Data Analysis

Velen Kong

摘要

基于Python的数据分析入门笔记，希望能自学掌握基本的数据分析能力，毕竟是统计学的基础之一。

内容安排主要参考

(美) Wes McKinney 著. Python for Data Analysis:2nd Edition [M] USA: O'Reilly Media 2017

该书作者同时也是pandas库的作者，同时借用其在GitHub上的数据资料。

第三方库与Python入门

IPython & Jupyter

用到的库：ipython，jupyter，numpy，matplotlib，pandas，scipy，scikit-learn，statsmodels。

jupyter确实好用，扩展了$Tab, ?, *$的功能。

还有一些快捷键仅限IPython使用。

另外就是一些常用的Magic Command，比如

```
%run
%load
%debug
%xmode Plain # less detailed
%xmode Verbose # more detailed
%matplotlib inline # draw in Jupyter
import numpy as np
import matplotlib.pyplot as plt
plt.plot(np.random.rand(50).cumsum())
```

```
# ----- Day 0 -----
```

Python基础

#注释。

```
# this is comment
```

所有变量都是对象(object)。

基本的函数调用，利用函数批量处理。

```
def append_element(some_list, element):
    some_list.append(element)
    return

data = [1, 2, 3]
append_element(data, 4)
data
# Out: [1, 2, 3, 4]
```

变量赋值类似引用(reference)。

动态类型，利用type()和isinstance()进行类型检查。后者可以传入tuple代替逻辑或操作。

```
a = 1.5
isinstance(a, (int, float))
# Out: True
```

print配合format输出。

```
a = 1; b = 1.5
print('a is {1}, b is {0}'.format(type(b),
type(a)))
# Out: a is <class 'int'>, b is <class 'float'>
```

Python中的object拥有各自的属性(attributes)和方法(methods)，可通过getattr, hasattr, setattr操作。其中getattr可以直接使用返回对象，setattr不改变原class。

```
class A(object):
    def set(self, a, b):
        x = a
        a = b
        b = x
        print a, b

a = A()
c = getattr(a, 'set')
c(a='1', b='2')
# Out: 2 1
```

可以利用try()检查容器是否是顺序，同时生成list。

```
def isiterable(obj):
    try:
        iter(obj)
        return True
    except TypeError:
```

```
        return False
    return

x = 'string'
isiterable(x)
# Out: True
isiterable(5)
# Out: False

if not isinstance(x, list) and isiterable(x):
    x = list(x)
x
# Out: ['s', 't', 'r', 'i', 'n', 'g']
```

import和as的使用。

is和is not的使用。

```
a = [1, 2, 3]
b = a
c = list(a)
d = None
print(a is b)
# Out: True
print(a is not c)
# Out: True
print(a == c)
# Out: True
print(d is None)
# Out: True
```

大部分Python的容器都是可变的(mutable)，而strings和tuples不可变(immutable)。

```
a = ['foo', 2, [4, 5]]
a[2] = (3, 4)
a
# Out: ['foo', 2, (3, 4)]
```

```
# ----- Day 1 -----
```

标量类型(scalar types)一般有$None, str, bytes, float, bool, int$。

多行字符串用三个引号。

```
c = """
This is a longer string that
spans multiple lines
"""
c.count('\n')
# Out: 3
```

str()方法可以生成字符串，+可以直接拼接字符串。而replace()方法可以替换字符串内容，但不改变原串。

```
a = 'this is a string'
b = a.replace('string', 'longer string')
print(b)
# Out: this is a longer string
print(a)
# Out: this is a string
```

Python采用$Unicode$编码，支持中文。对特殊字符可以采用\或者r进行转义。

```
s = '12\\34'
print(s)
# Out: 12\34
s = r'this\has\no\special\characters'
s
# Out: 'this\\has\\no\\special\\characters'
```

format和{}可以进行格式化输出。

```
template = '{0:.2f} {1:s} are worth US${2:d}'
template.format(4.5560, 'Argentine Pesos', 1)
# Out: '4.56 Argentine Pesos are worth US$1'
```

利用encode()和decode()函数可以对字符进行加解码。利用b生成二进制串。

```
val = "español"
val_utf8 = val.encode('utf-8')
val_utf8
# Out: b'espa\xc3\xb1ol'
type(val_utf8)
# Out: bytes
val_utf8.decode('utf-8')
# Out: 'español'
bytes_val = b'this is bytes'
bytes_val
# Out: b'this is bytes'
decoded = bytes_val.decode('utf8')
decoded
# Out: 'this is bytes'
```

None是一个单独的类型，常见于函数的默认参数中。

```
def add_and_maybe_multiply(a, b, c=None):
    result = a + b
    if c is not None:
        result = result * c
    return result

type(None)
# Out: NoneType
```

and和or运算。注意Python没有位运算操作符，&, |, ~都是逻辑运算。

Python自带时间与日期库。datetime是不可变类型，所以所有的操作都会产生新的对象而不改变原对象。

```
from datetime import datetime, date, time
dt = datetime(2011, 10, 29, 20, 30, 21)
dt.day
# Out: 29
dt.date()
# Out: datetime.date(2011, 10, 29)
dt.strftime('%m/%d/%Y %H:%M')
# Out: '10/29/2011 20:30'
datetime.strptime('20091031', '%Y%m%d')
# Out: datetime.datetime(2009, 10, 31, 0, 0)
dt.replace(minute=0, second=0)
# Out: datetime.datetime(2011, 10, 29, 20, 0)
```

