PYTHON程序设计

计算机学院 王纯

十二 数据计算与分析

数据计算与分析

- Numpy
- Pandas
- Scipy

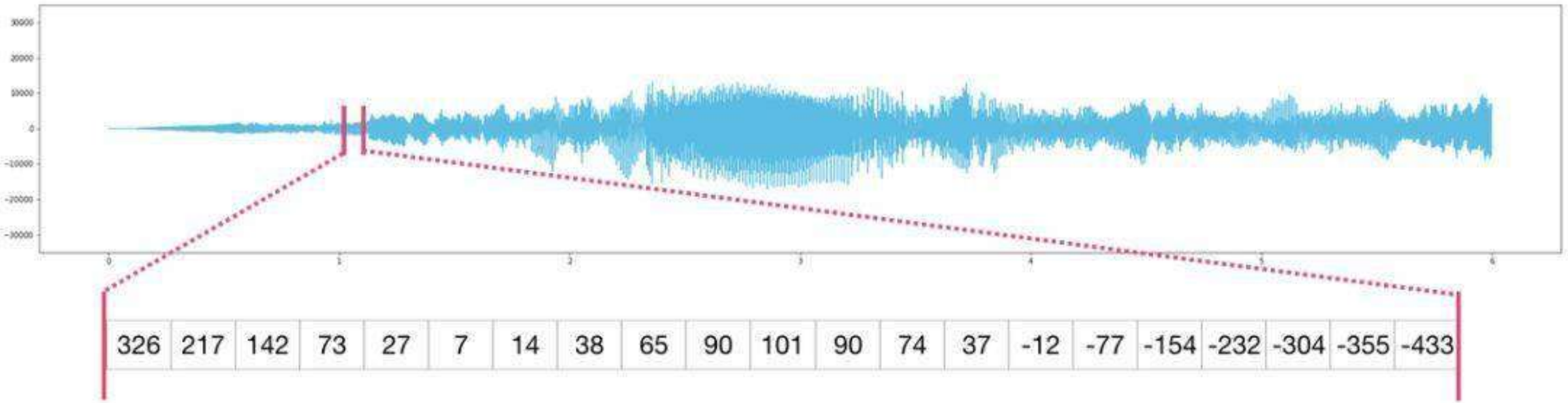
概述

- 随着计算机和智能移动终端的普及，以及互联网和物联网的广泛应用，使得数据的产生和收集都变得越来越容易。 每天各种设备都会产生海量的数据，被互联网、金融、通信等公司所收集，成为指导它们发展业务的重要资产。全球知名的咨询公司麦肯锡称：“数据，已经渗透到当今每一个行业和业务职能领域，成为重要的生产因素”。对海量的数据进行计算和分析，是Python运用的一个重点领域。

NUMPY-科学计算基础包

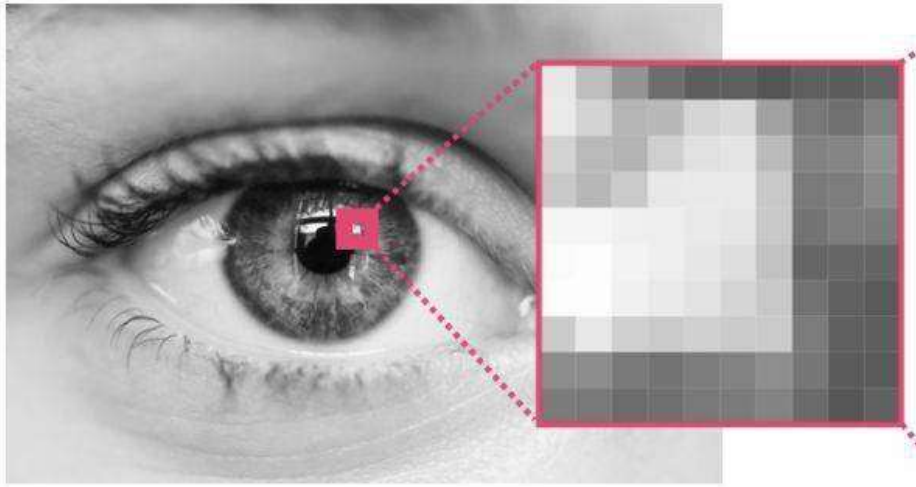
- NumPy，即 Numeric Python的缩写，是一个优秀的开源科学计算库，并已经成为 Python科学计算生态系统的重要组成部分
- NumPy为我们提供了丰富的数学函数、强大的多维数组对象以及优异的运算性能。 NumPy在保留 Python语言优势的同时大大增强了科学计算和数据处理的能力，非常适合用于大数据的分析和处理
  - 强大的N维数组结构
  - 精密复杂的函数
  - 可集成到C/C++和Fortran代码的工具
  - 有用的线性代数、傅里叶变换以及随机数能力
  - 读写基于数组的数据集的工具

一段音频的时间序列数据

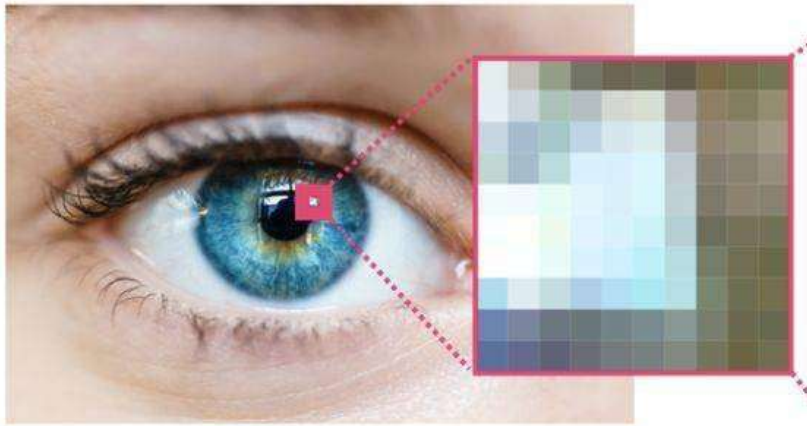


图像文件的像素矩阵

- 黑白图片的像素数据是灰度值（0-255）



230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97



	233	188	137	96	90	95	63	73	73	82
237	202	159	120	105	110	88	107	112	121	109
226	191	147	110	101	112	98	123	110	119	142	131
221	191	176	182	203	214	169	144	133	145	155	122
185	160	161	184	205	223	186	137	147	161	140	115
181	174	189	207	206	215	194	136	142	151	133	87
246	237	237	231	208	206	192	122	143	144	111	74
254	254	241	224	199	192	181	99	122	117	107	74
239	248	232	207	187	182	184	110	114	110	113	74
193	215	193	167	158	164	181	114	112	111	105	82
113	119	110	111	113	123	135	120	108	106	113	
93	97	91	103	107	111	122	112	104	114		

