Python 进阶

吴强

2020.07

NumPy 包

NumPy

NumPy(Numerical Python) 是 Python 语言的一个扩展程序库:

- 支持大量的维度数组与矩阵运算
  - 强大的N维数组对象 ndarray
- 针对数组运算提供大量的数学函数库
  - 整合 C/C++/Fortran 代码的工具
  - 线性代数、傅里叶变换、随机数生成等功能

NumPy 为开放源代码并且由许多协作者共同维护开发

- NumPy 的前身 Numeric 最早是由 Jim Hugunin 等开发.
- 2005 年, 结合了 Numarray 的特色, 并加入了其它扩展而开发 NumPy.

https://numpy.org/

NumPy

NumPy 通常与 SciPy (Scientific Python) 和 Matplotlib 一起使用

- SciPy 是一个开源的 Python 算法库和数学工具包
  - SciPy 包含的模块有最优化, 线性代数, 积分, 插值, 特殊函数,快速傅里叶变换, 信号处理和图像处理, 常微分方程求解等工具
- Matplotlib 是 NumPy 的可视化操作界面
- NumPy + SciPy + Matplotlib 的组合可以实现MATLAB的许多功能, 常学习数据科学或者机器学习.

ndarray 对象

- ndarray 是 一种 n 维数组对象
  - ndarray 对象是用于存放同类型元素的多维数组
  - 索引以 0 下标为开始
  - 每个元素在内存中都有相同存储大小的区域
- 语法:

```
numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)
```

- 参数:
  object: 数组或嵌套的数列
  dtype: 数组元素的数据类型
  copy: 对象是否需要复制
  order: 计算机内存中存储元素的顺序, C为行方向, F为列方向, A为任意方向(默认)
  subok: 默认返回一个与基类类型一致的数组
  ndmin: 指定生成数组的最小维度
- ndarray 对象由计算机内存的连续一维部分组成, 并结合索引模式, 将每个元素映射到内存块中的一个位置. 内存块以行顺序(C样式)或列顺序(FORTRAN或MATLAB风格, F样式)来保存元素.

ndarray 对象

```
import numpy as np
# one dimension
a = np.array([1,2,3])
print(a)

# multi dimension
a = np.array([[1,  2],  [3,  4]])
print(a)

# minimum dimension
a = np.array([1,  2,  3, 4, 5], ndmin =  2)
print(a)

# dtype
a = np.array([1,  2,  3], dtype = complex)
print(a)
```

NumPy 数据类型

内置数据类型

名称	描述
bool_	布尔型数据类型(True 或者 False)
int_	默认的整数类型(类似于 C 语言中的 long, int32 或 int64)
intc	与 C 的 int 类型一样, 一般是 int32 或 int 64
intp	用于索引的整数类型(类似于 C 的 ssize_t, 一般情况下仍然是 int32 或 int64)
int8	字节(-128 to 127)
int16	整数(-32768 to 32767)
int32	整数(-2147483648 to 2147483647)
int64	整数(-9223372036854775808 to 9223372036854775807)
uint8	无符号整数(0 to 255)
uint16	无符号整数(0 to 65535)
uint32	无符号整数(0 to 4294967295)
uint64	无符号整数(0 to 18446744073709551615)

NumPy 数据类型

内置数据类型

名称	描述
float_	float64 类型的简写
float16	半精度浮点数, 包括：1 个符号位, 5 个指数位, 10 个尾数位
float32	单精度浮点数, 包括：1 个符号位, 8 个指数位, 23 个尾数位
float64	双精度浮点数, 包括：1 个符号位, 11 个指数位, 52 个尾数位
complex_	complex128 类型的简写, 即 128 位复数
complex64	复数, 表示双 32 位浮点数(实数部分和虚数部分)
complex128	复数, 表示双 64 位浮点数(实数部分和虚数部分)

NumPy 数据类型

- numpy 的数值类型实际上是数据类型对象 (dtype)的实例, 并对应唯一的字符.
  - 数据类型对象用来描述与数组对应的内存区域如何使用
    - 数据的类型 (整数, 浮点数或者 Python 对象)
    - 数据的大小 (使用多少个字节存储)
    - 数据的字节顺序 (小端法"<"或大端法">")
    - 在结构化类型的情况下, 字段的名称, 每个字段的数据类型和每个字段所取的内存块的部分
- 语法:

```
numpy.dtype(object, align, copy)
```

- 参数:
  object: 要转换为的数据类型对象
  align: 如果为 true, 填充字段使其类似 C 的结构体。
  copy: 复制 dtype 对象, 如果为 false, 则是对内置数据类型对象的引用

NumPy 数据类型

```
import numpy as np

# use build-in types
dt = np.dtype(np.int32)
print(dt)

# int8, int16, int32, int64 can be represented using 'i1', 'i2','i4','i8'
dt = np.dtype('i4')
print(dt)

# little endian
dt = np.dtype('<i4')
print(dt)

# define a structured data type, create an array with defined dtype and get data
dt = np.dtype([('age',np.int8)])
print(dt)
a = np.array([(10,),(20,),(30,)], dtype = dt)
print(a)
print(a['age'])

# another example
student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
print(student)
a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype = student)
print(a)
print(a['name'], a['age'], a['marks'])
```

NumPy 数据类型

- 每个内建类型都有一个唯一定义它的字符代码

字符	对应类型
b	布尔型
i	(有符号) 整型
u	无符号整型 integer
f	浮点型
c	复数浮点型
m	时间间隔 timedelta
M	日期时间 datetime
O	(Python) 对象
S, a	(byte-)字符串
U	Unicode
V	原始数据 (void)

