matplotlib 练习', fontproperties=zhfont1)
plt.show()

# 自定义字体样式 fontdict 设置字体颜色大小
zhfont1 =
matplotlib.font_manager.FontProperties(fname='related_data/SourceHanSansCN-
Bold.otf', size=18)
font1 = {'color':'blue', 'size':20}
font2 = {'color':'darkred', 'size':20}
plt.title('matplotlib 练习', fontproperties=zhfont1, fontdict=font1)
plt.plot(x, y)
plt.xlabel('x轴', fontproperties=zhfont1)
plt.ylabel('y轴', fontproperties=zhfont1)
plt.show()

# 标题与标签的定位
'''
title() 方法提供loc参数设置标题位置 可以设置为'left','right'和'center',默认center
xlabel()方法提供loc参数设置x 轴位置 可以设置为'left','right'和'center',默认center
ylabel()方法提供loc参数设置y 轴位置 可以设置为'bottom','top'和'center',默认center
'''
plt.title('matplotlib 练习', fontproperties=zhfont1, fontdict=font1, loc='right')
plt.plot(x, y)
plt.xlabel('x轴', fontproperties=zhfont1, loc='left')
plt.ylabel('y轴', fontproperties=zhfont1, loc='bottom')
plt.show()

# 任意位置添加文本 text(x, y, str)
a = np.arange(0.0, 5.0, 0.02)
plt.plot(a, np.cos(2 * np.pi * a), 'r--')
plt.xlabel('横轴: 时间', fontproperties='SimHei', fontsize=15, color='green')
plt.ylabel('纵轴: 振幅', fontproperties='SimHei', fontsize=15)
plt.title(r'正弦波实例 $y=cos(2 \pi x)$', fontproperties='SimHei', fontsize=25) #
latex公式
plt.text(2, 1, r'$ \mu = 100 $')
```

```
plt.axis([-1, 6, -2, 2])
plt.grid(True)
plt.show()

# 在图形中增加带箭头的注解 annotate(s, xy=arrow_crd, xytext=text_crd,
arrowprop=dict) xy:箭头指向位置 xytext:文本位置
a = np.arange(0.0, 5.0, 0.02)
plt.plot(a, np.cos(2 * np.pi * a), 'r--')
plt.xlabel('横轴: 时间', fontproperties='SimHei', fontsize=15, color='green')
plt.ylabel('纵轴: 振幅', fontproperties='SimHei', fontsize=15)
plt.title(r'正弦波实例 $y=cos(2 \pi x)$', fontproperties='SimHei', fontsize=25) #
latex公式
plt.annotate(r'$\mu = 100$', xy=(2, 1), xytext=(3, 1.5),
arrowprops=dict(facecolor='black', shrink=0.1, width=2))
plt.axis([-1, 6, -2, 2])
plt.grid(True)
plt.show()
```

网格线

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

# grid() 设置图表中的网格线
'''
plt.grid(b=None, which='major', axis='both', )
b: 可选，默认None，可设置bool值，true显示网格线，false不显示，如果设置**kwargs参数，值
为true
which: 可选，可选值有'major'(默认),'minor','both' 表示应用更改的网格线
axis: 可选，设置显示哪个方向的网格线，可选'both'(默认) 'x','y'，分别表示x轴y轴两个方向
**kwargs: 可选，设置网格样式，可以是color='r',linestyle='-'和linewidth=2，分别表示网
格线的颜色，样式和宽度
'''
x = np.array([1, 2, 3, 4])
y = np.array([1, 4, 9, 16])
# 网格线参数默认值
plt.title('grid test')
plt.xlabel('x-label')
plt.ylabel('y-label')
plt.plot(x, y)
plt.grid()
plt.show()

# 网格线参数设置axis
plt.title('grid test')
plt.xlabel('x-label')
plt.ylabel('y-label')
plt.plot(x, y)
plt.grid(axis='x') # 设置x轴方向显示网格线
plt.show()
```

```
# 设置网格线的样式 样式同绘图线类型标记，颜色标记，线宽度
# grid(color='color', linestyle='linestyle', linewidth='linewidth')
plt.title('grid test')
plt.xlabel('x-label')
plt.ylabel('y-label')
plt.plot(x, y)
plt.grid(color='r', linestyle='--', linewidth=0.5) # 设置x轴方向显示网格线
plt.show()
```

绘制多图

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

# subplot subplots 绘制子图
'''
subplot() 绘图需要指定位置
subplot(nrows, ncols, index, **kwargs)
subplot(pos, **kwargs)
subplot(**kwargs)
subplot(ax)
将绘图区域分为nrows行和ncols列，从左到右从上到下对每个子区域编号1...N，编号可以通过index来设置
suptitle 设置多图的标题
'''
# 一行二列多图
xpoints = np.array([0, 6])
ypoints = np.array([0, 100])
plt.subplot(1, 2, 1)
plt.plot(xpoints, ypoints)
plt.title('plot 1')

x = np.array([1, 2, 3, 4])
y = np.array([1, 4, 9, 16])
plt.subplot(1, 2, 2)
plt.plot(x, y)
plt.title('plot 2')

plt.suptitle('subplot test')
plt.show()

# 二行二列多图
xpoints = np.array([0, 6])
ypoints = np.array([0, 100])
plt.subplot(2, 2, 1)
plt.plot(xpoints, ypoints)
plt.title('plot 1')

x = np.array([1, 2, 3, 4])
y = np.array([1, 4, 9, 16])
plt.subplot(2, 2, 2)
```

```
plt.plot(x, y)
plt.title('plot 2')

x = np.array([1, 2, 3, 4])
y = np.array([5, 16, 17, 8])
plt.subplot(2, 2, 3)
plt.plot(x, y)
plt.title('plot 3')

x = np.array([1, 2, 3, 4])
y = np.array([10, 2, 23, 4])
plt.subplot(2, 2, 4)
plt.plot(x, y)
plt.title('plot 4')

plt.suptitle('subplot test')
plt.show()

'''
subplots() 绘图一次生成多个，在调用时只需要调用生成对象的ax即可
subplots(nrows=1, ncols=1, *, sharex=False, sharey=False, squeeze=True,
        subplt_kw=None, gridspec_kw=None, **fig_kw)
nrows, ncols 设置图表的函数和列数
sharex, sharey 设置xy轴是否共享属性，可设置none, all, row, col  当为false或none时每个子图的x轴和y轴是独立的
squeeze: bool 默认True 表示额外的维度从返回的Axes(轴)对象中挤出，对于N*1或1*N个子图，返回一个1维数组，对于N*M，N>1和M>1返回一个二维数组，
        如果设置False，则不进行挤压操作，返回一个元素为的Axes的2维，即使最终是1*1
subplt_kw: 可选，字典 字典关键字传递给add_subplot()创建子图
gridspec_kw: 可选，字典 字典关键字传递给GridSpec构造函数创建子图放在网格里grid
**fig_kw: 把详细的关键字传给figure函数
suptitle 设置多图的标题
'''
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)

# 创建一个图像和子图
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_title('simple plot')

# 创建2个子图
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
ax1.plot(x, y)
ax1.set_title('sharing y axis')
ax2.scatter(x, y)

# 创建4个子图
fig, axs = plt.subplots(2, 2, subplot_kw=dict(projection='polar')) # 投影类型：极坐标图 通过subplot_kw传递给add_subplot中的projection参数
axs[0, 0].plot(x, y)
axs[1, 1].scatter(x, y)

# 共享x轴
```

```
plt.subplots(2, 2, sharex='col')

# 共享y轴
plt.subplots(2, 2, sharey='row')

# 共享x和y轴 两种方法
plt.subplots(2, 2, sharex='all', sharey='all')
plt.subplots(2, 2, sharex=True, sharey=True)

# 创建标识为10的图，已存在的则删除
fig, ax = plt.subplots(num=10, clear=True)
plt.show()

'''
总结：subplot和subplots的原理
fig = plt.figure()  #首先调用plt.figure()创建了一个**画窗对象fig**
ax = fig.add_subplot(111)  #然后再对fig创建默认的坐标区（一行一列一个坐标区）  笛卡尔坐标系
#这里的（111）相当于（1，1，1），当然，官方有规定，当子区域不超过9个的时候，我们可以这样简写
'''

'''
plt.subplot2grid(shape, location, colspan=1, rowspan=1)
设定网络 选中网格 确定选中行列区域数量 编号从0开始
shape：把该参数值规定的网格区域作为绘图区域；
location：在给定的位置绘制图形，初始位置（0,0）表示第1行第1列；
rowsapan/colspan：这两个参数用来设置让子区跨越几行几列。
'''