彩色图像的像素矩阵

- 增加一个维度表示颜色

- NumPy数组是一个多维数组对象，称为ndarray，通常被称 作数组。注意numpy.array和标准Python库类array.array并不相同，后者只处理一维数组和提供少量功能。
- numpy.array由两部分组成：实际的数据和描述这些数据的元数据。对于在数组中保存的实际数据，有如下规则：
  - NumPy数组的下标从0开始
  - 同一个NumPy数组中所有元素的类型必须是相同的

```
import numpy
import numpy as np
from numpy import *
from numpy import array, sin
```

- NumPy包的下载地址：https://pypi.python.org/pypi/numpy
- 也可以使用pip工具直接下载Numpy包或Pycharm自动安装
- 使用前一定要先导入NumPy包，导入的方法有如左几种
- 目前使用的版本是NumPy 1.21.4

NUMPY数据类型NDARRAY

列表

- 元素可以为多种类型

```
a = [1,8.0,”abc”,obj]
```

```
a = [0,1,2,3,4]
b = [5,6,7,8,9]
c = list()
for i in range(len(a)):
    c.append(a[i]*b[i])
```

VS

ndarray

- 元素必须为同一类型

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

```
a = numpy.array([0,1,2,3,4])
b = numpy.array([5,6,7,8,9])
c = a*b
```

ndarray支持的数据类型

名称	描述
bool	一个字节存储的布尔类型
int	所在平台决定其大小的整数类型（一般为int32或int64）
int8	一个字节大小，-128至127
int16	整数，-32768至32767
int32	整数，-2**31至2**32-1
int64	整数，-2**63至2**63-1
uint8	无符号整数，0至255
uint16	无符号整数，0至65535

名称	描述
uint32	无符号整数，0至2**32-1
uint64	无符号整数，0至2**64-1
float16	半精度浮点数：16位，正负号1位，指数5位，精度10位
float32	单精度浮点数：32位，正负号1位，指数8位，精度23位
float64或float	双精度浮点数：64位，正负号1位，指数11位，精度52位
complex64	复数，分别用两个32位浮点数表示实部和虚部
complex128或complex	复数，分别用两个64位浮点数表示实部和虚部

创建数组

共有5种创建数组的通用机制：

1) 通过其它Python结构（例如列表和元组）创建
2) numpy数组本身的数组创建对象（例如arange、ones、zeros等）
3) 从磁盘读取数组，无论是标准格式还是自定义格式
4) 通过使用字符串或缓冲区从原始字节创建数组
5) 使用特殊的库函数（例如随机函数）

代码	结果	说明
a = array([0,1,2,3,4])	[0,1,2,3,4]	创建一个一维数组，有5个元素，每个元素为整型
a1 = arange(5)	[0,1,2,3,4]	另外一种方法，效果与上面一种方法相同
a2 = arange(0, 6, 2)	[0,2,4]	arange函数的三个参数分别为：起始数(含)，终止数（不含），步长
b = array([(1.0, 2.0, 3.0), (4.0, 5.0, 6.0)])	[[1., 2., 3.] [4., 5., 6.]]	创建一个二维数组，2行3列，共有6个元 素，每个元素为浮点型64位
c = array([[1,2,3], [4,5,6]], dtype=complex)	[[1.+0.j, 2.+0.j, 3.+0.j], [4.+0.j, 5.+0.j, 6.+0.j]])	另外一个二维数组，2行3列，效果与上 面一种方法类似，但数据类型为 complex128
d = zeros((2,3))	[[0., 0., 0.], [0., 0., 0.]]	创建一个2行3列的二维数组，值全部为0

代码	结果	说明
e = ones((1,5))	[1., 1., 1., 1., 1.]	创建一个一维数组，值全为1
f = full((2,3,4),8)	[[[8, 8, 8, 8], [8, 8, 8, 8], [8, 8, 8, 8]], [[8, 8, 8, 8], [8, 8, 8, 8], [8, 8, 8, 8]]]	创建一个元素全是8的三维数组
g = eye(3,3)	[[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]]	创建3行3列的二维数组，对角线为1，其 它元素均为0
h = random.random((2,3))	[[0.98772846, 0.53233087, 0.81990949], [0.45654177, 0.25143958, 0.45555412]]	创建2行3列的二维数组，元素值为随机 数
I = np.arange(3) i.Repeat(2)	[0,0,1,1,2,2]	单个元素重复
j = np.arange(3) np.tile(j, 2) np.tile(j, (2,3))	[0, 1, 2, 0, 1, 2] [[0, 1, 2, 0, 1, 2, 0, 1, 2], [0, 1, 2, 0, 1, 2, 0, 1, 2]]	数组整体重复
k1 = np.linspace(0, 5, 5) k1 = np.linspace(0, 5, 5, endpoint=False)	[0. , 1.25, 2.5 , 3.75, 5.] [0., 1., 2., 3., 4.]	在指定的间隔内返回均匀间隔的数字，形成一个等差数列

NumPy支持**多维数组**，每个维度（dimensions）也称为轴 （axes）。维度的数量称为秩（rank），一维数组的秩为1， 二维数组的秩为2，以此类推。 数组的属性可以通过“数组名.属性名”或者“属性名(数组名)”查看，例如：a.ndim或者ndim(a)

```
a = array([0,1,2,3,4])
b = array( [ (1.0, 2.0, 3.0), (4.0, 5.0, 6.0) ] )
c = array( [ [1,2,3], [4,5,6] ], dtype=complex )
```

属性名	描述	数组a	数组b	数组c
ndim	维数，等于秩	1	2	2
shape	每个维度上的长度	5	2,3	2,3
size	数组元素的总个数	5	6	6
dtype	表示数组中元素类 型的对象	int32	float64	complex128
itemsize	数组中每个元素的 大小(以字节为单位)	4	8	16