时间差用timedelta表示，单位是天和秒。

```
dt2 = datetime(2011, 11, 15, 22, 30)
delta = dt2 - dt
delta
# Out: datetime.timedelta(17, 7179)
type(delta)
# Out: datetime.timedelta
dt + delta
# Out: datetime.datetime(2011, 11, 15, 22, 30)
```

时间的格式符号(format specification)如下

```
%Y #4位年数
%y #2位年数
%m #2位月数
%d #2位天数
%H #24小时制(2位)
%I #12小时制(2位)
%M #2位分钟数
%S #2位秒数，由于闰秒存在，范围为[00, 61]
```

```
%w #星期，范围为[0, 6]
%U #一年的第几个星期，以周日为标准，第一个周日之前的算00
%W #同上，以周一为标准
%z #用+HHMM或-HHMM表示相对于格林威治的时区偏移
%Z #时区名称，默认为空
%F #等价于%Y-%m-%d
%D #等价于%m/%d/%y
```

if, elif, else语句。单行if。

```
a = [1,2,3]
b = a if len(a) != 0 else ""
print(b)
# Out: [1, 2, 3]
```

for in语句。多变量for循环。

```
starts = [0,1,2,3,4]
ends = [5,6,7,8,9]
for start, end in zip(starts, ends):
    print((start, end))
'''
Out:
(0, 5)
(1, 6)
(2, 7)
(3, 8)
(4, 9)
'''
```

while语句。

range()函数。

```
# ----- Day 2 -----
```

数据结构

Tuple是Python的基本数据结构之一，可用逗号分隔直接创建，或者分解顺序容器，也可以通过range构造序列。

```
tup = 3, (4, 5, 6), (7, 8)
tup
# Out: (3, (4, 5, 6), (7, 8))
tuple([4, 0, 2])
# Out: (4, 0, 2)
tuple('string')
# Out: ('s', 't', 'r', 'i', 'n', 'g')
tuple(['foo', [1, 2], True])
# Out: ('foo', [1, 2], True)
tuple(range(10))
# Out: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

tuple也可以直接进行+, *运算，和解压操作。

```
(4, None, 'foo') + (6, 0) + ('bar',)
# Out: (4, None, 'foo', 6, 0, 'bar')
('foo', 'bar') * 4
# Out: ('foo', 'bar', 'foo', 'bar', 'foo', 'bar',
'foo', 'bar')
tup = 4, 5, (6, 7)
a, b, (c, d) = tup
d
# Out: 7
a, b = 1, 2
b, a = a, b
print(a,b)
# Out: 2 1
```

也可以配合多变量for循环。

```
seq = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
for a, b, c in seq:
    print('a={0}, b={1}, c={2}'.format(a, b, c))
'''
Out:
a=1, b=2, c=3
a=4, b=5, c=6
a=7, b=8, c=9
'''
```

用*可以代指tuple的剩余部分。

```
values = 1, 2, 3, 4, 5
a, b, *rest = values
rest
# Out: [3, 4, 5]
```

count是tuple的常用方法。

```
a = (1, 2, 2, 2, 3, 4, 2)
a.count(2)
# Out: 4
```

List近似于可变的tuple，定义方法和tuple类似，也可以进行+,*操作。

利用append，insert，extend，pop，remove可以对list进行修改。但extend比+速度快。

```
tup = ('foo', 'bar', 'baz')
b_list = list(tup)
b_list
# Out: ['foo', 'bar', 'baz']
b_list.append('dwarf')
b_list
# Out: ['foo', 'bar', 'baz', 'dwarf']
b_list.insert(1, 'red')
```

```
b_list
# Out: ['foo', 'red', 'bar', 'baz', 'dwarf']
b_list.extend([7, 8, (2, 3)])
b_list
# Out: ['foo', 'red', 'bar', 'baz', 'dwarf', 7, 8, (2, 3)]
b_list.pop(2)
b_list
# Out: ['foo', 'red', 'baz', 'dwarf', 7, 8, (2, 3)]
b_list.append('foo')
b_list.remove('foo')
b_list
# Out: ['red', 'baz', 'dwarf', 7, 8, (2, 3), 'foo']
```

sort可以对一个list排序，bisect库可以进行二分操作。但bisect需要预先排序。sorted可以生成排序后的list。还可以设置排序的key。

```
import bisect
a = [7, 2, 2, 5, 1, 3]
a.sort()
a
# Out: [1, 2, 2, 3, 5, 7]
b = ['saw', 'small', 'He', 'foxes', 'six']
b.sort(key=len)
b
# Out: ['He', 'saw', 'six', 'small', 'foxes']
import bisect
bisect.bisect(a, 2)
# Out: 3
bisect.bisect(a, 6)
# Out: 5
bisect.insort(a, 6)
a
# Out: [1, 2, 2, 3, 5, 6, 7]
```

```
sorted('horse race')
# Out: [' ', 'a', 'c', 'e', 'e', 'h', 'o', 'r', 'r', 's']
sorted('horse race', key = lambda x : -ord(x))
# Out: ['s', 'r', 'r', 'o', 'h', 'e', 'e', 'c', 'a', ' ']
```

利用[begin: end: step]可以轻松实现部分list的选取和修改，其中step也可以为负数。reverse能反转整个容器。