#使用 colspan指定列，使用rowspan指定行
a1 = plt.subplot2grid((3,3),(0,0),colspan = 2)
a2 = plt.subplot2grid((3,3),(0,2), rowspan = 3)
a3 = plt.subplot2grid((3,3),(1,0),rowspan = 2, colspan = 2)

x = np.arange(1,10)
a2.plot(x, x*x)
a2.set_title('square')
a1.plot(x, np.exp(x))
a1.set_title('exp')
a3.plot(x, np.log(x))
a3.set_title('log')
plt.tight_layout()
plt.show()
```

散点图

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

```
'''
scatter(x, y, s=None, c=None, marker=None, camap=None, norm=None, vmin=None,
vmax=None, alpha=None,
        linewidths=None, *, edgecolors=None, plotnonfinite=False, data=None,
**kwargs)
x, y:长度相同的数组。数据点
s:点的大小，默认20，也可以是数组，数组每个参数为对应点的大小
c:点的颜色，默认蓝色，也可以是RGB或RGBA二维行数组
marker:点的样式，默认o
cmap:colormap 默认None，标量或是一个colormap的名字，只有c是一个浮点数数组时才使用，如果
没有申明就是image.cmap
norm:归一化 默认None，数据亮度在0-1之间，只有c是一个浮点数数组才使用
vmin,vmax:亮度设置，在norm参数存在时会忽略
alpha:透明度设置，0-1之间，默认None即不透明
linewidth:标记点长度
edgecolors:颜色或颜色序列，可选'face'(默认) 'none' None
plotnonfinite:布尔值，设置是否使用非限定的c(inf,-inf或nan)绘制点
**kwargs:其他参数
'''

x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([1, 4, 9 ,16, 7, 11, 23, 18])

plt.scatter(x, y)
plt.show()

# 设置图标大小
sizes = np.array([20, 50, 100, 200, 500, 1000, 60, 90])
plt.scatter(x, y, s=sizes)
plt.show()

# 自定义点的颜色
x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([1, 4, 9 ,16, 7, 11, 23, 18])
plt.scatter(x, y, color='hotpink')
x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([20, 12, 62, 12, 52, 67, 10, 5])
plt.scatter(x, y, color='#88C999')
plt.show()

# 随机数设置散点图
np.random.seed(19680801) # 随机数生成器种子
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N)) ** 2
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.title('test')
plt.show()

# 颜色条colormap 像一个颜色列表，每种颜色有一个范围从0到100的值
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
```

```
y = np.array([1, 4, 9 ,16, 7, 11, 23, 18, 20, 2, 18])
colors = np.array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
plt.scatter(x, y, c=colors, cmap='viridis') # cmap参数默认viridis
plt.show()
# 要显示颜色条，使用plt.colorbar()方法
plt.scatter(x, y, c=colors, cmap='viridis')
plt.colorbar()
plt.show()
# 换个颜色条参数 设置afmhot_r 带r即为颜色条翻转 参数选择见学习笔记
plt.scatter(x, y, c=colors, cmap='afmhot')
plt.colorbar()
plt.show()
plt.scatter(x, y, c=colors, cmap='afmhot_r')
plt.colorbar()
plt.show()
```

柱形图

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

'''
bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)
x: 浮点型数组，图形x轴数组
height: 浮点型数组，柱形图高度
width: 浮点型数组，柱形图宽度
bottom: 浮点型数组，底座的y坐标，默认0
align: 柱形图与x坐标的对齐方式,'center'以x位置为中心，这是默认值。'edge'将柱形图左边缘与x位置对齐。要对齐右边缘的条形，可以传递负数的宽度值及align='edge'
**kwargs: 其他参数
'''

x = np.array(['test1', 'test2', 'test3', 'test4'])
y = np.array([10, 20, 30, 40])
plt.bar(x, y)
plt.show()

# 垂直方向的柱形图可以用barh()方法进行设置
plt.barh(x, y)
plt.show()

# 设置柱形图颜色
plt.bar(x, y, color='#4CAF50')
plt.show()

# 自定义各个柱形的颜色
plt.bar(x, y, color=['#4CAF50', 'red', 'hotpink', '#556B2F'])
plt.show()

# 设置柱形图宽度 bar用width barh用height
plt.subplot(121)
```

```
plt.bar(x, y, width=0.1)
plt.subplot(122)
plt.barh(x, y, height=0.5)
plt.show()
```

饼图

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

'''
pie(x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.6,
shadow=False, labelsiatance=1.1, startangle=0,
    radius=1, counterclock=True, wedgeprops=None, center=0, 0, frame=False,
totatelabels=False, *, normalize=None, data=None)[source]

x：浮点型数组或列表，用于绘制饼图的数据，表示每个扇形的面积。
explode：数组，表示各个扇形之间的间隔，默认值为0。
labels：列表，各个扇形的标签，默认值为 None。
colors：数组，表示各个扇形的颜色，默认值为 None。
autopct：设置饼图内各个扇形百分比显示格式，%d%% 整数百分比，%0.1f 一位小数， %0.1f%% 一位小数百分比， %0.2f%% 两位小数百分比。
labeldistance：标签标记的绘制位置，相对于半径的比例，默认值为 1.1，如 <1则绘制在饼图内侧。
pctdistance：：类似于 labeldistance，指定 autopct 的位置刻度，默认值为 0.6。
shadow：：布尔值 True 或 False，设置饼图的阴影，默认为 False，不设置阴影。
radius：：设置饼图的半径，默认为 1。
startangle：：用于指定饼图的起始角度，默认为从 x 轴正方向逆时针画起，如设定 =90 则从 y 轴正方向画起。
counterclock：布尔值，用于指定是否逆时针绘制扇形，默认为 True，即逆时针绘制，False 为顺时针。
wedgeprops ：property 字典类型，默认值 None。用于指定扇形的属性，比如边框线颜色、边框线宽度等。例如：wedgeprops={'linewidth':5} 设置 wedge 线宽为5。
textprops ：property 字典类型，用于指定文本标签的属性，比如字体大小、字体颜色等，默认值为 None。
center ：浮点类型的列表，用于指定饼图的中心位置，默认值：(0,0)。
frame ：布尔类型，用于指定是否绘制饼图的边框，默认值：False。如果是 True，绘制带有表的轴框架。
rotatelabels ：布尔类型，用于指定是否旋转文本标签，默认为 False。如果为 True，旋转每个 label 到指定的角度。
data：用于指定数据。如果设置了 data 参数，则可以直接使用数据框中的列作为 x、labels 等参数的值，无需再次传递。

除此之外，pie() 函数还可以返回三个参数：
1. wedges：一个包含扇形对象的列表。
2. texts：一个包含文本标签对象的列表。
3. autotexts：一个包含自动生成的文本标签对象的列表。
'''

y = np.array([35, 25, 25, 15])
plt.pie(y)
```

```
plt.show()

# 设置饼图各个扇区的标签和颜色
plt.pie(y, labels=['A', 'B', 'C', 'D'], colors=['#D5695D', '#5D8CA8', '#65A479',
'#A564C9'])
plt.title('pie test')
plt.show()

# 突出显示第二个扇形，并格式化输出百分比
sizes = [15, 30, 45, 10]
labels = ['A', 'B', 'C', 'D']
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
explode = (0, 0.1, 0, 0) # 突出显示
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
shadow=True, startangle=90)
plt.show()