数组输出

- 当打印输出一个数组时，NumPy以类似嵌套列表的形式显示它，遵循 以下规则：
  - 最后的轴从左到右打印
  - 次后的轴从顶向下打印
  - 剩下的轴从顶向下打印，每个切片通过一个空行与下一个隔开
- 可以使用print(a)打印出数组的内容，或者在python中直接输入数组的名称a。
- 一维数组被打印成行，二维数组被打印为矩阵，三维数组被打印为矩 阵列表。参见前面各个数组的结果列。

注意：如果一个数组太大了，无法完整输出，打印时NumPy将自动省 略中间部分而只打印角落的部分

```
>>> d = np.array([[11, 12, 13, 14, 15],
... [21, 22, 23, 24, 25],
... [31, 32, 33, 34, 35],
... [41, 42, 43, 44, 45],
... [51, 52, 53, 54, 55]])
```

注意，二维数组的第一个元素II
所在 的位置是0行0列。

```
>>> print(d[0, 1:4])
[12 13 14]
>>> print(d[1:4, 0])
[21 31 41]
>>> print(d[::2,::2])
[[11 13 15]
 [31 33 35]
 [51 53 55]]
>>> print(d[:, 1])
[12 22 32 42 52]
```

打印第一行部分元素

打印第一列部分元素

隔行隔列打印

打印每行的第一列

a = arange(10) #[0,1,2,3,4,5,6,7,8,9]

数组运算

代码	结果	说明
a.sum()	45	数组求和
a.min()	0	数组的最小值
a.max()	9	数组的最大值
a.cumsum()	[0,1,3,6,10,15,21,28,36,45]	输出结果中每一个元素的值等于数组前n 个元素的累加和
b=arange(0,100,10)	[0,10,20,30,40,50,60,70,80,90]	生成一个数组，元素值大于等于0，小于 100，步长为10
c = b[b>50]	[60,70,80,90]	只取出大于50的元素
d = b[b%3==0]	[0,30,60,90]	只取出B数组中能整除3的元素

a = array([10,20,30,40])， b = arange(4)

数组运算

代码	结果	说明
a+b	[10,21,32,43]	两个数组相加
a-b	[10,19,28,37]	两个数组相减
a*b	[0,20,60,120]	两个数组相乘
a/b	[inf,20.,15.,13.33333333]	两个数组相除，第一个元素为除0的结果
b**2	[0,1,4,9]	数组b元素的乘方
10*sin(a)	[-5.44021111, 9.12945251, -9.88031624, 7.4511316]	对a的元素求sin后再乘以10
a<21	[True, True, False, False]	判断a的元素值是否小于21，小于的显示为True, 不小于的显示为False
a.dot(b)	200	dot() 函数计算两个数组的点积，即向量乘法，相当于a1*b1+a2*b2+a3*b3+a4*b4，得到一个数值

数组索引

```
from numpy import *
a = arange(0, 100, 10)
print(a) # [ 0 10 20 30 40 50 60 70 80 90]
a[2] #20 a数组的第2个元素
a[2:5] #[20,30,40] a数组的第2个到第4个元素
indices = [1, 6, -2] #通过索引获取数组中的部分数据
print(a[indices]) # [10 60 80] 取a数组的正数第1和第6个，以及倒数第2个元素
mask = array([True,False,True,False,False,True,False,True,False,False])
print(a[mask]) # [0 20 50 70] 通过布尔值来索引，只输出布尔值为真的对应数据
```

形状调整

形状调整指的是数组维度方面的变化

```
from numpy import *
a = arange(16) #a是一个一维数组
b = a.reshape(4,4) #b是一个二维数组
c = a.reshape(2,2,2,2) #c是一个四维数组
```

注意，reshape后形成的新数组，和原数组共享同一段内存空间，并不会产生一个 新的地址空间。如果对其中a、b、c中任何一个数组的元素进行修改，其它2个数组 的内容也同时会发生变化。例如：b[0,0]=99，修改完毕后，a[0]和c[0,0,0,0]的值也会 变成99。

```
a = array([[n+m*10 for n in range(3)] for m in range(4)])
#a是一个4行3列的二维数组，[[0,1,2],[10,11,12],[20,21,22],[30,31,32]]
b = a. flatten() #b是一个向量[0,1,2,10,11,12,20,21,22,30,31,32]
b[0:4]=-1 #修改b的前4个元素值
print(a) #对b的修改并不会影响a，a的内容保持不变
printf(b) #b向量的内容修改为[-1,-1,-1,-1,11,12,20,21, 22,30,31,32]
```

使用flatten函数可以将高维数组转化为向量，和reshape不同的是， flatten函数会生成一份新的复制。

统计特征计算

a = array([0,1,2,3,4])

函数名	描述	结果
mean(a)	算术平均值	2.0
std(a)	标准差	1.1412135623730951
var(a)	方差	2.0
average(a.weights=[0,0.1,0.2,0.3,0.4])	加权平均值	3.0
median([1,3,100])	中位数	3

注：统计函数也能对多维数组的任意一个指定的维度进行统计计算。

常用函数

快速入门：https://docs.scipy.org/doc/numpy/user/quickstart.html
完整参考手册：https://docs.scipy.org/doc/numpy/reference/

类别	函数
创建数组	arange, array, copy, empty, empty_like, eye, fromfile, fromfunction, identity, linspace, logspace, mgrid, ogrid, ones, ones_like, zeros, zeros_like
转换	ndarray.astype, atleast_1d, atleast_2d, atleast_3d, mat
操作	array_split, column_stack, concatenate, diagonal, dsplit, dstack, hsplit, hstack, ndarray.item, newaxis, ravel, repeat, reshape, resize, squeeze, swapaxes, take, transpose, vsplit, vstack
判断	all, any, nonzero, where
排序	argmax, argmin, argsort, max, min, ptp, searchsorted, sort
运算	choose, compress, cumprod, cumsum, inner, ndarray.fill, imag, prod, put, putmask, real, sum
基本统计	cov, mean, std, var
基本线性代数	cross, dot, outer, linalg.svd, vdot