```
list(reversed(range(10)))
# Out: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

enumerate和zip可以用于生成变量组合。

```
some_list = ['foo', 'bar', 'baz']
mapping = {}
for i, v in enumerate(some_list):
    mapping[v] = i
mapping
# Out: {'foo': 0, 'bar': 1, 'baz': 2}
seq1 = ['foo', 'bar', 'baz']
seq2 = ['one', 'two', 'three']
zipped = zip(seq1, seq2)
list(zipped)
# Out: [('foo', 'one'), ('bar', 'two'), ('baz', 'three')]
seq3 = [False, True]
list(zip(seq1, seq2, seq3))
# Out: [('foo', 'one', False), ('bar', 'two', True)]
for i, (a, b) in enumerate(zip(seq1, seq2)):
    print('{0}: {1}, {2}'.format(i, a, b))
'''
Out:
0: foo, one
1: bar, two
```

```
2: baz, three
'''
```

zip也可以反过来进行解压。

```
pitchers = [('Nolan', 'Ryan'), ('Roger',
'Clemens'),
            ('Schilling', 'Curt')]
first_names, last_names = zip(*pitchers)
first_names
# Out: ('Nolan', 'Roger', 'Schilling')
```

```
# ----- Day 3 -----
```

字典(dict)是Python最重要的数据结构之一，存储键值对(key-value pair)，基本操作如下。

```
d1 = {'a' : 'some value', 'b' : [1, 2, 3, 4]}
d1
# Out: {'a': 'some value', 'b': [1, 2, 3, 4]}
d1['b']
# Out: [1, 2, 3, 4]
d1['dummy'] = 'another value'
d1
# Out: {'a': 'some value', 'b': [1, 2, 3, 4],
'dummy': 'another value'}
del d1['a']
d1
# Out: {'b': [1, 2, 3, 4], 'dummy': 'another
value'}
ret = d1.pop('dummy')
ret
# Out: 'another value'
d1
# Out: {'b': [1, 2, 3, 4]}
'b' in d1
# Out: True
```

```
list(d1.keys())
# Out: ['b']
list(d1.values())
# Out: [[1, 2, 3, 4]]
d1['b'] = 1
d1
# Out: {'b': 1}
d1.update({'b': 2, 2: ['a', 'b']})
d1
# Out: {'b': 2, 2: ['a', 'b']}
```

dict()可以直接生成dict。

```
map1 = dict(zip(range(5), range(5, -5, -1)))
map1
# Out: {0: 5, 1: 4, 2: 3, 3: 2, 4: 1}
map2 = dict(zip(range(10), range(5, 0, -1)))
map2
# Out: {0: 5, 1: 4, 2: 3, 3: 2, 4: 1}
```

读取dict值时还可以利用get设置默认值。如果不使用get，则会发生KeyError，比如pop()和del。所以使用dict前务必用in检查key是否存在。

```
# value = some_dict.get(key, default_value)
```

在使用list作为值时，要注意将default设为空list。可以使用setdefault方法或者defaultdict类型。

```
words = ['apple', 'bat', 'bar', 'atom', 'book']
by_letter = {}
for word in words:
    letter = word[0]
    by_letter.setdefault(letter, []).append(word)
by_letter
# Out: {'a': ['apple', 'atom'], 'b': ['bat',
'bar', 'book']}
```

```
from collections import defaultdict
by_letter = defaultdict(list)
by_letter
# Out: defaultdict(list, {})
for word in words:
    by_letter[word[0]].append(word)
by_letter
# Out: defaultdict(list, {'a': ['apple', 'atom'],
'b': ['bat', 'bar', 'book']})
```

dict利用了hash()，所以key必须是可以被hash的对象。例如list无法作为key使用，需要转化为tuple。因为在list变化后，hash值也被改变，因而dict中保存的list将找不到对应的value产生错误。

set是无序无重复的容器。利用set()或者{}来生成set。

set可以利用union(), intersection(), difference(), symmetrice_difference()来进行操作，也可以使用|, &, -, ^。

常见方法有:

add(), clear(), remove(), pop(), copy(), len();

union(), update(), intersection(), intersection_update(), difference(), difference_update(), symmetrice_difference(), symmetrice_difference_update();

issubset(), issuperset(), isdisjoint()。

和dict类似，remove()和pop()需要先用in检查，否则会抛出KeyError。

同样，set也利用了hash，所以也不能加入list。

for in if语句常被用于list，dict，set中。可以轻松进行条件筛选。

```
strings = ['a', 'as', 'bat', 'car', 'dove',
'python']
[x.upper() for x in strings if len(x) > 2]
```

```
# Out: ['BAT', 'CAR', 'DOVE', 'PYTHON']
unique_lengths = {len(x) for x in strings}
unique_lengths
# Out: {1, 2, 3, 4, 6}
loc_mapping = {val : index for index, val in enumerate(strings)}
loc_mapping
# Out: {'a': 0, 'as': 1, 'bat': 2, 'car': 3, 'dove': 4, 'python': 5}
all_data = [['John', 'Emily', 'Michael', 'Mary', 'Steven'],
            ['Maria', 'Juan', 'Javier', 'Natalia', 'Pilar']]
result = [name for names in all_data for name in names
          if name.count('e') >= 1]
result
# Out: ['Michael', 'Steven', 'Javier']
```