sizes = [35, 25, 25, 15]
labels = ['A', 'B', 'C', 'D']
colors = ['#D5695D', '#5D8CA8', '#65A479', '#A564C9']
explode = (0, 0.2, 0, 0) # 突出显示
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%.2f%%')
plt.show()
```

直方图

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

'''
hist(x, bins=None, range=None, density=False, weights=None, cumulative=False,
bottom=None, histtype='bar', align='mid',
    orientation='vertical', rwidth=None, log=False, color=None, stacked=False,
**kwargs)

x：表示要绘制直方图的数据，可以是一个一维数组或列表。
bins：可选参数，表示直方图的箱数。默认为10。
range：可选参数，表示直方图的值域范围，可以是一个二元组或列表。默认为None，即使用数据中的
最小值和最大值。
density：可选参数，表示是否将直方图归一化。默认为False，即直方图的高度为每个箱子内的样本
数，而不是频率或概率密度。
weights：可选参数，表示每个数据点的权重。默认为None。
cumulative：可选参数，表示是否绘制累积分布图。默认为False。
bottom：可选参数，表示直方图的起始高度。默认为None。
histtype：可选参数，表示直方图的类型，可以
是'bar'、'barstacked'、'step'、'stepfilled'等。默认为'bar'。
align：可选参数，表示直方图箱子的对齐方式，可以是'left'、'mid'、'right'。默认为'mid'。
orientation：可选参数，表示直方图的方向，可以是'vertical'、'horizontal'。默认
为'vertical'。
rwidth：可选参数，表示每个箱子的宽度。默认为None。
```

```
log：可选参数，表示是否在y轴上使用对数刻度。默认为False。
color：可选参数，表示直方图的颜色。
label：可选参数，表示直方图的标签。
stacked：可选参数，表示是否堆叠不同的直方图。默认为False。
**kwargs：可选参数，表示其他绘图参数。
'''
data = np.random.randn(1000)
plt.hist(data, bins=30, color='skyblue', alpha=0.8)
plt.title('hist test')
plt.xlabel('value')
plt.ylabel('frequency')
plt.show()

data1 = np.random.normal(0, 1, 100000) # 正态分布随机数组 mu sigma size
data2 = np.random.normal(2, 1, 100000)
data3 = np.random.normal(-2, 1, 100000)

plt.hist(data1, bins=30, alpha=0.5, label='Data 1')
plt.hist(data2, bins=30, alpha=0.5, label='Data 2')
plt.hist(data3, bins=30, alpha=0.5, label='Data 3')

plt.title('hist test')
plt.xlabel('value')
plt.ylabel('frequency')
plt.legend() # 显示图例
plt.show()

# * 结合pandas
random_data = np.random.normal(170, 10, 250)
dataframe = pd.DataFrame(random_data) # 数据转换为DataFrame
dataframe.hist() # 使用Pandas.hist() 方法绘制直方图
plt.title('Pandas hist test')
plt.xlabel('X-value')
plt.ylabel('y-value')
plt.show()

# 使用series对象绘制直方图 将数据框的列替换为series对象即可
data = pd.Series(np.random.normal(size=100))
plt.hist(data, bins=10)
plt.title('Pandas hist test')
plt.xlabel('X-value')
plt.ylabel('y-value')
plt.show()
```

极坐标图

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

N = 20
theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
```

```
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)

ax = plt.subplot(111, projection='polar')
bars = ax.bar(theta, radii, width=width, bottom=0.0)

for r, bar in zip(radii, bars):
    bar.set_facecolor(plt.cm.viridis(r / 10))
    bar.set_alpha(0.5)

plt.show()

N = 10
theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
radii = 10 * np.random.rand(N)
width = np.pi / 2 * np.random.rand(N)

ax = plt.subplot(111, projection='polar')
bars = ax.bar(theta, radii, width=width, bottom=0.0)
# 这里left对应从哪个角度开始 height对应扇区高度 width对应转过多少角度

# 设定颜色
for r, bar in zip(radii, bars):
    bar.set_facecolor(plt.cm.viridis(r / 10))
    bar.set_alpha(0.5)

plt.show()
```

imshow()

```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image

'''
imshow() 用于显示图像，常用于绘制二维的灰度图像或彩色图像，还可用于绘制矩阵、热力图、地图等
imshow(x, cmap=None, norm=None, aspect=None, interpolation=None, alpha=None, vimn-None, vmax=None, origin=None, extent=None,
        shape=None, filternorm=1, filterrad=4.0, imlim=None, resample=None, url=None, *, data=None, **kwargs)

X：输入数据。可以是二维数组、三维数组、PIL图像对象、matplotlib路径对象等。
cmap：颜色映射。用于控制图像中不同数值所对应的颜色。可以选择内置的颜色映射，如gray、hot、jet等，也可以自定义颜色映射。
norm：用于控制数值的归一化方式。可以选择Normalize、LogNorm等归一化方法。
aspect：控制图像纵横比（aspect ratio）。可以设置为auto或一个数字。
interpolation：插值方法。用于控制图像的平滑程度和细节程度。可以选择nearest、bilinear、bicubic等插值方法。
alpha：图像透明度。取值范围为0~1。
origin：坐标轴原点的位置。可以设置为upper或lower。
```

```
extent：控制显示的数据范围。可以设置为[xmin, xmax, ymin, ymax]。
vmin、vmax：控制颜色映射的值域范围。
filternorm 和 filterrad：用于图像滤波的对象。可以设置为None、antigrain、freetype等。
imlim：  用于指定图像显示范围。
resample：用于指定图像重采样方式。
url：用于指定图像链接。
'''

# 显示灰度图像
img = np.random.rand(10, 10)
plt.imshow(img, cmap='gray')
plt.show()

# 显示彩色图像
img = np.random.rand(10, 10, 3) # 生成size=(10,10,3)的均匀分布数组[0,1)
plt.imshow(img)
plt.show()

# 显示热力图
data = np.random.rand(10, 10)
plt.imshow(data, cmap='hot')
plt.colorbar() # 加颜色条
plt.show()

# 显示地图
img = Image.open('related_data/map.jpeg')
data = np.array(img)
plt.imshow(data)
plt.axis('off') # 隐藏坐标轴
plt.show()

# 显示矩阵
data = np.random.rand(10, 10)
plt.imshow(data)
plt.show()

# 三种不同imshow图像展示
n = 4
a = np.reshape(np.linspace(0, 1, n ** 2), (n, n)) # 创建n*n的二维数组
plt.figure(figsize=(12, 4.5))

# 展示灰度的色彩映射方式 进行没有进行颜色的混合
plt.subplot(131)
plt.imshow(a, cmap='gray', interpolation='nearest')
plt.xticks(range(n))
plt.yticks(range(n))
plt.title('gray color map, no blending', y=1.02, fontsize=12)

# 展示使用viridis颜色映射的图像 没有颜色的混合
plt.subplot(132)
plt.imshow(a, cmap='viridis', interpolation='nearest')
plt.yticks([]) # x轴刻度位置的列表，若传入空列表，即不显示x轴
plt.xticks(range(n))
plt.title('viridis color map, no blending', y=1.02, fontsize=12)
```

```
# 展示使用viridis颜色映射的图像 使用双立方插值的方法进行颜色混合
plt.subplot(133)
plt.imshow(a, cmap='viridis', interpolation='bicubic')
plt.yticks([])
plt.xticks(range(n))
plt.title('visidis color map, bicubic blending', y=1.02, fontsize=12)

plt.show()
```

imsave()

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

'''
imsave(fname, arr, **kwargs) 将图像数据保存在磁盘上 保存在指定目录 支持PNG JPEG BMP等多种图像格式
kwargs: 可选。用于指定保存的图像格式以及图像质量等参数
'''

img_data = np.random.random((100, 100)) # 功能同rand 但random传入一个完整的元组 rand接收分开的参数
plt.imshow(img_data)
plt.imsave('related_data/test.png', img_data)

# 创建灰度图像
img_gray = np.random.random((100, 100))
# 创建彩色图像
img_color = np.zeros((100, 100, 3))
img_color[:, :, 0] = np.random.random((100, 100))
img_color[:, :, 1] = np.random.random((100, 100))
img_color[:, :, 2] = np.random.random((100, 100))

plt.imshow(img_gray, cmap='gray')
plt.imsave('related_data/test_gray.png', img_gray, cmap='gray')
plt.imshow(img_color)
plt.imsave('related_data/test_color.jpg', img_color) # 若未指定图像格式 默认保存JPEG格式
```

imread()

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

'''
imread(fname, format=None) 从图像文件中读取图像数据，返回一个np.ndarray对象。形状(nrows, ncols, nchannels) 灰度图像通道数1 彩色图像通道数3或4 红绿蓝还有alpha通道
format: 指定图像文件格式，若不指定，默认根据文件后缀自动识别格式
```

```
'''

img = plt.imread('related_data/map.jpeg')
plt.imshow(img)
plt.show()

# 更改numpy数组修改图像 这里使图像变暗
img_map = img / 255
plt.figure(figsize=(10, 6))

for i in range(1, 5):
    plt.subplot(2, 2, i)
    x = 1 - 0.2 *(i - 1)
    plt.axis('off')
    plt.title('x={:.1f}'.format(x))
    plt.imshow(img_map * x)
plt.show()
```

中文显示

```
import matplotlib.font_manager
import matplotlib.pyplot as plt
import matplotlib
import numpy as np

# 获取系统字体库列表
# from matplotlib import pyplot as plt
# import matplotlib
# a=sorted([f.name for f in matplotlib.font_manager.fontManager.ttflist])

# for i in a:
#     print(i)


# 两种方法
# 1.替换Matplotlib默认字体
plt.rcParams['font.family'] = 'Source Han Sans CN'
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y)
plt.title('折线图实例(替换默认字体)')
plt.xlabel('x轴')
plt.ylabel('y轴')
plt.show()