PANDAS-数据分析核心库

- Pandas 是基于NumPy的一种工具，它是为了解决数据分析任务而创建的。Pandas 包含大量的库和一些标准的数据模型，能够高效地操作中大型的数据集
- Pandas 主要定义了两种数据类型：**Series** 和 **DataFrame**，它们使得数据操作更加简单。Pandas提供了大量处理数据的函数和方法

Series是一种一维的数据类型，其中的每个元素都有各自的标签。

标签	0	1	2	3
数据	1	2	3	4

标签	‘a’	‘b’	‘c’	‘d’
数据	1	2	3.14	‘abc’

DataFrame是一个二维的、表格型的数据结构。

行/列标签	name	quality	price
0	pen	200	9.9
1	ruler	400	3.1
2	rubber	800	2.3

下载安装

Pandas包的下载地址：https://pypi.org/project/pandas/

也可以使用pip工具直接下载Pandas包或从PyCharm安装

使用前要先导入 Pandas 包，导入的方法有以下几种：

注：目前使用的版本是Pandas 1.3.4。

```
import numpy
import pandas as pd
from pandas import Series, DataFrame
```

Series 可以看做一个定长的有序字典，基本任意的一维数据都可以用来构造 Series 对象
创建一个Series 对象的基本方法是：

- Series(data[,index])。
- data是任何的数据类型，包括整数、字符串、浮点数、 Python对象等。
- index是索引，可以是数字类型的，也可以是非数字类型的。

Series创建和基本操作

创建一个Series对象，只有整型数据。
如果不给出索引，系统会自动给Series对象增加索引， 是从0开始的整数序列，这一点和列表（list）类似。

```
>>> s1 = Series([1,2,3])
>>> s1
0    1
1    2
2    3
dtype: int64
```

创建一个Series对象，数据有多种类型。

```
>>> s2 = Series([1,2,3.0,'abc'])
>>> s2
0      1
1      2
2    3.0
3    abc
dtype: object
```

Series创建和基本操作

创建一个Series对象，数据是整 型，索引是字符串类型

```
>>> s3 = Series(data=[1,2,3,4],index = ['a','b','x','y'])
>>> s3
a    1
b    2
x    3
y    4
dtype: int64
```

通过values查看数据，通过index查看索引

```
>>> s3.values
array([1, 2, 3, 4], dtype=int64)
>>> s3.index
Index(['a', 'b', 'x', 'y'], dtype='object')
```

可以使用字典对象来创建一个Series对象

```
>>> sd = {'a':1, 'b':2, 'c':3 , 'd':4}
>>> s4 = Series(sd)
>>> s4
a    1
b    2
c    3
d    4
dtype: int64
```

Series创建和基本操作

可以通过数据在列表中的位置来访问，也可以通过索引来访问Series对象中的数据。

```
>>> s3 = Series(data=[1,2,3,4],index = ['a','b','x','y'])
>>> s3[1]
2
>>> s3['b']
2
```

Series对象创建后，其索引是可以修改的

```
>>> sd = {'a':1, 'b':2, 'c':3 , 'd':4}
>>> s4 = Series(sd)
>>> s4
a    1
b    2
c    3
d    4
dtype: int64
```

```
>>> index1 = {'a','b','c','d','e'}
...
>>> s4 = Series(sd, index1)
...
>>> s4
c    3.0
b    2.0
e    NaN
d    4.0
a    1.0
dtype: float64
```

1.类型发生变化 2.顺序发生变化 3.空值NaN

Series创建和基本操作

可以使用reindex函数，来产生一个按照新的索引生成的Series对象

```
>>> s6 = s4.reindex(['a','b','c','d','e'],fill_value=888)
>>> s6
a    1.0
b    2.0
c    3.0
d    4.0
e    NaN
dtype: float64
```

注意，这个函数并不改变原有对象s4

可以通过函数isnull()来判断Series对象中的空值，返回 的是布尔值的一个序列

```
>>> s4.isnull()
c    False
b    False
e     True
d    False
a    False
dtype: bool
```

或者用pd.isnull(s4)

函数describe可以查看Series对象的快速统计摘要

```
>>> s=Series([1,2,3,4,100],index=['a','b','c','d','e'])
>>> s.describe()
count      5.000000
mean      22.000000
std       43.617657
min        1.000000
25%        2.000000
50%        3.000000
75%        4.000000
max      100.000000
dtype: float64
```

Series常用函数

s=Series([1,2,3,4,100],index=['a','b','c','d','e'])

函数	结果	说明
count	5	非 NA 值的数量
min,max	1,100	最小值和最大值
idxmin,idxmax	'a', 'e'	最小值和最大值的索引值
sum	110	求和
mean	22.0	均值
median	3.0	中位数
mad	31.2	根据均值计算平均绝对离差
var	1902.5	方差
std	43.617656975128774	标准差
copy		复制一个对象并返回这个新的对象

Series常用函数

s=Series([1,2,3,4,100],index=['a','b','c','d','e'])

函数	结果	说明
diff	a NaN b 1.0 c 1.0 d 1.0 e 96.0 dtype: float64	计算一阶差分（对时间序列很有用）
cumsum	a 1 b 3 c 6 d 10 e 110 dtype: int64	样本值的累计和
drop('b')	a 1 c 3 d 4 e 100 dtype: int64	删除原对象中index所指示的一个数据， 生成新的对象返回

函数	结果	说明
sort_values(ascending=False)	e 100 d 4 c 3 b 2 a 1 dtype: int64	按照数据的大小排序(倒序)
sort_values(ascending=True)	a 1 b 2 c 3 d 4 e 100 dtype: int64	按照索引排序（顺序）
s.replace([1,3],['one' ,'three'])	a one b 2 c three d 4 e 100 dtype: object	用'one'代替1，用'three'代替3

Series运算

加法并不是按照数据直接相加，而是根据索引对位相加，即先对齐索引后再运算。未对齐的索引，按照并集运算，且运算的结果将标记为NaN（缺失）。

```
>>> s=Series([1,2,3,4,100],index=['a','b','c','d','e'])
>>> s[1:] + s[:-1]
a     NaN
b     4.0
c     6.0
d     8.0
e     NaN
dtype: float64
```

s[1:]

索引	数据
b	2
c	3
d	4
e	100

s[:-1]