也可以使用map函数，对顺序容器做一个映射。

```
set(map(len, strings))
# Out: {1, 2, 3, 4, 6}
list(map(lambda k: k ** 2, [1, 2, 3, 4, 5]))
# Out: [1, 4, 9, 16, 25]
dict(map(lambda x, y: (x, y), [1, 3, 5, 7, 9], [2, 4, 6, 8, 10]))
# Out: {1: 2, 3: 4, 5: 6, 7: 8, 9: 10}
```

多重for in语句。

```
some_tuples = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
[[x for x in tup] for tup in some_tuples]
# Out: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
# ----- Day 4 -----
```

函数

函数定义时可以设置默认参数。调用时可以无视默认参数。也可以乱序设定参数。

定义在函数内部的本地变量会在return时被销毁。但可以使用global来定义。但不建议过多使用全局变量，那说明需要使用class和面向对象编程(object-oriented programming)。

Python可以返回多个参数。

```
def f():
    a = 5
    b = 6
    c = 7
    return a, b, c

c = 1
a, b, d = f()
print(a, b, c)
# Out: 5 6 1
```

Python中函数也是一种对象。由此可以将不同函数进行组合。

```
import re
states = ['   Alabama ', 'Georgia!', 'Georgia',
'georgia',
          'FlOrIda', 'south   carolina##', 'West
virginia?']

def remove_punctuation(value):
    return re.sub('[!#?]', '', value)

clean_ops = [str.strip, remove_punctuation,
str.title]

def clean_strings(strings, ops):
```

```
    result = []
    for value in strings:
        for function in ops:
            value = function(value)
        result.append(value)
    return result

clean_strings(states, clean_ops)
'''
Out:
['Alabama',
 'Georgia',
 'Georgia',
 'Georgia',
 'Florida',
 'South   Carolina',
 'West Virginia']
'''
```

在数据分析中，还会经常使用匿名函数(anonymous function)。其本身在创建时没有__name__属性。

```
anon = lambda x: x * 2
anon(4)
# Out: 8
def apply_to_list(some_list, f):
    return [f(x) for x in some_list]

ints = [4, 0, 1, 5, 6]
apply_to_list(ints, lambda x: x * 2)
# Out: [8, 0, 2, 10, 12]
strings = ['foo', 'card', 'bar', 'aaaa', 'abab']
strings.sort(key=lambda x: len(set(list(x))))
strings
# Out: ['aaaa', 'foo', 'abab', 'bar', 'card']
```

偏函数用于对函数进行局部套用，可以理解为封装。

```python
# functools.partial(func, *args, **keywords)
def add(*args, **kwargs):
    for n in args:
        print(n)
    print("-"*10)
    for k, v in kwargs.items():
       print('%s=%s' % (k, v))

add(1, 2, 3, k1=10, k2=20)
'''
Out:
1
2
3
----------
k1=10
k2=20
'''
from functools import partial
add_partial = partial(add, 10, k1=10, k2=20)
add_partial(1, 2, 3, k3=20)
'''
Out:
10
1
2
3
----------
k1=10
k2=20
k3=20
'''
add_partial(1, k1=0)
'''
Out:
10
1
```

```
----------
k1=0
k2=20
'''
```

Python的生成器(generator)在数据科学领域有着广泛的应用，占用计算机资源较小。生成器也可以进行迭代，但与普通的顺序容器不同的是，它不会把所有数据存入内存中，而是只能被读取一次，实时生成数据。

生成器可以通过()构建，也可以使用关键字yield。每次生成器返回一个值时就会被冻结，直到下次调用再返回一个值。

```
gen = (x for x in range(5))
print(gen)
# Out: <generator object <genexpr> at 0x0000023E97215830>
for item in gen:
    print(item)
'''
Out:
0
1
2
3
4
'''
def createGenerator() :
    mylist = range(3)
    for i in mylist :
        yield i*i

mygenerator = createGenerator()
print(mygenerator)
# Out: <generator object createGenerator at 0x0000023E97215938>
for res in mygenerator:
```

```
    print(res)
'''
Out:
0
1
4
'''
mygenerator = createGenerator()
print(next(mygenerator))
# Out: 0
```

生成器可以用来代替列表推导式(list comprehension)。

```
dict((x, x ** 2) for x in range(5))
# Out: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

标准库itertools提供了许多常用生成器。

```
import itertools as it
list(it.combinations(range(3), 2))
# Out: [(0, 1), (0, 2), (1, 2)]
list(it.permutations(range(3), 2))
# Out: [(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)]
import itertools
first_letter = lambda x: x[0]
allnames = ['Alan', 'Adam', 'Wes', 'Will', 'Albert', 'Steven']
for letter, names in itertools.groupby(allnames, first_letter):
    print(letter, list(names)) # names is a generator
'''
Out:
A ['Alan', 'Adam']
W ['Wes', 'Will']
A ['Albert']
```

```
S ['Steven']
'''
list(it.product(range(2), ['a', 'b'], repeat = 1))
# Out: [(0, 'a'), (0, 'b'), (1, 'a'), (1, 'b')]
```

```
# ----- Day 5 -----
```

异常处理

可以通过try expect语句进行异常抛出。同时可以设置异常类型，还能使用as和Exception。

```
def attempt_float(x):
    try:
        return float(x)
    except (ValueError, TypeError) as e:
        print(e)
        return x
    except Exception:
        return x
attempt_float('1.234')
# Out: 1.234
attempt_float('1')
# Out: 1.0
attempt_float('1.a')
'''
Out:
could not convert string to float: '1.a'
'1.a'
'''
```

同时还可以使用else和finally。