NumPy 数组属性

- NumPy中, 数组的维数称为秩(rank)
  - 一维数组的秩为 1
  - 二维数组的秩为 2
  - 以此类推
- NumPy中, 每一个线性的数组称为一个轴(axis)
  - 很多时候可以声明 axis
  - axis=0, 表示沿着第 0 轴进行操作
  - axis=1, 表示沿着第 1 轴进行操作

NumPy 的数组中比较重要 ndarray 对象属性有:

NumPy 数组属性

属性	说明
ndarray.ndim	秩, 即轴的数量或维度的数量
ndarray.shape	数组的维度, 对于矩阵, n 行 m 列
ndarray.size	数组元素的总个数, 相当于 .shape 中 n*m 的值
ndarray.dtype	ndarray 对象的元素类型
ndarray.itemsize	ndarray 对象中每个元素的大小, 以字节为单位
ndarray.flags	ndarray 对象的内存信息
ndarray.real	ndarray元素的实部
ndarray.imag	ndarray 元素的虚部
ndarray.data	包含实际数组元素的缓冲区, 由于一般通过数组的索引获取元素, 所以通常不需要使用这个属性

NumPy 数组属性

```
import numpy as pd
# creat a 1d array and print the rank
a = np.arange(24)
print(a)
print(a.ndim)

# create an array and print its shape, the return is a tuple
a = np.array([[1,2,3],[4,5,6]])
print(a.shape)

# use shape to adjust shape of the array
a = np.array([[1,2,3],[4,5,6]])
print(a)
a.shape = (3,2)
print (a)

# adjust shape of the array
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print(b)

# use itemsize to check the size of each item (unit: byte)
y = np.array([1,2,3,4,5], dtype = np.float64)
print (y.itemsize)
```

创建数组

ndarray 数组除了可以用 ndarray 来创建外, 也可以通过以下方式来创建:

- numpy.empty
  - 创建一个指定形状(shape), 数据类型(dtype) 且未初始化的数组
- numpy.zeros
  - 创建指定大小的数组, 数组元素以 0 来填充.
- numpy.ones
  - 创建指定大小的数组, 数组元素以 1 来填充.
- 语法:

```
numpy.empty(shape, dtype = float, order = 'C')
numpy.zeros(shape, dtype = float, order = 'C')
numpy.ones(shape, dtype = None, order = 'C')
```

- 参数:
  shape: 数组形状
  dtype: 数据类型，可选
  order: 计算机内存中的存储元素的顺序, 有"C"和"F"两个选项

创建数组

```
import numpy as np
# create an emtpty array
x = np.empty([3,2], dtype = int)
print (x)

# default type
x = np.zeros(5)
print(x)

# custom type
y = np.zeros((5,), dtype = np.int)
print(y)
z = np.zeros((2,2), dtype = [('x', 'i4'), ('y', 'i4')])
print(z)

# use default type
x = np.ones(5)
print(x)

# custom type
x = np.ones([2,2], dtype = int)
print(x)
```

从已有的数组创建数组

numpy.asarray 可用于从已有的数组创建数组

- 语法:

```
numpy.asarray(a, dtype = None, order = None)
```

- 参数:
  a: 任意形式的输入参数, 可以是列表, 列表的元组, 元组, 元组的元组, 元组的列表, 多维数组等
  dtype: 数据类型，可选
  order: 计算机内存中的存储元素的顺序, 有"C"和"F"两个选项

从已有的数组创建数组

```
import numpy as py
# from a list
x =  [1,2,3]
a = np.asarray(x)
print (a)

# from a tuple
x =  (1,2,3)
a = np.asarray(x)
print (a)

# from a tuple list
x =  [(1,2,3),(4,5)]
a = np.asarray(x)
print (a)

# set the dtype
x =  [1,2,3]
a = np.asarray(x, dtype =  float)
print (a)
```

从已有的数组创建数组

numpy.frombuffer 用于实现动态数组, 它接受 buffer 输入参数, 以流的形式读入转化成 ndarray 对象

- 语法:

```
numpy.frombuffer(buffer, dtype = float, count = -1, offset = 0)
```

- 参数:
  buffer: 以流的形式读入的任意对象, buffer 是字符串的时候,Python3 默认 str 是 Unicode 类型, 要转成 bytestring 需在原 str 前加上 b
  dtype: 返回数组的数据类型
  count: 读取的数据数量, 默认为-1, 读取所有数据
  offset: 读取的起始位置, 默认为0

```
import numpy as np

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

从已有的数组创建数组

numpy.fromiter 从可迭代对象中建立 ndarray 对象, 返回一维数组.

- 语法:

```
numpy.fromiter(iterable, dtype, count=-1)
```

- 参数:
  iterable: 可迭代对象
  dtype: 返回数组的数据类型
  count: 读取的数据数量, 默认为-1, 读取所有数据

```
import numpy as np
x = np.fromiter(range(5), dtype=float)
print(x)
```

从数值范围创建数组

numpy.arange 可以创建数值范围并返回 ndarray 对象.

- 语法:

```
numpy.arange(start, stop, step, dtype)
```

- 参数:
  start: 起始值, 默认为0
  stop: 终止值(不包含)
  step: 步长，默认为1
  dtype: 返回数组的数据类型, 如果没有提供, 则会使用输入数据的

```
import numpy as np

x = np.arange(5)
print (x)