# 2. 使用OTF字体库
font = matplotlib.font_manager.FontProperties(fname='related_data/SourceHanSansCN-
Bold.otf')
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y)
```

```
plt.title('折线图实例(设置字体属性)', fontproperties=font)
plt.xlabel('x轴', fontproperties=font)
plt.ylabel('y轴', fontproperties=font)
plt.show()
```

Seaborn入门

```
'''
Seaborn 是一个建立在matplotlib基础之上的python数据可视化库
简化统计数据可视化过程，提供高级接口和美观的默认主题
提供高级接口，轻松绘制散点图、折线图、柱状图、热图等
注重美观性，绘图更吸引人
'''
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
'''
sns.set_theme() 选择不同的主题和模板

主题theme:
darkgrid(默认)：深色网格主题
whitegold：浅色网格主题
dark：深色主题 没有网络
white：浅色主题 没有网络
ticks：深色主题 带有刻度标记

模板Context
paper：适用于小图，具有较小的标签和线条
notebook(默认)：适用于笔记本电脑和类型环境，具有中等大小的标签和线条
talk：适用于演讲幻灯片，大尺寸标签和线条
poster：  适用于海报，非常大的标签和线条
'''
sns.set_theme(style='whitegrid', palette='pastel')
products = ['product A', 'product B', 'product C', 'product D']
sales = [120, 210, 150, 180]
sns.barplot(x=products, y=sales) # 创建柱状图
plt.xlabel('products')
plt.ylabel('sales')
plt.title('product sales by category')
plt.show()

# * 绘图函数
# 散点图 sns.scatterplot() 用于绘制两个变量之间的散点图，可选择增加趋势线
data = {'A': [1, 2, 3, 4, 5], 'B':[5, 4, 3, 2, 1]}
df = pd.DataFrame(data)
sns.scatterplot(x='A', y='B', data=df)
plt.show()

# 折线图 sns.lineplot() 绘制变量随着另一个变量变化的趋势线图
data = {'X': [1, 2, 3, 4, 5], 'Y':[5, 4, 3, 2, 1]}
df = pd.DataFrame(data)
```

```
sns.lineplot(x='X', y='Y', data=df)
plt.show()

# 柱状图 sns.barplot() 绘制变量的均值或其他聚合函数的柱状图
data = {'Category':['A', 'B', 'C'], 'Value':[3, 7, 5]}
df = pd.DataFrame(data)
sns.barplot(x='Category', y='Value', data=df)
plt.show()

# 箱线图 sns.boxplot() 绘制变量的分布情况，包括中位数、四分位数等
data = {'Category':['A', 'A', 'B', 'B', 'C', 'C'], 'Value':[3, 7, 5, 9, 2, 6]}
df = pd.DataFrame(data)
sns.boxplot(x='Category', y='Value', data=df)
plt.show()

# 热图 sns.heatmap() 绘制矩阵数据的热图，展示相关性矩阵
data = {'A': [1, 2, 3, 4, 5], 'B':[5, 4, 3, 2, 1]}
df = pd.DataFrame(data)
correlation_matrix = df.corr() # 创建一个相关性矩阵
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.show()

# 小提琴图 sns.violinplot() 显示分布的形状和密度估计，结合了箱线图和核密度估计
data = {'Category':['A', 'A', 'B', 'B', 'C', 'C'], 'Value':[3, 7, 5, 9, 2, 6]}
df = pd.DataFrame(data)
sns.violinplot(x='Category', y='Value', data=df)
plt.show()
```

实例_引力波的绘制

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import wavfile # 读取波形文件的库

rate_h, hstrain = wavfile.read(r'related_data/H1_Strain.wav', 'rb') # 读取声音文件
rate_l, lstrain = wavfile.read(r'related_data/L1_Strain.wav', 'rb')
# 读取时间序列和信号数据 genfromtxt执行两个循环 第一个循环将文件每一行转化为字符串 第二个循环将每个字符串转换成相应的类型 因为读取出的是一个两行的矩阵 不方便使用 因此使用tranpose转置
reftime, ref_H1 = np.genfromtxt('related_data/wf_template.txt').transpose()

htime_interval = 1 / rate_h
ltime_interval = 1 / rate_l

htime_len = hstrain.shape[0] / rate_h
htime = np.arange(-htime_len / 2, htime_len / 2, htime_interval)
ltime_len = lstrain.shape[0] / rate_l
ltime = np.arange(-ltime_len / 2, ltime_len / 2, ltime_interval)

fig = plt.figure(figsize=(12, 6))
```

```
plth = fig.add_subplot(221)
plth.plot(htime, hstrain, 'y')
plth.set_xlabel('Time(seconds)')
plth.set_ylabel('H1 Strain')
plth.set_title('H1 Strain')

pltl = fig.add_subplot(222)
pltl.plot(ltime, lstrain, 'g')
pltl.set_xlabel('Time(seconds)')
pltl.set_ylabel('L1 Strain')
pltl.set_title('L1 Strain')

plth = fig.add_subplot(212)
plth.plot(reftime, ref_H1)
plth.set_xlabel('Time(seconds)')
plth.set_ylabel('Template Strain')
plth.set_title('Template Strain')
fig.tight_layout() # 自动调整外部边缘

plt.savefig('related_data/Gravitational_Waves_Original.png')
plt.show()
plt.close(fig)
```

Pandas

Pandas

1. 是 Python 语言的一个扩展程序库，用于数据分析。
2. 名字衍生自术语 "panel data"（面板数据）和 "Python data analysis"（Python 数据分析）。
3. 是一个开放源码、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。
4. 是一个强大的分析结构化数据的工具集，基础是 Numpy（提供高性能的矩阵运算）。

应用场景

1. 可以从各种文件格式比如 CSV、JSON、SQL、Microsoft Excel 导入数据。
2. 可以对各种数据进行运算操作，比如归并、再成形、选择，还有数据清洗和数据加工特征。
3. 广泛应用在学术、金融、统计学等各个数据分析领域。

数据结构 Pandas 的主要数据结构是 Series （一维数据）与 DataFrame（二维数据）。

1. Series 是一种类似于一维数组的对象，它由一组数据（各种 Numpy 数据类型）以及一组与之相关的数据标签（即索引）组成。
2. DataFrame 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔型值）。DataFrame 既有行索引也有列索引，它可以被看做由 Series 组成的字典（共同用一个索引）。

特点 Pandas 可以轻松地处理各种数据结构，尤其是表格型数据，如 SQL 数据库或 Excel 表格。

1. 数据结构：Pandas 提供了两种主要的数据结构：Series 和 DataFrame。Series 是一维标记数组，类似于 Python 中的列表或 NumPy 中的数组，而 DataFrame 是一个二维的表格型数据结构，类似于 SQL 表或 Excel 表格。

2. 数据加载与保存：Pandas 可以从各种数据源加载数据，包括 CSV 文件、Excel 表格、SQL 数据库、JSON 文件等，并且可以将处理后的数据保存到这些格式中。
3. 数据清洗与转换：Pandas 提供了丰富的函数和方法，用于数据清洗、处理缺失值、重复值、异常值等，以及进行数据转换、重塑和合并操作。
4. 数据分析与统计：Pandas 提供了各种统计函数和方法，用于描述性统计、聚合操作、分组运算、透视表等数据分析任务。
5. 数据可视化：Pandas 结合了 Matplotlib 库，可以轻松地进行数据可视化，绘制各种统计图表，如折线图、散点图、直方图等。