```
try:
    f = open(path, 'w')
except:
    print('Not Found')
else:
    try:
        write_to_file(f)
    except:
        print('Failed')
    else:
        print('Succeeded')
    finally:
        f.close()
```

还可以使用raise触发异常。

```
# raise [Exception [, args [, traceback]]]
def ThorwErr():
    raise Exception("抛出一个异常")
ThorwErr()
```

捕捉到了异常，但是又想重新引发它(传递异常)，可以使用不带参数的raise语句。

```
class MuffledCalculator:
    muffled = False
    def calc(self, expr):
        try:
            return eval(expr)
        except ZeroDivisionError:
            if self.muffled:
                print('Division by zero is
illegal')
            else:
                raise
a = MuffledCalculator()
# a.muffled = True
a.calc('2/0')
```

也可以自定义异常类型，对Exception类进行继承即可。

```
class ShortInputException(Exception):
    def __init__(self, length, atleast):
        self.length = length
        self.atleast = atleast
try:
    s = input()
    if len(s) < 3:
        raise ShortInputException(len(s), 3)
except ShortInputException as e:
    print('输入长度是%s,长度至少是%s' %(e.length,
e.atleast))
else:
    print(s)
# In: ab
# Out: 输入长度是2,长度至少是3
```

文件系统

利用open()和close()打开关闭文件，可以设置编码方式，r只读，w只写，x新建只写，a添加，r+读写，b二进制模式，t文本模式。打开后tell()表示当前位置，不同编码tell()效果不同。

```
path = 'segismundo.txt'
f1 = open(path, 'rb')
lines = (x.rstrip() for x in open(path))
a = f1.read(10)
a.decode(encoding = 'utf-8')
# Out: 'Sueña el '
f2 = open(path)
lines = (x.rstrip() for x in open(path))
a = f2.read(10)
a.encode(encoding = 'gbk').decode('utf-8')
# Out: 'Sueña el r'
print(f1.tell(), f2.tell())
# Out: 10 11
f1.close()
f2.close()
```

也可以用with open读取文件。

```
with open('segismundo.txt', 'r', encoding='gbk')
as f:
    lines =
[x.rstrip().encode('cp936').decode('utf-8')\
             for x in open(path)]
lines[0]
# Out: 'Sueña el rico en su riqueza,'
```

文件读取也与系统默认编码有关，可以通过sys和locale库查看系统默认编码和Python默认编码。

```
import sys
sys.getdefaultencoding()
# Out: 'utf-8'
import locale
locale.getpreferredencoding()
# Out: 'cp936'
# cp936 = gbk
```

seek()和read()分别用来移动到指定位置和读取字符。慎用seek()，因为seek()转移到某个多字节字符中间时进行read()会发生UnicodeDecodeError。

```
f = open(path, 'r')
f.seek(3)
# Out: 3
f.read(1).encode('gbk').decode('utf-8')
# Out: 'ñ'
```

文件的常用方法还有readlines(), write(), writelines(), flush(), closed。

```
# ----- Day 6 -----
```

NumPy库

N维数组

numpy库全称Numerical Python，专用于科学计算，由于底层C语言的实现，速度比纯python快几十倍。rand()返回[0, 1)上的均匀分布，randn()返回标准正态分布。

```
import numpy as np
data = np.random.randn(2, 3)
data
'''
Out:
array([[-0.97835473,  0.41601724, -0.75555622],
       [ 1.90504659,  0.22870654, -0.45414597]])
'''
```

可以直接对数组进行数学运算。shape()和dtype()可以查看数组信息。

```
data * 10
'''
Out:
array([[-9.7835,  4.1602, -7.5556],
       [19.0505,  2.2871, -4.5415]])
'''
data + data
'''
Out:
array([[-1.9567,  0.832 , -1.5111],
       [ 3.8101,  0.4574, -0.9083]])
'''
data.shape
# Out: (2, 3)
data.dtype
# Out: dtype('float64')
```

array()方法可以将传入参数转化为数组，而如果参数是数组那么会产生一个副本，而使用asarray()会直接返回传入的数组。

用ndim返回第一维的维数。

```
data1 = [[1, 2, 3], [2, 4, 6]]
arr1 = np.array(data1)
arr1.ndim
# Out: 2
```

zeros(), ones(), zeros_like(), ones_like(), full(), full_like()用来创建新数组。

最好不要使用没有初始化的empty()。

arange()创建一个一维顺序数组。

eye()和identity()用来创建单位矩阵。

默认数据类型均为float64。

通过设定dtype参数改变数据类型。

```
arr2 = np.identity(3)
arr2
'''
Out:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
'''
arr3 = np.full_like(arr2, 3, dtype=np.int32)
arr3
'''
Out:
array([[3, 3, 3],
       [3, 3, 3],
       [3, 3, 3]])
'''
```

numpy支持的数据类型有：

int8, uint8, int16, uint16, int64, uint64;

float16, float32, float64, float128;

complex64, complex128, complex256;

bool, object, string_, unicode_。

缩写为i1, u1, i2, u2, i4, u4, i8, u8;

f2, f4 or f, f8 or d, f16 or g;

c8, c16, c21;

?, 0, S, U。

对于非数学类型最好使用pandas。

astype()可以进行类型转换，调用时总会创建一个新的array。