# set dtype
x = np.arange(5, dtype =  float)
print (x)

# set start, end and step
x = np.arange(10,20,2)
print (x)
```

从数值范围创建数组

numpy.linspace 函数用于创建一个等差数列.

- 语法:

```
np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
```

- 参数:
  start: 序列的起始值
  stop: 序列的终止值, 如果endpoint为true, 该值包含于数列中
  num: 要生成的等步长的样本数量, 默认为50
  endpoint: 该值为 true 时, 数列中中包含stop值, 反之不包含,默认是True
  retstep: 如果为 True 时, 生成的数组中会显示间距, 反之不显示
  dtype: 返回数组的数据类型

```
import numpy as np
a = np.linspace(1,10,10)
print(a)
a = np.linspace(1,1,10)
print(a)
a = np.linspace(10, 20,  5, endpoint =  False)
print(a)
a = np.linspace(1,10,10,retstep= True)
print(a)
b = np.linspace(1,10,10).reshape([10,1])
print(b)
```

从数值范围创建数组

numpy.logspace 函数用于创建一个于等比数列.

- 语法:

```
np.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)
```

- 参数:
  start: 序列的起始值为: base ** start
  stop: 序列的终止值为: base ** stop. 如果endpoint为true,该值包含于数列中
  num: 要生成的等步长的样本数量, 默认为50
  endpoint: 该值为 true 时, 数列中中包含stop值, 反之不包含,默认是True
  base: 对数 log 的底数
  dtype: 返回数组的数据类型

```
import numpy as np
a = np.logspace(1.0, 2.0, num =  10)
print (a)

a = np.logspace(0,9,10,base=2)
print (a)
```

切片和索引

ndarray对象的内容可以通过索引或切片来访问和修改

- 索引或切片与 Python 中 list 的操作一样
  - 基于 0 - n 的下标进行索引
- 切片也可以通过内置的 slice 函数, 并设置 start, stop 及 step 进行

```
import numpy as np

a = np.arange(10)
s = slice(2,7,2)
print (a[s])

a = np.arange(10)
print(a[2:7:2])
print(a[5])
print(a[2:])
print(a[2:5])

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

切片和索引

- 切片还可以包括省略号 "..." 来使选择元组的长度与数组的维度相同
  - 如果在行位置使用省略号, 它将返回包含行中元素的 ndarray

```
import numpy as np

a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print(a)
print(a[...,1])   # second column
print(a[1,...])   # second row
print(a[...,1:])  # second to the end columns
```

高级索引

NumPy 比一般的 Python 序列提供更多的索引方式. 除用整数和切片的索引外, 数组可以由整数数组索引, 布尔索引及花式索引.

- 整数数组索引

```
import numpy as np
# specified elements as a 1D array
x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0,1,2],  [0,1,0]]
print(y)

# specified elements as a 2D array
x = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])
rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
y = x[rows,cols]
print(y)

# using the ... to represent the whole row/column
a = np.array([[1,2,3], [4,5,6],[7,8,9]])
b = a[1:3, 1:3]
c = a[1:3,[1,2]]
d = a[...,1:]
print(b)
print(c)
print(d)
```

高级索引

- 布尔索引: 布尔索引通过布尔运算(如比较运算符)来获取符合指定条件的元素的数组.

```
import numpy as np

# elements greater than 5 as a 1D array
x = np.array([[  0,  1,  2],[  3,  4,  5],[  6,  7,  8],[  9,  10,  11]])
print('elements greater than 5:')
print (x[x >  5])

# use ~ to filter NaN
a = np.array([np.nan,  1,2,np.nan,3,4,5])
print(a[~np.isnan(a)])

# filter non complex number
a = np.array([1,  2+6j,  5,  3.5+5j])
print (a[np.iscomplex(a)])
```

高级索引

- 花式索引: 花式索引指的是利用整数数组进行索引.
  - 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
  - 对于使用一维整型数组作为索引
    - 如果目标是一维数组, 那么索引的结果就是对应位置的元素
    - 如果目标是二维数组, 那么就是对应下标的行

```
import numpy as np

x=np.arange(32).reshape((8,4))
print(x)
print('\n')

# use ordered index
print(x[[4,2,1,7]])
print('\n')

# use reverse ordered index
print(x[[-4,-2,-1,-7]])
print('\n')

# use multiple indices
print(x[np.ix_([1,5,7,2],[0,3,1,2])])
```

广播

广播(Broadcast)是 Numpy 对不同形状(shape)的数组进行数值计算的方式, 对数组的算术运算通常在相应的元素上进行.

- 如果两个数组 a 和 b 形状相同, 即满足 a.shape == b.shape, 那么 a*b 的结果就是 a 与 b 数组对应元素相乘.

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

- 当运算中的 2 个数组的形状不同时, Numpy 将自动触发广播机制.

```
import numpy as np

a = np.array([[ 0, 0, 0],
        [10,10,10],
        [20,20,20],
        [30,30,30]])
b = np.array([1,2,3])
print(a + b)
```

广播

```
import numpy as np

a = np.array([[ 0, 0, 0],
              [10,10,10],
              [20,20,20],
              [30,30,30]])
b = np.array([1,2,3])
bb = np.tile(b, (4, 1))    # MATLAB repmat(b, [4, 1])
print(bb)
print('\n')
print(a + bb)
```

迭代数组

NumPy 迭代器对象 numpy.nditer 提供了一种灵活访问一个或者多个数组元素的方式. 迭代器最基本的任务的可以完成对数组元素的访问.

```
import numpy as np

a = np.arange(6).reshape(2,3)
print (a)
print ('\n')
for x in np.nditer(a):
    print (x, end=", " )
print ('\n')
```

以上实例不是使用标准 C 或者 Fortran 顺序, 选择的顺序是和数组内存布局一致的, 这样做是为了提升访问的效率, 默认是行序优先(row-major order, 或者说是 C-order).