Pandas 是 Python 数据科学领域中不可或缺的工具之一，它的灵活性和强大的功能使得数据处理和分析变得更加简单和高效。

Series

Series类似表格的一个列。类似一维数组，可以保存任何数据类型 Series可以处理各种数据类型，同时保持高效的数据操作能力 特点：

1. 一维数组：一维的，只有一个轴，类似列表
2. 索引：每个Series都有一个索引，可以是整数、字符串、日期等类型。如果不指定索引，Pandas 将默认创建一个从 0 开始的整数索引。
3. 数据类型： Series 可以容纳不同数据类型的元素，包括整数、浮点数、字符串、Python 对象等。
4. 大小不变性：Series 的大小在创建后是不变的，但可以通过某些操作（如 append 或 delete）来改变。
5. 操作：Series 支持各种操作，如数学运算、统计分析、字符串处理等。
6. 缺失数据：Series 可以包含缺失数据，Pandas 使用NaN（Not a Number）来表示缺失或无值。

```
import pandas as pd
import numpy as np

# 创建Series
'''
pandas.Series(data=None, index=None, dtype=None, name=None, copy=False,
fastpath=False)

data：Series 的数据部分，可以是列表、数组、字典、标量值等。如果不提供此参数，则创建一个空的 Series。
index：Series 的索引部分，用于对数据进行标记。可以是列表、数组、索引对象等。如果不提供此参数，则创建一个默认的整数索引。
dtype：指定 Series 的数据类型。可以是 NumPy 的数据类型，例如 np.int64、np.float64 等。如果不提供此参数，则根据数据自动推断数据类型。
name：Series 的名称，用于标识 Series 对象。如果提供了此参数，则创建的 Series 对象将具有指定的名称。
copy：是否复制数据。默认为 False，表示不复制数据。如果设置为 True，则复制输入的数据。
fastpath：是否启用快速路径。默认为 False。启用快速路径可能会在某些情况下提高性能。
'''
a = [1, 2, 3]
myvar = pd.Series(a)
print(myvar) # 索引在左 数据在右 数据类型在下
print(myvar[1]) # 指定索引

# 指定索引值
```

```
a = ['Iven', 'Rosenn', 'Starry']
myvar = pd.Series(a, index=['x', 'y', 'z'])
print(myvar['y'])

# 也可以用key/value对象，类似字典来创建Series
sites = {1:'Iven', 2:'Rosenn', 3:'Starry'}
myvar = pd.Series(sites)
print(myvar)
# 可以看出key变成了索引值，可以通过key取出数据
print(myvar[1])

# 如果只需要字典中的一部分数据，只需要指定需要数据的索引即可
myvar = pd.Series(sites, index=[1, 2])
print(myvar)

# 设置Series名称参数
myvar = pd.Series(sites, index=[1, 2], name='Iven_Series_test')
print(myvar)

# 使用列表、字典或数组创建一个默认索引的Series
s = pd.Series([1, 2, 3, 4])
s = pd.Series(np.array([1, 2, 3, 4]))
s = pd.Series({'a':1, 'b':2, 'c':3, 'd':4})

# todo 基本操作
# 指定索引创建Series
s = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])

# 获取值
value = s[2] # 不良好的书写习惯 尽量使用数据标签获取值
print(value)
print(s['a'])

# 保留字 in
b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])
print('c' in b)
print(0 in b)

# python .get()方法
'''
python字典的get方法会返回指定键的值，dict.get('键')，返回"键"对应的"值"，如果键不在字典中则返回默认值None。

dict1={'国家':'中国','首都':'北京'}
print(dict1.get('国家'))
print(dict1.get('首都'))
print(dict1.get('省会'))
输出为：中国 北京 None

如果键不在字典中，想要自己设置返回值，可以这样处理，
例如dict.get('键','never')
键在字典中，则返回键对应的值，键不在字典中，则返回never。
'''
print(b.get('f', 100))
```

```
# 获取多个值
subset = s[1:4]
print(subset)

# 使用自定义索引
value = s['b']
print(value)

# 索引与值对应的关系
for index, value in s.items():
    print(f'Index: {index}, value:{value}')

# 使用切片语法来访问Series的一部分
print(s['a':'c'])
print(s[:3])

# 为特定的索引标签赋值
s['a'] = 10

# 通过赋值给新的索引标签来添加元素
s['e'] = 5

# 使用 del 删除指定索引标签的元素
del s['a']

# 使用 drop 方法删除一个或多个索引标签，返回一个新的Series
s_dropped = s.drop(['b'])
print(s_dropped)

# todo 基本运算
# 算术运算
result = s_dropped * 2 # 所有元素乘2
print(result)

# 过滤
filtered_series = s_dropped[s_dropped > 3]
print(filtered_series)

# 数学函数
result = np.sqrt(s_dropped)
print(result)

# todo 对齐操作 series + series
a = pd.Series([1, 2, 3], ['c', 'd', 'e'])
b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])
print(a + b)

# todo 计算统计数据 使用Series的方法来计算描述性统计
print(s_dropped.sum()) # 计算Series的总和
print(s_dropped.mean()) # 平均值
print(s_dropped.max()) # 最大值
print(s_dropped.min()) # 最小值
print(s_dropped.std()) # 标准差
```

```
# todo 属性和方法
# 获取索引
index = s.index
print(index)

# 获取值数组
values = s.values
print(values)

# 获取描述统计信息
stats = s.describe()
print(stats)

# 获取最大值最小值索引
max_index = s.idxmax()
print(max_index)
min_index = s.idxmin()
print(min_index)

# name 属性 series对象和索引都可以有一个名字，存放在属性name中
b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])
b.name = 'Series对象'
b.index.name = '索引列'
print(b)

# 其他属性和方法
print(s.dtype) # 数据类型
print(s.shape) # 形状
print(s.size)  # 元素个数
print(s.head())# 前几个元素，默认前五个
print(s.tail())# 后几个元素，默认后五个
print(s.sum()) # 求和
print(s.mean())# 平均值
print(s.max()) # 最大值
print(s.min()) # 最小值
print(s.std()) # 标准差

# 布尔表达式 根据条件过滤Series
print(s > 3)

# 转换数据类型 astype 将Series转换为另一种数据类型
s_astype = s.astype('float64')
print(s.dtype)
print(s_astype.dtype)
```

DataFrame

DataFrame

1. 是 Pandas 中的另一个核心数据结构，用于表示二维表格型数据。

2. 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔型值）。
3. 既有行索引也有列索引，它可以被看做由 Series 组成的字典（共同用一个索引）。
4. 提供了各种功能来进行数据访问、筛选、分割、合并、重塑、聚合以及转换等操作。