```
arr4 = arr3.astype('f8')
arr4
'''
Out:
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
'''
```

除了array相互加减乘之外，numpy还支持常数乘除法和指数运算。

对size相同的array使用不等号还可以返回一个bool类型的数组。

```
arr5 = np.random.randn(2, 3)
arr6 = 1 / arr5
arr6 < arr5
'''
Out:
array([[False, False, False],
       [ True,  True, False]])
'''
```

和list类似，array可以使用[begin1: end1, begin2: end2, ...]来获得子数组。注意获得子数组并未新建，利用copy()来获得副本。

```
arr0 = np.array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])
arr7 = arr0[0]
arr7
# Out: array([1, 2, 3])
arr7[:] = 0
arr0
'''
Out:
array([[0, 0, 0],
       [4, 5, 6],
       [7, 8, 9]])
'''
arr8 = arr0[1:3, :2].copy()
arr8
'''
Out:
array([[4, 5],
       [7, 8]])
'''
```

```
# ----- Day 7 -----
```

除了基本索引之外，还可以使用布尔索引，但要保证维数相同，Python并不会报错。也可以两者结合使用。

```
import numpy as np
names = np.array(['A', 'B', 'A', 'C'])
data = np.eye(4, dtype='i4')
names == 'A'
# Out: array([ True, False,  True, False])
data[names == 'A']
'''
Out:
array([[1, 0, 0, 0],
       [0, 0, 1, 0]])
'''
data[names == 'A', :2]
'''
Out:
array([[1, 0],
       [0, 0]])
'''
data[~((names == 'A') | (names == 'C')), 2:]
# Out: array([[1, 0, 0]])
```

也可以利用逻辑表达式来改变array的内容。

```
data[data > 0] = -1
data
'''
Out:
array([[-1,  0,  0,  0],
       [ 0, -1,  0,  0],
       [ 0,  0, -1,  0],
       [ 0,  0,  0, -1]])
'''
data[names != 'A'] = 1
data
'''
```

```
Out:
array([[-1,  0,  0,  0],
       [ 1,  1,  1,  1],
       [ 0,  0, -1,  0],
       [ 1,  1,  1,  1]])
'''
```

花式索引(fancy index)是一种智能索引。通过传递索引数组来一次获得多个元素。

```
arr = np.zeros((8, 4))
for i in range(8):
    arr[i] = i
arr[[4, 3, -1]]
'''
Out:
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [7., 7., 7., 7.]])
'''
arr = np.arange(20).reshape(5, 4) + 1
arr
'''
Out:
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20]])
'''
arr[[1, 2, -1], [3, 0, -1]]
# Out: array([ 8,  9, 20])
```

也可以多重嵌套fancy index。

```
arr[[1, 2]]
'''
Out:
array([[ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
'''
arr[[1, 2]][:, [1, 0]]
'''
Out:
array([[ 6,  5],
       [10,  9]])
'''
```

可用dot()来计算array的点积。T获得转置，也可以使用transpose()，高维建议使用后者，前者相当于整个array倒置。swapaxes()则能交换两个维度。

```
arr = np.arange(24).reshape((2, 4, 3)) +1
arr
'''
Out:
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9],
        [10, 11, 12]],

       [[13, 14, 15],
        [16, 17, 18],
        [19, 20, 21],
        [22, 23, 24]]])
'''
arr.T
'''
Out:
array([[[ 1, 13],
        [ 4, 16],
```

```
        [ 7, 19],
        [10, 22]],

       [[ 2, 14],
        [ 5, 17],
        [ 8, 20],
        [11, 23]],

       [[ 3, 15],
        [ 6, 18],
        [ 9, 21],
        [12, 24]]])
'''
arr.transpose((1, 0, 2))
'''
Out:
array([[[ 1,  2,  3],
        [13, 14, 15]],

       [[ 4,  5,  6],
        [16, 17, 18]],

       [[ 7,  8,  9],
        [19, 20, 21]],

       [[10, 11, 12],
        [22, 23, 24]]])
'''
arr.swapaxes(0, 1)
'''
Out:
array([[[ 1,  2,  3],
        [13, 14, 15]],

       [[ 4,  5,  6],
        [16, 17, 18]],

```

```
       [[ 7,  8,  9],
        [19, 20, 21]],

       [[10, 11, 12],
        [22, 23, 24]]])
'''
```

```
# ----- Day 8 -----
```

通用函数

array有许多通用函数都可以使用，比如sqrt, exp等。常用的函数有：

abs, fabs;

sqrt, square, exp, expm1, log, log10, log2, log1p;

其中expm1相当于exp(x)-1，与log1p相当于ln(x+1)互逆，常用来数据预处理，比如求RMSLE(均方根对数误差)。

sign, ceil, floor, rint, modf;

分别是符号函数，和四个取整函数(返回float)，其中modf会返回两个数组。

```
arr = np.random.randn(3) * 5
arr
# Out: array([1.40907511, 4.3517101 ,
2.37764427])
remainded, whole_part = np.modf(arr)
remainded
# Out: array([0.40907511, 0.3517101 ,
0.37764427])
whole_part
# Out: array([1., 4., 2.])
np.ceil(whole_part)
whole_part
# Out: array([1., 4., 2.])
```

isnan, isfinite, isinf;

注意nan不能使用==来判断，而且会被同时当作极大和极小。