迭代数组

- 遍历顺序
  - 默认情况下迭代器只需访问每个元素, 而无需考虑其特定顺序.
  - 可以通过迭代上述数组的转置来看到这一点, 并与以 C 顺序访问数组转置的 copy 方式做对比.

```
import numpy as np

a = np.arange(6).reshape(2,3)
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=", " )
print ('\n')
```

从上述例子可以看出, a 和 a.T 的遍历顺序是一样的, 也就是他们在内存中的存储顺序也是一样的, 但是 a.T.copy(order = 'C') 的遍历结果是不同的, 那是因为它和前两种的存储方式是不一样的, 默认是按行访问.

迭代数组

```
a = np.arange(0,60,5)
a = a.reshape(3,4)
print('Original Array: ')
print(a)
print('\n')

print('After Transpose:')
b = a.T
print(b)
print('\n')

print('In C-form order: ')
c = b.copy(order='C')
print(c)
for x in np.nditer(c):
    print (x, end=", " )
print('\n')

print('In F-form order: ')
c = b.copy(order='F')
print(c)
for x in np.nditer(c):
    print (x, end=", " )
```

迭代数组

- 控制遍历顺序
  - 可以通过显式设置, 来强制 nditer 对象使用某种顺序
  - 列序优先: Fortran order

```
for x in np.nditer(a, order='F')
```

  - 行序优先: C order

```
for x in np.nditer(a.T, order='C')
```

```
a = np.arange(0,60,5)
a = a.reshape(3,4)
print('Original Array: ')
print (a)
print ('\n')

print('In C-form order: ')
for x in np.nditer(a, order =  'C'):
    print (x, end=", " )
print ('\n')

print('In F-form order: ')
for x in np.nditer(a, order =  'F'):
    print (x, end=", " )
```

迭代数组

- 修改数组中元素的值
  - nditer 对象有另一个可选参数 op_flags. 默认情况下, nditer 将视待迭代遍历的数组为只读对象(read-only), 为了在遍历数组的同时, 实现对数组元素值得修改, 必须指定 read-write 或者 write-only 的模式.

```
a = np.arange(0,60,5)
a = a.reshape(3,4)
print('Original Array: ')
print (a)
print ('\n')

for x in np.nditer(a, op_flags=['readwrite']):
    x[...]=2*x
print ('Modified Array: ')
print (a)
```

迭代数组

- 使用外部循环
  - nditer类的构造器拥有flags参数, 它可以接受下列值:
    c_index: 可以跟踪 C 顺序的索引
    f_index: 可以跟踪 Fortran 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

```
a = np.arange(0,60,5)
a = a.reshape(3,4)
print('Original Array: ')
print (a)
print ('\n')
print ('Using External Loop: ')
for x in np.nditer(a, flags =  ['external_loop'], order =  'F'):
    print (x, end=", " )
```

迭代数组

- 广播迭代
  - 如果两个数组是可广播的, nditer 组合对象能够同时迭代它们.
  - 假设数组 a 的维度为 3X4, 数组 b 的维度为 1X4 , 则使用以下迭代器, 数组 b 被广播到 a 的大小.

```
a = np.arange(0,60,5)
a = a.reshape(3,4)
print  ('First Array:')
print (a)
print  ('\n')

print ('Second Array:')
b = np.array([1,  2,  3,  4], dtype =  int)
print (b)
print ('\n')

print ('Boardcasted Array: ')
for x,y in np.nditer([a,b]):
    print ("(%d, %d)"  %  (x,y), end=", " )
```

数组操作

Numpy 中包含了一些函数用于处理数组, 主要包含以下几类:

- 修改数组形状
- 翻转数组
- 修改数组维度
- 连接数组
- 分割数组
- 数组元素的添加与删除

修改数组形状

numpy.reshape 在不改变数据的条件下修改形状

- 语法:

```
numpy.reshape(arr, newshape, order='C')
```

- 参数:
  arr: 要修改形状的数组
  newshape: 整数或者整数数组, 新的形状应当兼容原有形状
  order: 计算机内存中存储元素的顺序, C为行方向, F为列方向, A为原顺序(默认)

```
a = np.arange(8)
print ('Original Array: ')
print (a)
print ('\n')

b = a.reshape(4,2)
print ('Reshaped Array: ')
print (b)
```

修改数组形状

numpy.ndarray.flat 是一个数组元素迭代器. 对数组中每个元素都进行处理, 可以使用flat属性, 该属性是一个数组元素迭代器.

```
a = np.arange(9).reshape(3,3)
print('Original Array')
print(a)
print('\n')

for row in a:
    print(row)
print('\n')

for element in a.flat:
    print(element)
```

修改数组形状

numpy.ndarray.flatten 返回一份数组拷贝, 对拷贝所做的修改不会影响原始数组.

- 语法:

```
ndarray.flatten(order='C')
```

- 参数:
  order: 计算机内存中存储元素的顺序, C为行方向, F为列方向, A为原顺序, K 元素在内存中的出现顺序.

```
a = np.arange(8).reshape(2,4)

print ('Original Array: ')
print (a)
print ('\n')

print ('Flattened Array (in C-style): ')
print (a.flatten())
print ('\n')

print ('Flattened Array (in F-style):')
print (a.flatten(order = 'F'))
```

修改数组形状

numpy.ravel() 展平的数组元素, 顺序通常是"C风格", 返回的是数组视图, 修改会影响原始数组.