特点

1. 二维结构： DataFrame 是一个二维表格，可以被看作是一个 Excel 电子表格或 SQL 表，具有行和列。可以将其视为多个 Series 对象组成的字典。
2. 列的数据类型： 不同的列可以包含不同的数据类型，例如整数、浮点数、字符串或 Python 对象等。
3. 索引：DataFrame 可以拥有行索引和列索引，类似于 Excel 中的行号和列标。
4. 大小可变：可以添加和删除列，类似于 Python 中的字典。
5. 自动对齐：在进行算术运算或数据对齐操作时，DataFrame 会自动对齐索引。
6. 处理缺失数据：DataFrame 可以包含缺失数据，Pandas 使用 NaN（Not a Number）来表示。
7. 数据操作：支持数据切片、索引、子集分割等操作。
8. 时间序列支持：DataFrame 对时间序列数据有特别的支持，可以轻松地进行时间数据的切片、索引和操作。
9. 丰富的数据访问功能：通过 .loc、.iloc 和 .query() 方法，可以灵活地访问和筛选数据。
10. 灵活的数据处理功能：包括数据合并、重塑、透视、分组和聚合等。
11. 数据可视化：虽然 DataFrame 本身不是可视化工具，但它可以与 Matplotlib 或 Seaborn 等可视化库结合使用，进行数据可视化。
12. 高效的数据输入输出：可以方便地读取和写入数据，支持多种格式，如 CSV、Excel、SQL 数据库和 HDF5 格式。
13. 描述性统计：提供了一系列方法来计算描述性统计数据，如 .describe()、.mean()、.sum() 等。
14. 灵活的数据对齐和集成：可以轻松地与其他 DataFrame 或 Series 对象进行合并、连接或更新操作。
15. 转换功能：可以对数据集中的值进行转换，例如使用 .apply() 方法应用自定义函数。
16. 滚动窗口和时间序列分析：支持对数据集进行滚动窗口统计和时间序列分析。

```
import pandas as pd
import numpy as np

# todo 创建DataFrame
'''
pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)

data：DataFrame 的数据部分，可以是字典、二维数组、Series、DataFrame 或其他可转换为 DataFrame 的对象。如果不提供此参数，则创建一个空的 DataFrame。
index：DataFrame 的行索引，用于标识每行数据。可以是列表、数组、索引对象等。如果不提供此参数，则创建一个默认的整数索引。
columns：DataFrame 的列索引，用于标识每列数据。可以是列表、数组、索引对象等。如果不提供此参数，则创建一个默认的整数索引。
dtype：指定 DataFrame 的数据类型。可以是 NumPy 的数据类型，例如 np.int64、np.float64 等。如果不提供此参数，则根据数据自动推断数据类型。
copy：是否复制数据。默认为 False，表示不复制数据。如果设置为 True，则复制输入的数据。
'''

# 使用列表创建
data = [['Iven', 21], ['Rosenn', 25], ['Starry', 19]]
df = pd.DataFrame(data, columns=['Name', 'Age'])
```

```
df['Name'] = df['Name'].astype(str) # astype方法设置每列的数据类型
df['Age'] = df['Age'].astype(float)
print(df)

# 使用字典创建
data = {'Name':['Iven', 'Rosenn', 'Starry'], 'Age':[21, 25, 19]}
df = pd.DataFrame(data)
print(df)

# 使用ndarray对象创建 长度必须相同，若传递了index，索引长度要等于数组长度，若未传递索引，默认是range(n),n是数组长度
ndarray_data = np.array([['Iven', 21], ['Rosenn', 25], ['Starry', 19]])
df = pd.DataFrame(ndarray_data, columns=['Name', 'Age'])
print(df)

# 使用字典创建 key为列名
data = [{'a':1, 'b':2}, {'a':5, 'b':10, 'c':20}]
df = pd.DataFrame(data)
print(df) # 没有相对应的数据为NAN

# 使用Series创建
s1 = pd.Series(['Iven', 'Rosenn', 'Starry'])
s2 = pd.Series([21, 23, 24])
s3 = pd.Series(['Chengdu', 'Hangzhou', 'Xi\'an'])
df = pd.DataFrame({'Name':s1, 'Age':s2, 'City':s3})
print(df)

# todo 访问DataFrame元素
# 访问行 loc 返回指定行的数据，若没有设置索引，第一行索引为0，第二行为1
data = {'calories': [420, 280, 400], 'duration':[50, 45, 40]}
df = pd.DataFrame(data)
print(df.loc[0])
print(df.loc[1]) # 返回结果是一个Series数据
print(df.loc[[0,1]]) # 返回一个dataframe数据

# 访问列 loc[] iloc[]
data = {'calories': [420, 280, 400], 'duration':[50, 45, 40]}
df = pd.DataFrame(data, index=['day1', 'day2', 'day3']) # 指定索引值
print(df)
print(df.loc['day2']) # df.loc[row_label, column_label] 索引名 day1 day2
print(df.iloc[:, 0])  # df.iloc[row_index, column_index] 整数 0, 1, ...

# 访问单个元素 [列][行] 先得到一列Series 在得到Series中的一个数据
print(df['calories'][0])

# todo DataFrame 属性和方法
print(df.shape)      # 形状
print(df.columns)    # 列名
print(df.index)      # 索引
print(df.head())     # 前几行数据，默认前五行
print(df.tail())     # 后几行数据，默认后五行
print(df.info())     # 数据信息
print(df.describe()) # 描述统计信息
print(df.mean())     # 求平均值
```

```
print(df.sum())      # 求和
print(df.max())      # 最大值
print(df.min())      # 最小值

# todo 修改DataFrame
# 修改列数据
df['calories'] = [200, 300, 400]
# 修改行数据
df.loc['day1'] = [100, 90]
# 添加新列
df['food'] = ['rice', 'banana', 'apple']
# 添加新行 loc 指定特定索引添加新行 concat 合并两个或多个DataFrame  append(已被弃用)
df.loc['day4'] = [500, 60, 'bread']
new_row = pd.DataFrame([[600, 70, 'cococola']], index=['day5'], columns=
['calories', 'duration', 'food'])
df = pd.concat([df, new_row], ignore_index=False)
print(df)
# // new_row = {'calories':600, 'duration':70, 'food':'cococola'}
# // df = df.append(new_row, ignore_index=True)
# 删除列 drop
df = df.drop('duration', axis=1)  # 删除轴1的duration，即列名为duration的列
print(df)
# 删除行 drop
df_drop = df.drop('day1') # 默认删除轴0的day1，即行名为day1的行
print(df_drop)

# todo 索引操作
# 重置索引 reset_index
df_reset = df.reset_index(drop=True)
print(df_reset)
# 设置索引 set_index 将一列设置为索引 drop是否保留被转换的列
df_set = df.set_index(['calories'], drop=True)
print(df_set)
# 布尔索引
print(df[df['calories'] > 400])
'''
append(idx)          连接另一个index对象，产生新的index对象
diff(idx)            计算差集，产生新的index对象
intersection(idx)    计算交集
union(idx)           计算并集
delete(loc)          删除loc位置的元素
insert(loc,e)        在loc位置添加一个元素e
'''
df_del = df.columns.delete(1)
print(df_del)
df_ins = df.index.insert(1, 'day6')
print(df_ins)
print(df)
df_rei = df.reindex(index=df_ins, columns=df_del)
print(df_rei)
df_rei = df.reindex(index=df_ins, columns=df_del, method='bfill')
print(df_rei)
df_rei = df.reindex(index=df_ins, columns=df_del, method='ffill')
print(df_rei)
```

```
# 重排索引
'''
reindex(index=None, columns=None, ...) 重拍已有的序列

index,columns 新的行列自定义索引
fill_value    重新索引中，用于填充缺失位置的值
method        填充方法，ffill 会将上一个非 NaN 的值填充到此位置  bfill会将下一个非 NaN 的值填充到此位置
                跟行列插入的位置无关(1,)(不是找索引0 索引1的数值)  跟索引值名称有关(找day6前一个索引day5数值和后一个索引NAN)
limit         最大填充量
copy          True生成新对象
'''
df_reindex = df.reindex(index = ['day1', 'day3', 'day2', 'day5', 'day4'], columns=['food', 'calories'])
print(df)
print(df_reindex)
new_df_insert = df_reindex.columns.insert(2, '新增') # insert 在列的指定位置加新列后的所有列名
print(new_df_insert)
new_df = df_reindex.reindex(columns=new_df_insert, fill_value=200)
print(new_df)

# todo 数据类型
# 查看数据类型 dtypes
print(df.dtypes)
# 转换数据类型 astype
df['calories'] = df['calories'].astype('float32')
print(df)

# todo 合并与分割
# concat
'''
pd.concat([df1, df2], ignore_index=True, join='outer',axis=1)

axis：0轴（默认）按行拼接（增加行），1轴，按列拼接（增加列）；
join：outer默认（拼接时取并集）；inner（拼接时取交集）
ignore_index：默认False，即不重置dataframe的索引；True重置索引，从0开始

'''
new_row = pd.DataFrame([[700, 'fenta']], index=['day6'], columns=['calories', 'food'])
df_row_concat = pd.concat([df, new_row])
print(df_row_concat)
new_col = pd.DataFrame({'name':['Iven', 'Rosenn', 'Starry', 'Bob', 'Alen']}, index=['day1', 'day2', 'day3', 'day4', 'day5'])
df_col_concat = pd.concat([df, new_col], axis=1)
print(df_col_concat)

# merge
'''
pd.merge(df1,df2,on='列名',how='outer')
how：合并方式：how = 'inner'（默认）类似于取交集  'outer'，类似于取并集 left以左表为主
```

```
表 right以右表为主表
on:  用于连接的列名，若不指定则以两个Dataframe的列名的交集作为连接键
'''
new_col = pd.DataFrame({'food':['rice', 'banana', 'apple', 'bread', 'cococola'],
'name':['Iven', 'Rosenn', 'Starry', 'Bob', 'Alen']})
df_col_merge = pd.merge(df, new_col, how='outer')
print(df_col_merge)
new_row = pd.DataFrame({'calories':900, 'food':'orange'},index=['day6'])
df_row_merge = pd.merge(df, new_row, how='outer')
print(df_row_merge)


# todo 算术运算
# 根据行列索引，补齐后运算，默认产生浮点数，补齐时填充NAN。二维和一维、一维和零维间为广播运算，+ - * / 运算产生新对象
a = pd.DataFrame(np.arange(12).reshape(3, 4))
b = pd.DataFrame(np.arange(20).reshape(4, 5))
print(a)
print(b)
print(a + b)
print(a * b)
'''
add(d, **argws)  类型间加法运算
sub(d, **argws)  类型间减法运算
mul(d, **argws)  类型间乘法运算
div(d, **argws)  类型间除法运算
'''
print(b.add(a, fill_value=8888)) # 哪空替代哪 在运算
print(a.mul(b, fill_value=1))

# 不同维度间为广播运算，一维Series默认在轴1运算
a = pd.Series(np.arange(4))
b = pd.DataFrame(np.arange(20).reshape(4, 5))
print(a - 10)
print(b - a) # b每行减去a的转置
print(b.sub(a, axis=0)) # 使用运算方法可以让一维Series在轴0运算

# todo 比较运算法则
# 只能比较相同索引的元素，不可以补齐，二维和一维、一维和零维间为广播运算，> < >= <= == !=运算产生布尔对象
a = pd.DataFrame(np.arange(12).reshape(3, 4))
b = pd.DataFrame(np.arange(12, 0, -1).reshape(3, 4))
print(a)
print(b)
print(a > b)
print(a == b)

# 不同维度广播
a = pd.Series(np.arange(4))
b = pd.DataFrame(np.arange(12).reshape(3, 4))
print(a)
print(b)
print(a > b)
print(b > 0)
```