```
np.nan == np.nan
# Out: False
np.nan is np.nan
# Out: True
import math
math.nan is np.nan
# Out: False
np.isnan(math.nan)
# Out: True
```

cos, cosh, sin, sinh, tan, tanh, arccos, arccosh, arcsin, arcsinh, arctan, arctanh;

logical_not(等价于~array);

还有一些带有两个参数的通用函数：

add(), subtract(), multiply(), divide(), floor_divide(), power();

这些函数可以接受单个值也可以接受array。

```
arrd = np.divide(arr, 2)
arrd
# Out: array([0.70453755, 2.17585505,
1.18882213])
arrd = np.divide(arr, whole_part)
arrd
# Out: array([1.40907511, 1.08792752,
1.18882213])
arrp = np.power(arr, 2)
arrp
# Out: array([ 1.98549266, 18.93738078,
5.65319227])
arrp = np.power(arr, remainded)
arrp
# Out: array([1.15060233, 1.67734792,
1.38691458])
```

maximum, fmax, minimum, fmin;

其中fmax和fmin无视NaN。

```
arr0 = np.array([np.nan, 1, 3, 1])
arr1 = np.array([2, 1, 1, 1])
print(np.maximum(arr0, arr1))
# Out: [nan  1.  3.  1.]
print(np.fmax(arr0, arr1))
# Out: [2. 1. 3. 1.]
```

copysign();

greater(), greater_equal(), less(), less_equal(), equal(), not_equal();

等价于不等号判断，返回boolean数组。

logical_and(), logical_or(), logical_xor();

等价于&, |, ^。

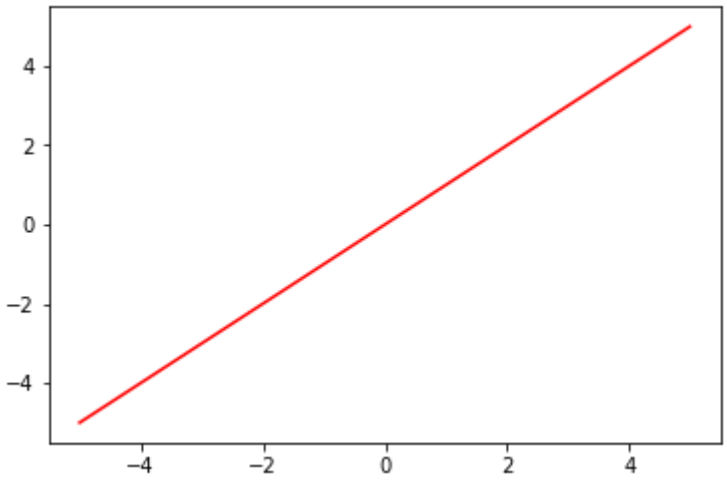
```
# ----- Day 9 -----
```

面向数组编程

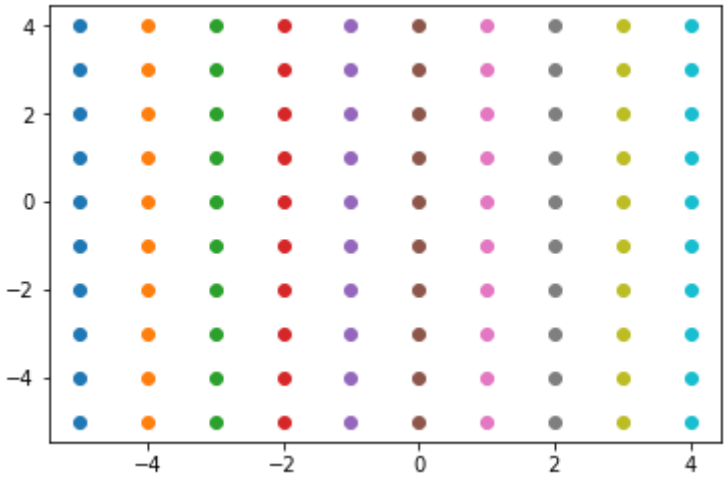
NumPy提供了许多面向数组的函数和方法，能更快速地对array进行操作。

比如当我们需要网格点时，可以利用meshgrid()对x, y坐标数组做笛卡儿积。

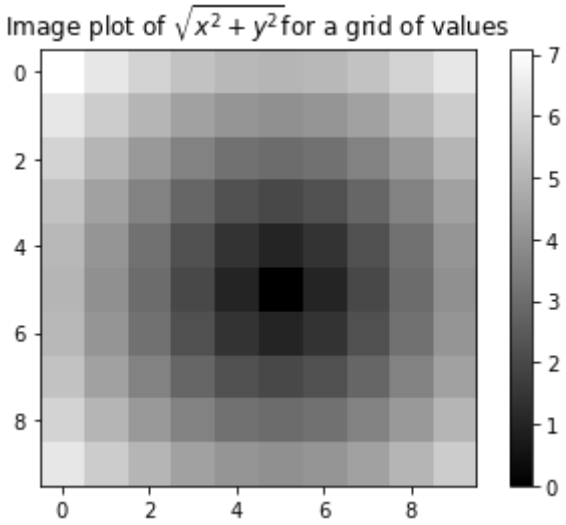
```
points = np.arange(-5, 5, 1)
plt.plot(points, points, 'r')
plt.show()
```




```
xs, ys = np.meshgrid(points, points)
plt.plot(xs, ys, 'o')
plt.show()
```




```
z = np.sqrt(xs**2 + ys**2)
plt.imshow(z, cmap=plt.cm.gray)
plt.colorbar()
plt.title('Image plot of $\sqrt{x^2 + y^2}$\
for a grid of values')
plt.show()
```



where()是一个三元函数，用来进行类似if else的操作，需要传入一个boolean数组和两个其他数组。也可以传入值。