- 语法:

```
numpy.ravel(a, order='C')
```

- 参数:
  order: 计算机内存中存储元素的顺序, C为行方向, F为列方向, A为原顺序, K 元素在内存中的出现顺序.

```
a = np.arange(8).reshape(2,4)

print('Original Array: ')
print(a)
print('\n')

print ('After using ravel():')
print (a.ravel())
print ('\n')

print ('After using ravel() in F-form:')
print (a.ravel(order = 'F'))
```

翻转数组

numpy.transpose 函数用于对换数组的维度.

- 语法:

```
numpy.transpose(arr, axes)
```

- 参数:
  arr: 要操作的数组
  axes: 整数列表, 对应维度, 通常所有维度都会对换

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

print('Original Array: ')
print (a )

print('After Transpose: ')
print (np.transpose(a))
```

翻转数组

numpy.ndarray.T 类似 numpy.transpose.

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

print('Original Array: ')
print (a )

print('After Transpose: ')
print (a.T)
```

翻转数组

numpy.swapaxes 函数用于交换数组的两个轴.

- 语法:

```
numpy.swapaxes(arr, axis1, axis2)
```

- 参数:
  arr：输入的数组
  axis1：对应第一个轴的整数
  axis2：对应第二个轴的整数

```
a = np.arange(8).reshape(2,2,2)

print('Original Array: ')
print (a)
print ('\n')
# swap axes 2 and axes 0

print ('After swapaxes(): ')
print (np.swapaxes(a, 2, 0))
```

数组元素的添加与删除

numpy.resize 函数返回指定大小的新数组.

- 如果新数组大小大于原始大小, 则包含原始数组中的元素的副本.
- 语法:

```
numpy.resize(arr, shape)
```

- 参数:
  arr: 要修改大小的数组
  shape: 返回数组的新形状

数组元素的添加与删除

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

print ('Array a:')
print (a)

print ('Shape of Array a:')
print (a.shape)


b = np.resize(a, (3,2))

print ('Resized Array:')
print (b)

print ('Shape of Resized Array:')
print (b.shape)

print ('Resized with a larger shape: ')
c = np.resize(a,(3,3))
print (c)
```

数组元素的添加与删除

numpy.append 函数在数组的末尾添加值.

- 追加操作会分配整个数组, 并把原来的数组复制到新数组中.
- 语法:

```
numpy.append(arr, values, axis=None)
```

- 参数:
  arr: 输入数组
  values: 要向arr添加的值
  axis: 默认为 None. 当axis无定义时, 返回总是为一维数组. 当axis为0时，列数要相同, 当axis为1时, 行数要相同.

数组元素的添加与删除

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

print ('Array a:')
print (a)

print ('Append elements to array:')
print (np.append(a, [7,8,9]))

print ('Append elements to array along axis 0: ')
print (np.append(a, [[7,8,9]],axis = 0))

print ('Append elements to array along axis 1: ')
print (np.append(a, [[5,5,5],[7,8,9]],axis = 1))
```

数组元素的添加与删除

numpy.insert 函数在给定索引之前, 沿给定轴在输入数组中插入值.

- 如果值的类型转换为要插入, 则它与输入数组不同. 插入没有原地的, 函数会返回一个新数组.
- 如果未提供轴, 则输入数组会被展开.
- 语法:

```
numpy.insert(arr, obj, values, axis)
```

- 参数:
  arr: 输入数组
  obj: 在其之前插入值的索引
  values: 要插入的值
  axis: 沿着它插入的轴，如果未提供，则输入数组会被展开

数组元素的添加与删除

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

print ('The original Array')
print (a)

print ('Without axis, the array is flatten before insert')
print (np.insert(a,3,[11,12]))

print ('With axis, the inserted data is brocasted to fit the array')

print ('along axis 0')
print (np.insert(a,1,[11],axis = 0))

print ('along axis 1')
print (np.insert(a,1,11,axis = 1))
```

数组元素的添加与删除

numpy.delete 函数返回从输入数组中删除指定子数组的新数组. 与 insert() 函数的情况一样. 如果未提供轴参数, 则输入数组将展开.

- 语法:

```
numpy.delete(arr, obj, axis)
```

- 参数:
  arr: 输入数组
  obj: 可以被切片, 整数或者整数数组, 表明要从输入数组删除的子数组
  axis: 沿着它删除给定子数组的轴, 如果未提供, 则输入数组会被展开

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

print ('Array a:')
print (a)

print ('Without param Axis, the array is expanded first')
print (np.delete(a,5))

print ('Delete second column')
print (np.delete(a,1,axis = 1))
```

数组元素的添加与删除

numpy.unique 函数用于去除数组中的重复元素.