索引	数据
a	1
b	2
c	3
d	4

s[1:]+s[:-1]

索引	数据
a	NaN
b	4
c	6
d	8
e	NaN

DATAFRAME的创建

DataFrame是一种二维的数据结构，非常接近于电子表格或者类似 mysql 数据库的形式。
它的竖列称之为 columns，横行称之为 index。 可以使用 dict 来定义一个DataFrame对象

```
>>> data = {"name":["pen","ruler","rubber"], "quantity":[200,400,800] ,"price":[9.9, 3.1, 2.3]}
>>> f1 = DataFrame(data)
>>> f1
     name  quantity  price
0     pen       200    9.9
1   ruler       400    3.1
2  rubber       800    2.3
```

DataFrame创建

定义了一个DataFrame对象后，还可以修改其列的顺序，并对行增加索引

```
>>> f1 = DataFrame(data, columns=['name', 'price', 'quantity', 'sold'], index=['a','b','c'])
>>> f1
     name  price  quantity sold
a     pen    9.9       200  NaN
b   ruler    3.1       400  NaN
c  rubber    2.3       800  NaN
```

可以通过columns和index函数来查看DataFrame对象的属性

```
>>> f1.columns
Index(['name', 'price', 'quantity', 'sold'], dtype='object')
>>> f1.index
Index(['a', 'b', 'c'], dtype='object')
```

DataFrame使用

取出DataFrame对象的指定列：对象名.列名 或者 对象名[列名]

```
>>> f1.name
a       pen
b     ruler
c    rubber
Name: name, dtype: object
```

取出的列，其实就是一个**Series对象**。
可以将DataFrame理解为是由多个Series对象组成的字典

通过列名，可以对该列的数据进行统一的修改，或者有条件的修改

```
>>> f1.sold=100
>>> f1
     name  price  quantity  sold
a     pen    9.9       200   100
b   ruler    3.1       400   100
c  rubber    2.3       800   100
>>> f1[f1.quantity>500]
     name  price  quantity  sold
c  rubber    2.3       800   100
```

DataFrame使用

通过函数loc和iloc，可以取出指定行

```
>>> f1.iloc[1]
name         ruler
price          3.1
quantity       400
sold           100
Name: b, dtype: object
```

访问某行某列的元素

```
>>> f1['name']['b']
'ruler'
>>> f1['sold']['b']=150
<input>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata
 .org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

DataFrame使用

按照列的数值排序

```
>>> f1.sort_values(by='price',ascending=True)
     name  price  quantity  sold
c  rubber    2.3       800   100
b   ruler    3.1       400   150
a     pen    9.9       200   100
```

```
>>> f1.sort_values(by=['sold','price'],ascending=False,inplace=True)
>>> f1
     name  price  quantity  sold
b   ruler    3.1       400   150
a     pen    9.9       200   100
c  rubber    2.3       800   100
```

注意inplace参数，这个参数如果设置为True，则表示修改将会影响原来的对象。如果为缺省值False，并不会改变原有对象 的内容。其它很多函数中，inplace参数也是同样的用法。

按照索引排序

```
>>> f1.sort_index(axis =0 ,ascending=False)
     name  price  quantity  sold
c  rubber    2.3       800   100
b   ruler    3.1       400   150
a     pen    9.9       200   100
```

```
>>> f1.sort_index(axis =1 ,ascending=False)
   sold  quantity  price    name
b   150       400    3.1   ruler
a   100       200    9.9     pen
c   100       800    2.3  rubber
```

DataFrame打开csv文件

将cvs文件直接导入到DataFrame对象中

```
>>> f2= pd.read_csv("score.csv")
>>> f2
      name  english  math  c++  python
0      sam       78    93   91      90
1  michael       82    89   84      87
2     lucy       92    80   79      83
3     rose       87    85   90      92
```

```
1  name,english,math,c++,python
2  sam,78,93,91,90
3  michael,82,89,84,87
4  lucy,92,80,79,83
5  rose,87,85,90,92
```

- python使用的默认编码为utf-8，不支持中文。
- 如果文件中含有中文，在读取csv或者 xls文件时，在read函数中写入参 数encoding="gbk"即可。
- 如果 gbk也不能解码，可以使用收录字符更广的"gb18030"解码。
- 如果是excel文件，可以使用read_excel函数。

DataFrame打开csv文件

```
>>> f2.sum()#默认按照列分别求和
name        sammichaellucyrose
english                    339
math                       347
c++                        344
python                     352
dtype: object
```

常用函数，和Series对象中的函数类似

```
>>> f2.mean(axis=1) #按照行求均值（只包括整型或浮点型数据）
<input>:1: FutureWarning: Dropping of nuisance colum
0    88.0
1    85.5
2    83.5
3    88.5
dtype: float64
```

describe函数：查看快速统计摘要