CSV

```
import pandas as pd

'''
CSV comma separated values 逗号分隔值 以纯文本形式存储表格数据
是一种通用的、箱底简单的文件格式
pandas可以处理CSV文件
'''

# 读取CSV文件 read_csv
df = pd.read_csv('related_data/nba.csv')
print(df.to_string()) # to_string可以返回dataframe类型的数据，若不使用，数据只显示前五行和后五行，中间用...代替
print(df)

# 存储CSV文件 to_csv
name = ['Iven', 'Rosenn', 'Starry']
age = [19, 20, 21]
hobby = ['food', 'coding', 'travel']
dict = {'name':name, 'age':age, 'hobby':hobby}
df = pd.DataFrame(dict)
df.to_csv('related_data/savetest.csv')

# 数据处理
df = pd.read_csv('related_data/nba.csv')
# head(n) 读取前面n行，默认5行
print(df.head(3))
# tail(n) 读取尾部n行，默认5行
print(df.tail(3))
# info() 返回表格的一些基本信息
print(df.info())
```

JSON

```
import pandas as pd
import json
from glom import glom

'''
JSON Java Script Objection Notation  Java Script对象表示法，是存储和交换文本信息的语法，类似XML
JSON比XML更小、更快、更易解析
pandas可以处理JSON数据
'''

# 读取JSON文件 read_json
df = pd.read_json('related_data/test.json')
```

```
print(df.to_string())

# 也可以直接处理JSON字符串
data = [
    {
        "id": "1",
        "name": "Iven",
        "age": 19
    },
    {
        "id": "2",
        "name": "Rosenn",
        "age": 21
    },
    {
        "id": "3",
        "name": "Bob",
        "age": 22
    }
]
df = pd.DataFrame(data)
print(df)

# 字典转DataFrame JSON对象与字典具有相同的格式
s = {
    'col1':{'row1':1, 'row2':2, 'row3':3},
    'col2':{'row4':4, 'row5':5, 'row6':6},
}
df = pd.DataFrame(s) # 读取JSON文件转dataframe
print(df)

# 内嵌的JSON数据文件
df = pd.read_json('related_data/nested_list.json')
print(df)
# 这时候需要json_normalize()将内嵌的数据完整解析 record_path设置要展开的内嵌数据名
meta展示无需展开的其他数据名
with open('related_data/nested_list.json', 'r') as f:
    data = json.loads(f.read()) # 使用python的json模块读取数据 字典结构
df_nested_list = pd.json_normalize(data, record_path=['students'], meta=
['school_name', 'class'])
print(df_nested_list)

# 更复杂的json文件
with open ('related_data/nested_mix.json', 'r') as f:
    data = json.loads(f.read())
df = pd.json_normalize(data, record_path=['students'], meta=['class', ['info',
'president'], ['info', 'contacts', 'tel']])
print(df)

# 读取内嵌数据中的一组数据 glom模块允许我们使用. 来访问内嵌对象的属性
df = pd.read_json('related_data/nested_deep.json')
data = df['students'].apply(lambda row: glom(row, 'grade.math'))
print(data)
```

数据清洗

```
import pandas as pd

'''
数据清洗是对一些没有用的数据进行处理的过程
很多数据集存在数据缺失、数据格式错误或重复数据的情况，为了使数据分析更加准确，就需要对这些数据进行处理
有四种空数据：n/a NA -- na
'''

# 清洗空值
'''
dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)

axis：默认为 0，表示逢空值剔除整行，如果设置参数 axis＝1 表示逢空值去掉整列。
how：默认为 'any' 如果一行（或一列）里任何一个数据有出现 NA 就去掉整行，如果设置 how='all' 一行（或列）都是 NA 才去掉这整行。
thresh：设置需要多少非空值的数据才可以保留下来的。
subset：设置想要检查的列。如果是多个列，可以使用列名的 list 作为参数。
inplace：如果设置 True，将计算得到的值直接覆盖之前的值并返回 None，修改的是源数据。

isnull() 判断各个单元格是否为空
'''

df = pd.read_csv('related_data/property-data.csv')
print(df.to_string())
print(df['NUM_BEDROOMS'])
print(df['NUM_BEDROOMS'].isnull()) # na是空数据但没被判为空数据

# 指定空数据类型 na_values
missing_values = ['n/a', 'na', '--']
df = pd.read_csv('related_data/property-data.csv', na_values=missing_values)
print(df['NUM_BEDROOMS'])
print(df['NUM_BEDROOMS'].isnull())

# 删除空数据的行 默认返回新的DataFrame
new_df = df.dropna()
print(new_df.to_string())
df.dropna(inplace=True) # 直接修改原数据
print(df.to_string())

# 指定列有空值的行
df = pd.read_csv('related_data/property-data.csv')
df.dropna(subset=['ST_NUM'], inplace=True)
print(df.to_string())

# fillna() 替换一些空字段
df = pd.read_csv('related_data/property-data.csv')
df.fillna(12345, inplace=True)
print(df.to_string())
```

```
# 指定某一列来替换数据
df = pd.read_csv('related_data/property-data.csv')
df['PID'].fillna(12345, inplace=True)
print(df.to_string())

# 替换空单元格的常用方法是计算列的均值、中位数或众数
df = pd.read_csv('related_data/property-data.csv')
x = df['ST_NUM'].mean() # mean() 计算均值
df_mean = df['ST_NUM'].fillna(x)
print(df_mean.to_string())

x = df['ST_NUM'].median() # median() 计算中位数
df_median = df['ST_NUM'].fillna(x)
print(df_median.to_string())

x = df['ST_NUM'].mode() # mode() 计算众数
df_mode = df['ST_NUM'].fillna(x)
print(df_mode.to_string())

# 清洗格式错误数据
# 通过包含空单元格的行，或者将列中的所有单元格转换为相同格式的数据
data = {
    'Date':['2020/12/01', '2020/12/02', '20201226'],
    'duration':[50, 40, 45]
}
df = pd.DataFrame(data, index=['day1', 'day2', 'day3'])
df['Date'] = pd.to_datetime(df['Date'], format='mixed') # to_datetime格式化日期
print(df.to_string())

data = {
    'Date':['2020/12/01', '2020/12/02', '20201226'],
    'duration':[50, 40, 12345] # 12345 数据是错误的
}
df = pd.DataFrame(data)
df.loc[2, 'duration'] = 30 # 修改错误日期  loc(row_index, column_index)
print(df.to_string())

data = {
    'Date':['2020/12/01', '2020/12/02', '20201226'],
    'duration':[50, 40, 12345] # 12345 数据是错误的
}
df = pd.DataFrame(data)
for x in df.index:
    if df.loc[x, 'duration'] > 120: # 设置条件语句将大于120的值设为120
        df.loc[x, 'duration'] = 120
print(df.to_string())

data = {
    'Date':['2020/12/01', '2020/12/02', '20201226'],
    'duration':[50, 40, 12345] # 12345 数据是错误的
}
df = pd.DataFrame(data)
for x in df.index:
```

```
    if df.loc[x, 'duration'] > 120: # 将错误数据的行删除
        df.drop(x, inplace=True)
print(df.to_string())

# 清洗重复数据
data = {
    'name':['Iven', 'Iven', 'Rosenn', 'Starry'],
    'age':[21, 21, 22, 23] # 数据重读
}
df = pd.DataFrame(data)
# duplicated() 如果重复返回True 否则False
print(df.duplicated())
# drop_duplicates()删除重复数据
df.drop_duplicates(inplace=True)
print(df)
```