- 语法:

```
numpy.unique(arr, return_index, return_inverse, return_counts)
```

- 参数:
  arr: 输入数组, 如果不是一维数组则会展开
  return_index: 如果为true, 返回新列表元素在旧列表的位置, 并以列表形式储
  return_inverse: 如果为true, 返回旧列表元素在新列表的位置, 并以列表形式储
  return_counts: 如果为true, 返回去重数组中的元素在原数组中的出现次数

数组元素的添加与删除

```
a = np.array([5,2,6,2,7,5,6,8,2,9])

print ('Array a:')
print (a)

print ('Uniqued array:')
u = np.unique(a)
print (u)

print ('indices of uniqued array elements in old array')
u,indices = np.unique(a, return_index = True)
print (indices)

print ('indices of old array elements in the unique array')
u,indices = np.unique(a,return_inverse = True)
print (indices)

print ('reconstruct the old array using indices:')
print (u[indices])

print ('counts of uniqued array elements in the old array:')
u,indices = np.unique(a,return_counts = True)
print (indices)
```

数学函数

NumPy 包含大量的各种数学运算的函数, 包括三角函数, 算术运算的函数, 复数处理函数等.

- 三角函数:
  - sin(), cos(), tan()
  - arcsin(), arccos(), arctan() 是 sin(), cos(), tan() 的反三角函数。
  - numpy.degrees() 函数将弧度转换为角度.

```
a = np.array([0,30,45,60,90])
print ('sin():')
# convert to rad first
print (np.sin(a*np.pi/180))
print ('cos():')
print (np.cos(a*np.pi/180))
print ('tan():')
print (np.tan(a*np.pi/180))
```

数学函数

```
a = np.array([0,30,45,60,90])
print ('sin():')
sin = np.sin(a*np.pi/180)
print (sin)
print ('arcsin(), retrun in rad')
inv = np.arcsin(sin)
print (inv)
print ('in degree')
print (np.degrees(inv))
print ('cos():')
cos = np.cos(a*np.pi/180)
print (cos)
print ('arccos(), retrun in rad')
inv = np.arccos(cos)
print (inv)
print ('in degree')
print (np.degrees(inv))
print ('tan():')
tan = np.tan(a*np.pi/180)
print (tan)
print ('arctan():')
inv = np.arctan(tan)
print (inv)
print ('in degree:')
print (np.degrees(inv))
```

数学函数

- 舍入函数:
  - numpy.around(), numpy.floor(), numpy.ceil()
- numpy.around() 函数返回指定数字的四舍五入值.
  - 语法:

```
numpy.around(a,decimals)
```

  - 参数:
    a: 数组
    decimals: 舍入的小数位数. 默认值为0. 如果为负, 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  ('Original Array:')
print (a)
print ('Rounded:')
print (np.around(a))
print (np.around(a, decimals =  1))
print (np.around(a, decimals =  -1))
```

数学函数

- numpy.floor() 返回小于或者等于指定表达式的最大整数, 即向下取整.

```
a = np.array([-1.7,  1.5,  -0.2,  0.6,  10])
print ('Original Array:')
print (a)
print ('Floor of Array:')
print (np.floor(a))
```

- numpy.ceil() 返回大于或者等于指定表达式的最小整数, 即向上取整.

```
a = np.array([-1.7,  1.5,  -0.2,  0.6,  10])
print ('Original Array:')
print (a)
print ('Ceiling of Array:')
print (np.ceil(a))
```

数学函数

- 算术函数(加减乘除): add(), subtract(), multiply(), divide()
  - 需要注意的是数组必须具有相同的形状或符合数组广播规则

```
a = np.arange(9, dtype = np.float_).reshape(3,3)
print ('Array a:')
print (a)
print ('Array b:')
b = np.array([10,10,10])
print (b)
print ('Add:')
print (np.add(a,b))
print ('Subtract:')
print (np.subtract(a,b))
print ('Multiply:')
print (np.multiply(a,b))
print ('Divide:')
print (np.divide(a,b))
```

数学函数

- numpy.reciprocal() 函数返回参数逐元素的倒数.

```
a = np.array([0.25,  1.33,  1,  100])
print ('Original Array:')
print (a)
print ('Use reciprocal():')
print (np.reciprocal(a))
```

- numpy.power() 函数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.

```
a = np.array([10,100,1000])
print  ('Original Array:')
print (a)
print ('Use power():')
print (np.power(a,2))
print ('Another Array:')
b = np.array([1,2,3])
print (b)
print ('Use power():')
print (np.power(a,b))
```

数学函数

- numpy.mod() 计算输入数组中相应元素的相除后的余数.
- 函数 numpy.remainder() 也产生相同的结果.

```
a = np.array([10,20,30])
b = np.array([3,5,7])
print ('Array a:')
print (a)
print ('Array b:')
print (b)
print ('Use mod():')
print (np.mod(a,b))
print ('Use remainder():')
print (np.remainder(a,b))
```

统计函数

- 最小/大元素
  - numpy.amin() 用于计算数组中的元素沿指定轴的最小值
  - numpy.amax() 用于计算数组中的元素沿指定轴的最大值

```
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print ('Original Array:')
print (a)
print ('Use amin():')
print (np.amin(a,1))
print ('Use amin() again:')
print (np.amin(a,0))
print ('Use amax():')
print (np.amax(a))
print ('Use amax() again:')
print (np.amax(a, axis =  0))
```

统计函数

- numpy.ptp()函数计算数组中元素最大值与最小值的差(最大值 - 最小值)

```
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print ('Original Array:')
print (a)
print ('Use ptp():')
print (np.ptp(a))
print ('Along axis 1, use ptp():')
print (np.ptp(a, axis =  1))
print ('Along axis 0, use ptp()')
print (np.ptp(a, axis =  0))
```

统计函数

- numpy.percentile() 百分位数是统计中使用的度量, 表示小于这个值的观察值的百分比.
  - 语法:

```
numpy.percentile(a, q, axis)
```

  - 参数:
    a: 输入数组
    q: 要计算的百分位数, 在 0 ~ 100 之间
    axis: 沿着它计算百分位数的轴