```
>>> f2.describe()
         english       math         c++    python
count   4.000000   4.000000   4.000000   4.00000
mean   84.750000  86.750000  86.000000  88.00000
std     6.075909   5.560276   5.597619   3.91578
min    78.000000  80.000000  79.000000  83.00000
25%    81.000000  83.750000  82.750000  86.00000
50%    84.500000  87.000000  87.000000  88.50000
75%    88.250000  90.000000  90.250000  90.50000
max    92.000000  93.000000  91.000000  92.00000
```

DataFrame常用函数

类别	函数名	说明
读入数据	pd.read_table(filename)	从限定分隔符的文本文件导入数据
	pd.read_sql(query,connection_object)	从SQL表/库导入数据
	pd.read_json(json_string)	从JSON格式的字符串导入数据
	pd.read_html(url)	解析URL、字符串或者HTML文件，抽取其中的tables表格
	pd.read_clipboard()	从你的粘贴板获取内容，并传给read_table()
导出文件	df.to_csv(filename)	导出数据到CSV文件
	df.to_excel(filename)	导出数据到Excel文件
	df.to_sql(table_name, connection_object)	导出数据到SQL表
	df.to_json(filename)	以Json格式导出数据到文本文件
查看数据	df.head(n)	查看DataFrame对象的前n行
	df.tail(n)	查看DataFrame对象的最后n行
	df.shape()	查看行数和列数

DataFrame常用函数

类别	函数名	说明
查看数据	df.info()	查看索引、数据类型和内存信息
	s.value_counts(dropna=False)	查看Series对象的唯一值和计数
数据处理	df.columns = ['a','b','c']	重命名列名
	pd.isnull()	检查DataFrame对象中的空值，并返回一个 Boolean数组
	pd.notnull()	检查DataFrame对象中的非空值，并返回一个 Boolean数组
	df.dropna()	删除所有包含空值的行
	df.dropna(axis=1)	删除所有包含空值的列
	df.dropna(axis=1,thresh=n)	删除所有小于n个非空值的行
	df.fillna(x)	用x替换DataFrame对象中所有的空值
	s.astype(float)	将Series中的数据类型更改为float类型
	s.replace(1,'one')	用‘one’代替所有等于1的值
	用'one'代替1，用'three'代替3	用'one'代替1，用'three'代替3

DataFrame常用函数

类别	函数名	说明
数据处理	df.rename(columns=lambda x: x + 1)	批量更改列名
	df.rename(columns={'old_name': 'new_name'})	选择性更改列名
	df.set_index('column_one')	更改索引列
	df.rename(index=lambda x: x + 1)	批量重命名索引
	df[df[col] > 0.5]	选择col列的值大于0.5的行
	df.sort_values(col1)	按照列col1排序数据，默认升序排列
	df.sort_values(col2, ascending=False)	按照列col1降序排列数据
	df.sort_values([col1,col2], ascending=[True,False])	先按列col1升序排列，后按col2降序排列数据
	df.groupby(col)	返回一个按列col进行分组的Groupby对象
	df.groupby([col1,col2])	返回一个按多列进行分组的Groupby对象
	df.groupby(col1)[col2]	返回按列col1进行分组后，列col2的均值

DataFrame常用函数

类别	函数名	说明
	df.pivot_table(index=col1, values=[col2,col3], aggfunc=max)	创建一个按列col1进行分组，并计算col2和 col3的最大值的数据透视表