常用函数

```
import pandas as pd
import numpy as np
'''
读取数据

pd.read_csv(filename)                    读取 CSV 文件
pd.read_excel(filename)                  读取 Excel 文件
pd.read_sql(query, connection_object)    从 SQL 数据库读取数据
pd.read_json(json_string)                从 JSON 字符串中读取数据
pd.read_html(url)                        从 HTML 页面中读取数据
'''
# # 读取 CSV 文件
# df = pd.read_csv('data.csv')
# # 读取 Excel 文件
# df = pd.read_excel('data.xlsx')
# # 从 SQL 数据库读取数据
# import sqlite3
# conn = sqlite3.connect('database.db')
# df = pd.read_sql('SELECT * FROM table_name', conn)
# # 从 JSON 字符串中读取数据
# json_string = '{"name":"john", "age":30, "city":"chengdu"}'
# df = pd.read_json(json_string)
# # 从 HTML 页面中读取数据
# url = 'https://github.com/IvenStarry'
# dfs = pd.read_html(url)
# df = dfs[0]
# print(df)

# '''
# 查看数据

# df.head(n)        显示前 n 行数据
# df.tail(n)        显示后 n 行数据
```

```
# df.info()      显示数据的信息，包括列名、数据类型、缺失值等
# df.describe() 显示数据的基本统计信息，包括均值、方差、最大值、最小值等
# df.shape       显示数据的行数和列数
# '''
# print(df.head())
# print(df.tail())
# print(df.info())
# print(df.describe())
# print(df.shape)

'''
数据清洗

df.dropna()                           删除包含缺失值的行或列
df.fillna(value)                      将缺失值替换为指定的值
df.replace(old_value, new_value)      将指定值替换为新值
df.duplicated()                       检查是否有重复的数据
df.drop_duplicates()                  删除重复的数据
'''

'''
数据选择和切片

df[column_name]                                    选择指定的列;
df.loc[row_index, column_name]                     通过标签选择数据;
df.iloc[row_index, column_index]                   通过位置选择数据;
df.ix[row_index, column_name]                      通过标签或位置选择数据;
df.filter(items=[column_name1, column_name2])      选择指定的列;
df.filter(regex='regex')                           选择列名匹配正则表达式的列;
df.sample(n)                                       随机选择 n 行数据。
'''

'''
数组排序

df.sort_values(column_name)                                                  按照指定列
的值排序
df.sort_values([column_name1, column_name2], ascending=[True, False])        按照多个列
的值排序
Series.sort_values(axis=0, ascending=True)
DataFrame.sort_values(by, axis=0, ascending=True)                            by axis轴
上某个索引或索引列表
df.sort_index(axis=0, ascending=True)                                        按照索引排
序
'''
a = pd.DataFrame(np.arange(20).reshape(4, 5), index=['c', 'a', 'd', 'b'])
print(a)
print(a.sort_index())
print(a.sort_index(ascending=False))
print(a.sort_index(axis=1, ascending=False))
print(a.sort_values(2, ascending=False))
print(a.sort_values('a', axis=1, ascending=False))

# 若有NAN 排序统一置于末尾
```

```
a = pd.DataFrame(np.arange(20).reshape(4, 5), index=['c', 'a', 'd', 'b'])
b = pd.DataFrame(np.arange(12).reshape(3, 4), index=['a', 'b', 'c'])
print(a + b)
print((a+b).sort_values(2))
print((a+b).sort_values(2, ascending=False))

'''
数组的分组和聚合

df.groupby(column_name)                              按照指定列进行分组;
df.aggregate(function_name)                          对分组后的数据进行聚合操作;
df.pivot_table(values, index, columns, aggfunc)  生成透视表。
'''

'''
数据合并

pd.concat([df1, df2])    将多个数据框按照行或列进行合并;
pd.merge(df1, df2, on=column_name)  按照指定列将两个数据框进行合并。
'''

'''
数据选择和过滤

df.loc[row_indexer, column_indexer]      按标签选择行和列。
df.iloc[row_indexer, column_indexer]     按位置选择行和列。
df[df['column_name'] > value]            选择列中满足条件的行。
df.query('column_name > value')          使用字符串表达式选择列中满足条件的行。
'''

'''
数据统计和描述

df.describe()    计算基本统计信息，如均值、标准差、最小值、最大值等。
df.mean()        计算每列的平均值。
df.median()      计算每列的中位数。
df.mode()        计算每列的众数。
df.count()       计算每列非缺失值的数量。
.argmin()        计算数据最大值的索引位置(自动索引) 或被弃用
.argmax()        计算数据最小值的索引位置(自动索引)
.idxmin()        计算数据最大值的索引(自定义索引)
.idxmax()        计算数据最小值的索引(自定义索引)
'''

df = pd.read_json('related_data/data.json')
print(df)
# df = df.dropna()
# print(df)
df = df.fillna({'age':0, 'score':0})
print(df)
# 重命名列名
df = df.rename(columns={'name':'姓名', 'age':'年龄', 'gender':'性别', 'score':'成
绩'})
print(df)
```

```
# 按成绩排序
df = df.sort_values(by='成绩', ascending=False)
print(df)
# 按性别分组计算平均年龄和成绩
grouped = df.groupby('性别').agg({'年龄':'mean', '成绩':'mean'}) #
df.aggregate(function_name) 在这个例子中，对 年龄和成绩 应用了 mean 函数
print(grouped)
# 选择成绩大于90的行，并只保留姓名和成绩两列
sorted = df.loc[df['成绩'] >= 90, ['姓名' , '成绩']]
print(sorted)
print(df.count())
print(df.idxmax())

'''
累计统计分析函数

.cumsum()              前n个数的和
.cumprod()             前n个数的积
.cummax()              前n个数的最大值
.cummin()              前n个数的最小值

.rolling(w).sum()      依次计算相邻w个元素的和
.rolling(w).mean()     依次计算相邻w个元素的算术平均值
.rolling(w).var()      依次计算相邻w个元素的方差
.rolling(w).std()      依次计算相邻w个元素的标准差
.rolling(w).min()      依次计算相邻w个元素的最小值
.rolling(w).max()      依次计算相邻w个元素的最大值
'''

a = pd.DataFrame(np.arange(20).reshape(4, 5), index=['c', 'a', 'd', 'b'])
print(a)
print(a.cumsum() )
print(a.cumprod())
print(a.cummax() )
print(a.cummin() )
print(a.rolling(2).sum()) # 计算包括自己的前两项和 第一列前面没有数值 返回NAN
print(a.rolling(2).mean())
print(a.rolling(2).var())
print(a.rolling(2).std())
print(a.rolling(2).min())
print(a.rolling(2).max())
```

相关性分析

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

'''
```

```
df.cov()  计算协方差矩阵
df.corr() 计算相关系数矩阵 pearson spearman kendall系数

在pandas中，数据的相关性分析是通过计算不同变量之间的相关系数来了解它们之间的关系
数据相关性是一项重要的分析任务，可以帮助我们理解各个变量之间的关系
使用 corr() 方法计算数据集中每列之间的关系

df.corr(method='pearson', minperiods=1)
method (可选): 字符串类型，用于指定计算相关系数的方法。默认是 'pearson',
                还可以选择 'kendall' (Kendall Tau 相关系数) 或 'spearman' (Spearman
秩相关系数) 。
min_periods (可选): 表示计算相关系数时所需的最小观测值数量。默认值是 1，即只要有至少一个
非空值，就会进行计算。
                    如果指定了 min_periods，并且在某些列中的非空值数量小于该值，则相应
列的相关系数将被设为 NaN。

返回一个相关系数矩阵，矩阵的行和列对应数据框的列名，矩阵的元素是对应列之间的相关系数。

常见的相关性系数包括 Pearson 相关系数和 Spearman 秩相关系数:
Pearson 相关系数: 即皮尔逊相关系数，用于衡量了两个变量之间的线性关系强度和方向。它的取值
范围在 -1 到 1 之间，其中 -1 表示完全负相关，
                    1 表示完全正相关，0 表示无线性相关。可以使用 corr() 方法计算数据框
中各列之间的 Pearson 相关系数。
Spearman 相关系数: 即斯皮尔曼相关系数，是一种秩相关系数。用于衡量两个变量之间的单调关系，
即不一定是线性关系。
                它通过比较变量的秩次来计算相关性。可以使用 corr(method='spearman') 方
法计算数据框中各列之间的 Spearman 相关系数。
'''

data = {'A':[1, 2, 3, 4, 5], 'B':[5, 4, 3, 2, 1]}
df =pd.DataFrame(data)
# 计算 pearson 相关系数
correlation_matrix = df.corr()
print(correlation_matrix) # 因为数据集是线性相关的，因此主对角线值为1，副对角线值为-1完
全负相关
# 计算 spearman 秩相关系数
correlation_matrix = df.corr(method='spearman')
print(correlation_matrix) # 结果与pearman相关系数矩阵相同，因为两个变量之间完全负相关

# 可视化相关性    使用Seaborn库绘制更加精美的相关系数矩阵图像
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f') # 绘制热力
图 annot: 默认为False，为True的话，会在格子上显示数字
plt.show()
```

实例_房价涨幅与M2增幅的相关性

```
import pandas as pd
import matplotlib.pyplot as plt

hprice = pd.Series([3.04, 22.93, 12.75, 22.6, 12.33], index=['2008', '2009',
'2010', '2011', '2012'])
```

```
m2 = pd.Series([8.18, 18.38, 9.13, 7.82, 6.69], index=['2008', '2009', '2010',
'2011', '2012'])

plt.rcParams['font.family'] = 'Source Han Sans CN'
plt.plot(hprice, '-o')
plt.plot(m2, '-x')
plt.title(f'相关系数矩阵：{hprice.corr(m2)}')
plt.show()
```