```
a = np.array([[10, 7, 4], [3, 2, 1]])
print ('Original Array:')
print (a)

print ('Use percentile():')
# 50%
print (np.percentile(a, 50))

# axis 0, along columns
print (np.percentile(a, 50, axis=0))

# axis 1, along rows
print (np.percentile(a, 50, axis=1))

# keep dimensions
print (np.percentile(a, 50, axis=1, keepdims=True))
```

统计函数

- numpy.median() 函数用于计算数组 a 中元素的中位数

```
a = np.array([[30,65,70],[80,95,10],[50,90,60]])
print ('Original Array:')
print (a)
print ('Use median():')
print (np.median(a))
print ('Along axis 0, use median():')
print (np.median(a, axis =  0))
print ('Along axis 1, use median():')
print (np.median(a, axis =  1))
```

- numpy.mean() 函数返回数组中元素的算术平均值, 如果提供了轴, 则沿其计算. 算术平均值是沿轴的元素的总和除以元素的数量.

```
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print ('Original Array:')
print (a)
print ('Use mean():')
print (np.mean(a))
print ('Use mean() along axis 0:')
print (np.mean(a, axis =  0))
print ('Use mean() along axis 1:')
print (np.mean(a, axis =  1))
```

统计函数

- numpy.average() 函数根据在另一个数组中给出的各自的权重计算数组中元素的加权平均值. 该函数可以接受一个轴参数. 如果没有指定轴, 则数组会被展开. 加权平均值即将各数值乘以相应的权数, 然后加总求和得到总体值, 再除以总的单位数.

```
a = np.array([1,2,3,4])
print ('Original Array:')
print (a)
print ('Use average(), without weight:')
print (np.average(a))

wts = np.array([4,3,2,1])
print ('Weight:')
print (wts)

print ('Use average(), with weight:')
print (np.average(a,weights = wts))

# If returned is true, sum of weights is returned
print ('Sum of weights:')
print (np.average([1,2,3,  4],weights =  [4,3,2,1], returned =  True))
```

统计函数

- 在多维数组中, 可以指定用于计算的轴.

```
a = np.arange(6).reshape(3,2)
print ('Original Array:')
print (a)
print ('Weight:')
wt = np.array([3,5])
print ('Use average(), with weight:')
print (np.average(a, axis =  1, weights = wt))
print ('Use average(), with weight and return sum of weights:')
print (np.average(a, axis =  1, weights = wt, returned =  True))
```

统计函数

- 标准差是一组数据平均值分散程度的一种度量, 标准差是方差的算术平方根.
  - 标准差公式: std = sqrt(mean((x - x.mean())**2))

```
print (np.std([1,2,3,4]))
```

- 方差: 统计中的方差(样本方差)是每个样本值与全体样本值的平均数之差的平方值的平均数, 即 mean((x - x.mean())** 2).

```
print (np.var([1,2,3,4]))
```

Matplib 包

Matplotlib

Matplotlib是一个Python 2D绘图库. 它能让使用者很轻松地将数据图形化, 并且提供多样化的输出格式.

Matplotlib可用于Python脚本, Python和IPython Shell, Jupyter 笔记本, Web应用程序服务器和四个图形用户界面工具包.

Matplotlib 可以生成图表, 直方图, 功率谱, 条形图, 误差图, 散点图等.

为了简单绘图, 其中的 pyplot 模块提供了类似于MATLAB的界面, 尤其是与IPython结合使用时.

https://matplotlib.org/

Matplotlib 相关概念

matplotlib中的所有内容都是按层次结构组织的

- 层次结构的顶部是matplotlib “状态机环境”，由matplotlib.pyplot模块提供.
- 函数用于将绘图元素(线, 图像, 文本等)添加到当前图形中的当前轴.
- pyplot的状态机环境的行为类似于MATLAB, 对于有matlab经验的用户来说应该是熟悉的.
  - Figure: 画布
  - Axes: 画轴/子图
  - Axis: 坐标轴
  - Artist: 包括Text, Line2D 等所有对象
- 以 Matplotlib 画图的数据 np.array 以及类似array 的类, 如pandas 数据对象等. 画图之前最好将数据转化为 np.array.

Pyplot

- 创建绘图
- 格式化绘图
- 使用关键字绘图
- 使用类别变量绘图
- 设置线的性质
- 多个画布和画轴
- 设置文本
  - 文本中设置数学表达式
  - 标注文本
- 非线性坐标轴

在函数调用中保留各种状态, 以便跟踪当前图形和绘图区域等内容, 并且绘图函数指向当前轴.

其他类型图

- 普通图
- 散点图
- 条形图
- 等高线图
- 灰度图
- 饼状图
- 量场图

Panads 包

Pandas

Pandas 是一个 Python 的包, 提供快速, 灵活和富有表现力的数据结构, 旨在使"关系"或"标记"数据的使用既简单又直观.

- 它的目标是成为用Python进行实际的、真实的数据分析的基础高级模块.
- 它还有更宏远的目标: 成为超过任何语言的最强大最灵活的开源数据分析/操作工具.
- pandas 非常适合许多不同类型的数据:
  - 具有异构类型列的表格数据, 如SQL表或Excel电子表格
  - 有序和无序(不一定是固定频率)时间序列数据
  - 具有行和列标签的任意矩阵数据(均匀类型或异构)
  - 任何其他形式的观察/统计数据集
- pandas 的两个主要数据结构, Series(1维)和DataFrame(2维).
- pandas 建立在NumPy之上, 旨在与许多其他第三方库完美地集成在科学计算环境中.

数据结构

Series是一维标记的数组, 能够保存任何数据类型(整数, 字符串，浮点数, Python对象等), 轴标签统称为索引.

- 创建 Series 的基本方法是调用:

```
s = pd.Series(data, index=index)
```

  - data可以有很多不同的东西: 一个Python词典, 一个ndarray, 标量值
  - 传递的索引是轴标签列表
    - 如果data是ndarray, 则索引的长度必须与数据的长度相同. 如果没有传递索引, 将创建一个具有值的索引。[0, ..., len(data) - 1]
    - 当数据是dict, 并且未传递Series索引时, 则索引将按dict的插入顺序排序(Python > = 3.6, Pandas > = 0.23), 或词汇顺序的dict键列表(Python < 3.6或Pandas < 0.23). 如果传递索引, 则将拉出与索引中的标签对应的数据中的值.
    - 如果data是标量值，则必须提供索引. 将重复该值以匹配索引的长度.

数据结构

- Series的行为和ndarray非常的相似, 并且是大多数NumPy函数的有效参数. 但是. 切片等操作也会对索引进行切片.
- 像NumPy数组一样, Pandas的Series有一个dtype.
- Series.array将序列包装成array. 当需要在没有索引的情况下执行某些操作时, 访问该数组非常有用.
- 虽然 Series 是ndarray, 如果你需要一个真正的 ndarray, 那么使用 Series.to_numpy(). Series.to_numpy()会返回NumPy ndarray.
- Series是类似dict的数据类型, Series类似于固定大小的dict, 可以通过索引标签获取和设置值.
- 矢量化操作和标签对齐Series
- Series和ndarray之间的主要区别在于Series之间的操作会根据标签自动对齐数据. 因此, 可以在不考虑所涉及的Series是否具有相同标签的情况下编写计算.
- 未对齐 Series 之间的操作结果将包含所涉及的索引的并集. 如果在一个 Series 或另一个 Series 中找不到标签, 则结果将标记为缺失NaN.

数据结构

- Series 还可以有一个name属性. name在许多情况下, Series 会自动分配, 特别是在获取1D DataFrame切片时. 可以使用pandas.Series.rename()方法重命名 Series.

数据结构

数据帧/DataFrame是一个二维标记数据结构, 具有可能不同类型的列.

- 可以将其视为电子表格或SQL表, 或Series对象的字典.
- 它是最常用的pandas对象.
- 与Series一样, DataFrame接受许多不同类型的输入.
- 除了数据, 还可以选择传递索引(行标签)和列(列标签)参数. 如果传递索引和/或列, 则可以保证生成的DataFrame的索引和/或列. 如果未传递轴标签, 则将根据常识规则从输入数据构造它们.

数据结构

- 当数据是dict columns且未指定时, DataFrame列将按dict的插入顺序排序(Python > = 3.6, Pandas > = 0.23), 或词汇顺序的dict键列表排序(Python < 3.6或Pandas < 0.23).
- 来自 Series 的 dict 或 dicts 得到的索引将是各个Series的索引的并集. 如果有任何嵌套的dict, 则首先将这些dicts转换为Series. 如果没有传递列, 则列将是dict键的有序列表.
  - 可以通过访问index和columns属性分别访问行标签和列标签.
- 来自 ndarrays / lists的字典. ndarrays必须都是相同的长度. 如果传递索引, 则它必须明显与数组的长度相同. 如果没有传递索引, 结果将是range(n).
- 来自结构化或数组记录. 这种情况的处理方式与数组的字典相同. DataFrame并不像二维NumPy ndarray那样工作.
- 来自dicts列表.
- 来自元组的词典. 可以通过传递元组字典自动创建MultiIndexed帧.
- 来自 Series.

数据结构

- 列选择, 添加, 删除, 插入
  - 可以在语义上将DataFrame视为类似索引的Series对象的dict. 获取, 设置和删除列的工作方式与类似的dict操作相同.
  - 列可以像dict一样删除或弹出.
  - 插入标量值时, 它会自然地传播以填充列.
  - 插入与DataFrame不具有相同索引的Series时, 它将符合DataFrame的索引.
  - 可以插入原始ndarrays, 但它们的长度必须与DataFrame索引的长度相匹配. 默认情况下, 列会在末尾插入. insert函数可用于插入列中的特定位置.

数据结构

- 对象创建
  - 通过传递带有日期时间索引和带标签列的NumPy数组来创建DataFrame.
  - 通过传递可以转化为类似Series的dict对象来创建DataFrame.
  - DataFrame的列具有不同的数据类型.
- 查看数据
  - 顶部和尾部的数据
  - 显示索引, 列和底层NumPy数据
  - describe() 方法显示数据的快速统计摘要
  - 转置数据
  - 按轴排序
  - 按值排序

数据结构

- 获取
  - 选择一个列, 产生一个 Series
  - 通过[ ]选择, 对行进行切片
  - 按标签选择
    - 通过标签获取一行数据
    - 通过标签在多个轴上选择数据
    - 通过标签同时在两个轴上切片
  - 布尔索引
    - 使用单个列的值来选择数据
    - 从满足布尔条件的DataFrame中选择值
    - 使用 isin() 方法过滤

数据结构

- 赋值
  - 添加新列将自动根据索引对齐数据
  - 通过标签赋值
  - 通过位置赋值
  - 使用NumPy数组赋值

THE END