	df.groupby(col1).agg(np.mean)	返回按列col1分组的所有列的均值
合并数据	df1.append(df2)	将df2中的行添加到df1的尾部
	df.concat([df1, df2],axis=1)	将df2中的列添加到df1的尾部
	df1.join(df2,on=col1,how='inner')	对df1的列和df2的列执行SQL形式的join

SCIPY-科学计算标准问题域的模块集合

- SciPy是构建在NumPy的基础之上的，它提供了许多操作 NumPy数组的函数。它是一款方便易用、专为科学和工程设计的python工具包，它包括了统计、优化、整合以及线性代数模块、傅里叶变换、信号和图像图例、常微分方差的求解等

SciPy子包

子包	说明
cluster	聚类
contants	物理和数学常数
fftpack	傅里叶变换
integrate	积分和常微分方程求解器
interpolate	插值
io	数据输入和输出
linalg	线性代数例程
ndimage	n维图像包

子包	说明
odr	正交距离回归
optimize	优化
signal	信号处理
sparse	稀疏矩阵
spatial	空间数据结构和算法
special	特殊函数
stats	统计

这些子包在使用前需要单独导入，例如： from scipy import ndimage, optimize

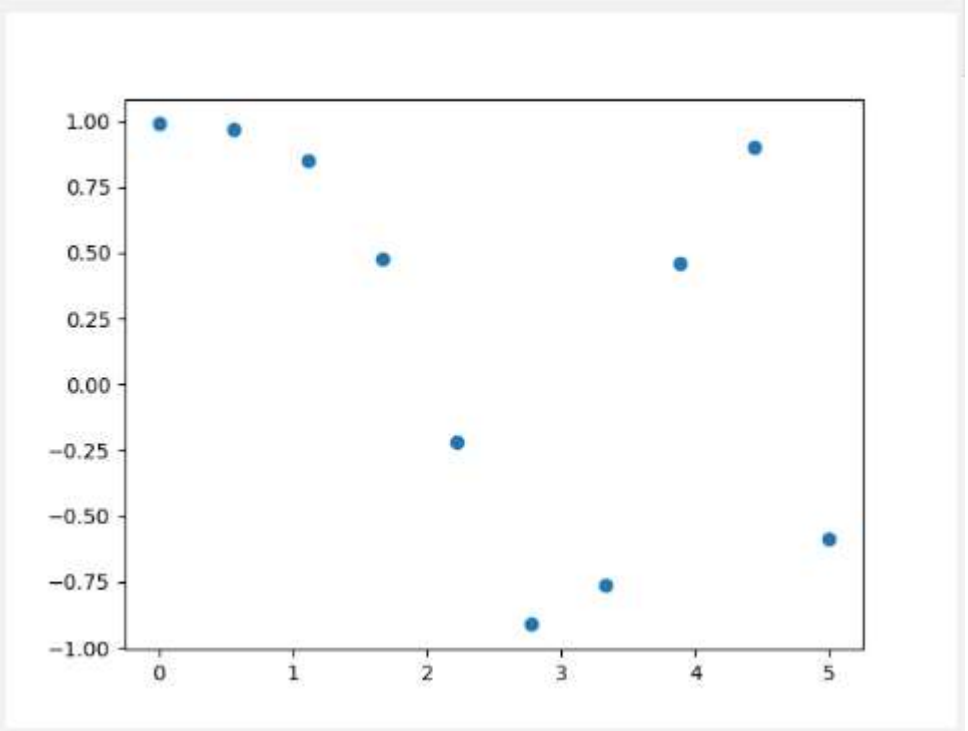
在子包constants中，定义了一些科学计算经常会被用到的常数或常量

编号	常量	说明	编号	常量	说明
1	pi	PI值	16	minute	一分钟秒数（60）
2	golden	黄金比例	17	day	一天的秒数
3	c	真空中的光速	18	inch	一英寸的米数
4	speed_of_light	真空中的光速	19	micron	一微米的米数
5	h	普朗克常数	20	light_year	一光年的米数
6	Planck	普朗克常数	21	atm	一标准大气压的帕斯卡数
7	G	牛顿的引力常数	22	acre	一英亩的平方米数
8	e	基本电荷	23	liter	一升的立方米数
9	R	摩尔气体常数	24	gallon	一加仑的立方米数
10	electron_mass/m_e	电子质量	25	kmh	一公里/小时的米/秒数
11	proton_mass/m_p	质子质量	26	degree_Fahrenheit	一华氏度的开尔文数
12	neutron_mass/m_n	中子质量	27	eV	一电子伏特的焦耳数
13	gram	一克的千克数	28	hp	一马力的瓦特数
14	atomic_mass	原子质量常数	29	dyn	一达因的牛顿数
15	degree	弧度	30	lambda2nu	将波长转换为光频率

插值INTERPOLATE

插值是在离散数据的基础上补插连续函数，使得这条连续曲线通过 全部给定的离散数据点。 插值是离散函数逼近的重要方法，利用它可通过函数在有限个点处的取值状况，估算出函数在其他点处的近似值。

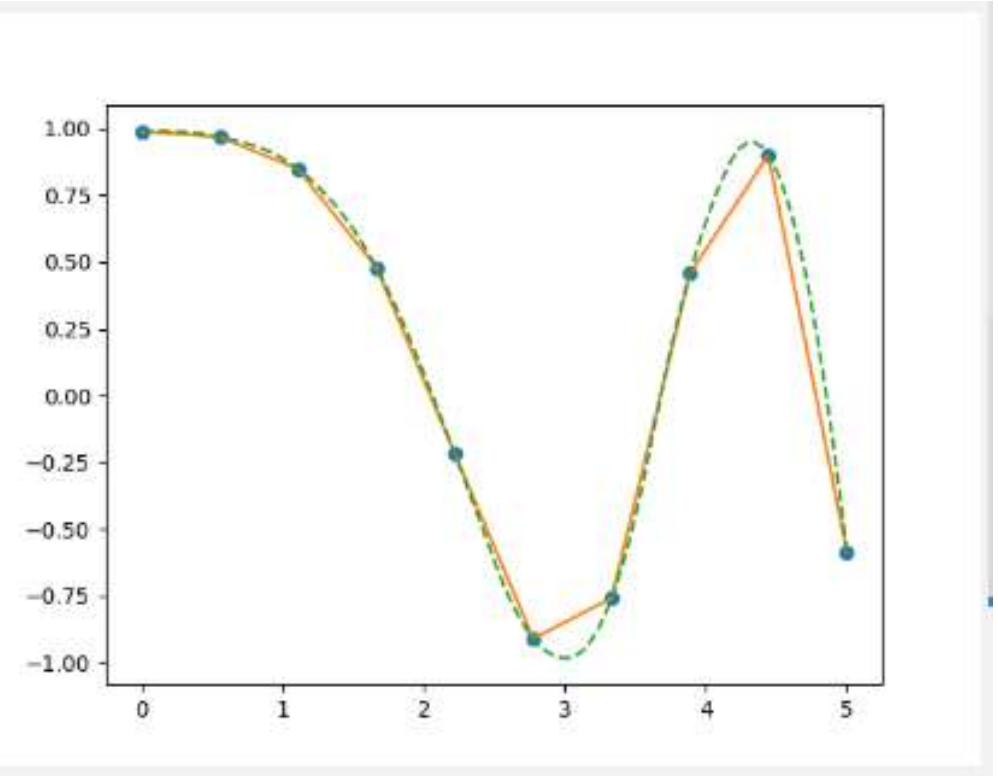
```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
x = np.linspace(0, 5, 10) #在0到5之间生成10个数据
y = np.sin(x**2/3+8) #y是x的某种三角函数
plt.plot(x, y,'o') #使用matplotlib库生成图形
plt.show()
```



插值

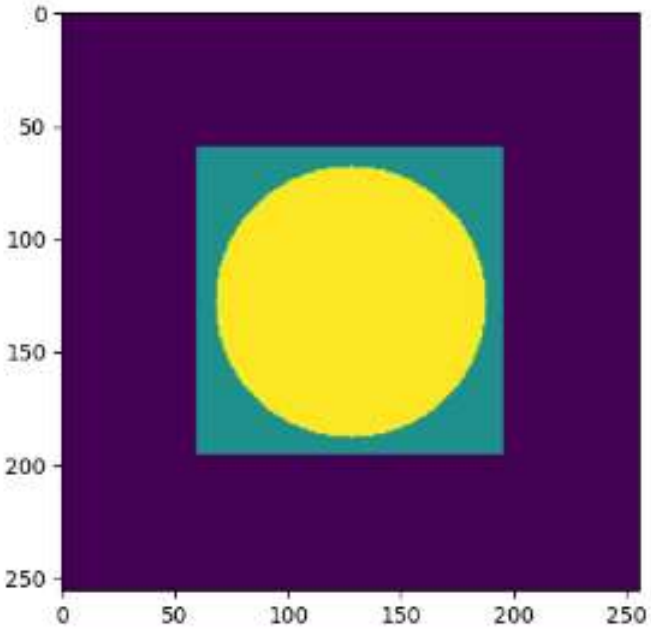
scipy.interpolate中的interp1d类，是一种创建基于固定数据点的函数的便捷 方法，可以使用插值方法在给定数据定义区域内的任意位置评估该函数。

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt

x = np.linspace(0, 5, 10) #在0到5之间生成10个数据
y = np.sin(x**2/3+8) #y是x的某种三角函数
plt.plot(x, y,'o') #使用matplotlib库生成图形
plt.show()

f1 = interpolate.interp1d(x, y,kind = 'linear') #线性插值
f2 = interpolate.interp1d(x, y, kind = 'quadratic') # 2阶样条插值
xnew = np.linspace(0, 5,100)
plt.plot(x, y, 'o', xnew, f1(xnew), '-', xnew, f2(xnew), '--')
plt.show()
```



图像处理：使用子包NDIMAGE

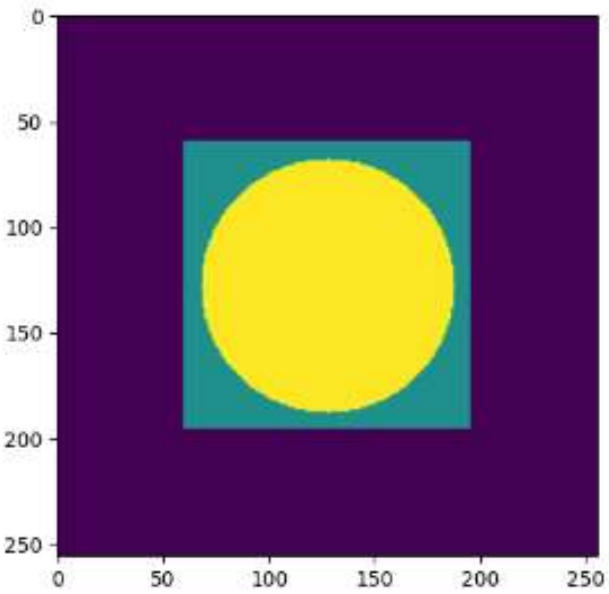
```
import numpy as np
import math
from matplotlib import pyplot as plt
from scipy import ndimage

im = np.zeros((256, 256))
im[60:-60, 60:-60] = 100 #在中间画出一个蓝色正方形
for i in range(0,256): #在中间画出一个黄色的圆形
        for j in range(0,256):
         if math.sqrt((i-128)**2+(j-128)**2)<= 60:
             im[i,j]=200

plt.imshow(im)
plt.show()
```

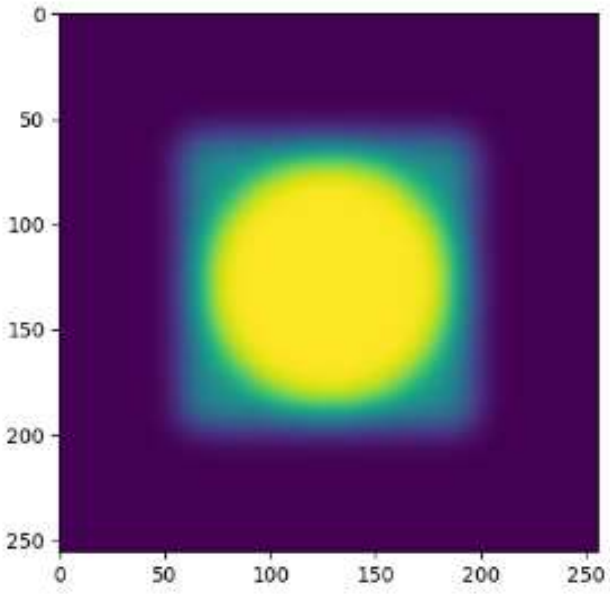


使用ndimage中的gaussian_filter函数，对图像做模糊处理

模糊处理

```
im = ndimage.gaussian_filter(im, 8) #增加高斯模糊效果，
plt.imshow(im)
plt.show()
```

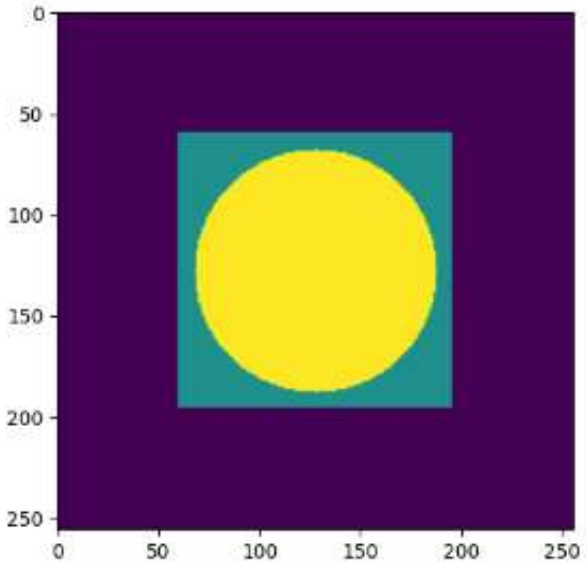


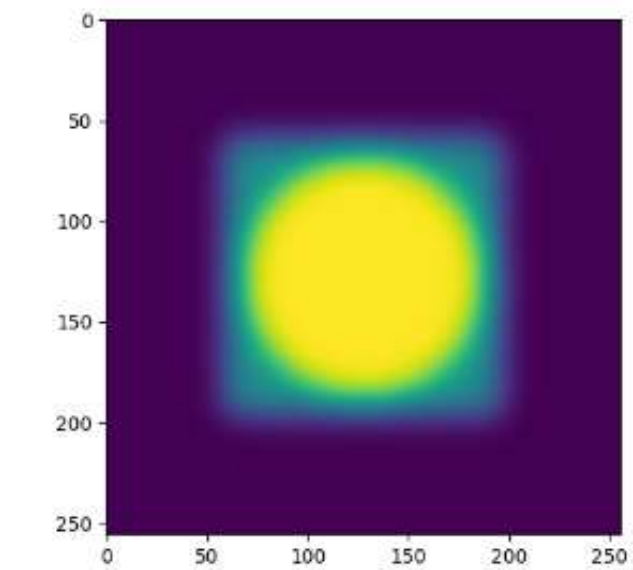


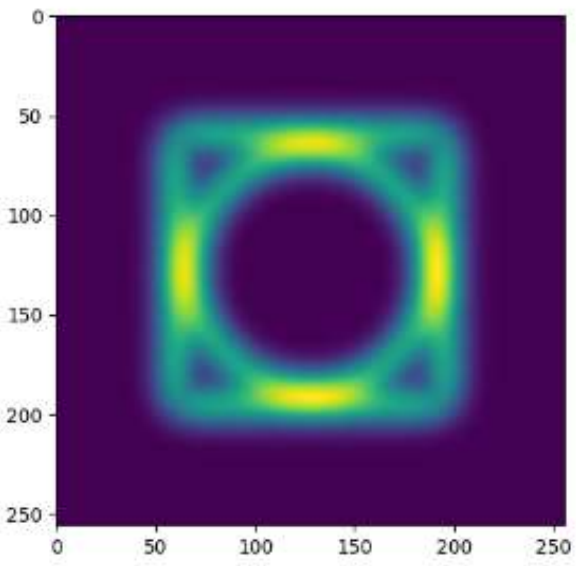
用ndimage 中的sobel函数，对上述模糊的图像，进行边缘检测

边缘检测

```
sx = ndimage.sobel(im, axis = 0, mode = 'reflect')
sy = ndimage.sobel(im, axis = 1, mode = 'reflect')
sob = np.hypot(sx, sy)
plt.imshow(sob)
plt.show()
```







数据计算与分析

- Numpy
- Pandas
- Scipy

谢谢