```
xarr = [1, 2, 3]
yarr = [-1, -2, -3]
cond = [True, False, True]
result = np.where(cond, xarr, yarr)
result
# Out: array([ 1, -2,  3])
arr = np.random.randn(3, 3)
```

```
np.where(arr > 0, 1, arr)
'''
Out:
array([[ 1.        ,  1.        , -0.49359797],
       [ 1.        , -1.88837102, -1.03168442],
       [-0.04999441, -1.3569411 ,  1.        ]])
'''
```

NumPy还自带了很多数学函数。比如mean()和sum()，可以使用axis来控制计算的维度，axis=0即表示将除了0号维度之外全部相同的元素进行计算，从而消去0号维度。

```
arr.mean()
# Out: -0.04499350890122711
np.sum(arr)
# Out: -0.404941580111044
arr.sum(axis=0)
# Out: array([ 2.13961385, -1.55606646,
-0.98848896])
arr = np.ones((2, 3, 4))
arr.sum(axis=0)
'''
Out:
array([[2., 2., 2., 2.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.]])
'''
```

还有累加和累乘，cumsum(), cumprod()，也可以用axis控制维度。如果不控制纬度则会顺序遍历返回一个一维数组。

```
arr.cumsum(axis = 1)
'''
Out:
array([[[1., 1., 1., 1.],
        [2., 2., 2., 2.],
        [3., 3., 3., 3.]],

       [[1., 1., 1., 1.],
        [2., 2., 2., 2.],
        [3., 3., 3., 3.]]])
'''
```

除此以外还有var(), std(), argmin(), argmax()等函数。其中后两者返回最值第一次出现的位置。

```
# ----- Day 10 -----
```

boolean数组也可以执行sum()，同时还有any(), all()来进行整体与或。这也能用在其他array中。

array还可以执行sort()，利用axis参数将给定维度坐标不同，其余坐标相同的元素排序。

```
arr = np.random.randn(3, 3, 3) *5
arr = np.ceil(arr)
arr
'''
Out:
array([[[ -3.,  -0.,  -0.],
        [  5.,   8.,  -3.],
        [  1.,   1.,   5.]],

       [[  5.,   4.,  -3.],
        [ -1.,   2.,   1.],
        [ -2.,  -6.,  -8.]],

       [[ -1.,   6.,   6.],
```

```
        [  3.,  -4.,   6.],
        [  3., -11.,  -0.]]])
'''
arr.sort(axis=1)
arr
'''
Out:
array([[[ -3.,  -0.,  -3.],
        [  1.,   1.,  -0.],
        [  5.,   8.,   5.]],

       [[ -2.,  -6.,  -8.],
        [ -1.,   2.,  -3.],
        [  5.,   4.,   1.]],

       [[ -1., -11.,  -0.],
        [  3.,  -4.,   6.],
        [  3.,   6.,   6.]]])
'''
```

利用sort()可以很方便地计算q分位数。

```
arr = np.random.randn(1000)
arr.sort()
arr[int(0.05 * len(arr))]
# Out: -1.5488900316485399
```

另外还有一些常用函数：

unique(x), intersect1d(x, y), union1d(x, y), in1d(x, y), setdiff1d(x, y), setxor1d(x, y)。

其中intersect1d等返回的都是一维值数组。in1d()返回一个一维的boolean数组。

```
arr1 = np.array([[3, 4, 5],
                 [4, 2, 3],
```

```
                    [5, 3, 5]])
arr2 = np.array([[1, 2, 5],
                    [3, 4, 3],
                    [5, 3, 5]])
np.unique(arr1)
# Out: array([0, 2, 3, 4, 5])
np.union1d(arr1, arr2)
# Out: array([0, 1, 2, 3, 4, 5])
np.in1d(arr1, arr2)
# Out: array([ True,  True,  True,  True,  True,
True,  True,  True, False])
np.setdiff1d(arr1, arr2)
# Out: array([0])
```

文件读写

NumPy拥有自己的文件读写函数。

通过save(), load(), savez(), savez_compressed()可以进行基本的读写。其中savez类似于dict，而compressed进行了压缩。

```
arr = np.arange(5)
np.save('some_array', arr)
np.load('some_array.npy')
np.savez('array_archive.npz', a=arr, b=arr)
arch = np.load('array_archive.npz')
arch['b']
# Out: array([0, 1, 2, 3, 4, 5])
np.savez_compressed('arrays_compressed.npz',
a=arr, b=arr)
```

```
# ----- Day 11 -----
```

线性代数

在NumPy中，*并不能进行矩阵乘法，而是单纯地将各个位置上的元素相乘。进行点乘需要使用dot()。

```
x = np.array([[2, 3],
              [0, 1]])
y = np.array([[0, 1],
              [1, 0]])
x * y
'''
Out:
array([[0, 3],
       [0, 0]])
'''
x.dot(y)
'''
Out:
array([[3, 2],
       [1, 0]])
'''
```

另外，在点乘中一个二维矩阵和一个一维数组相乘，后者会被转化成向量，并返回一个一维数组。这个操作也可以用运算符@实现。

```
x.dot(np.array([1, 2]))
# Out: array([8, 2])
x @ np.array([2, 1])
# Out: array([7, 1])
```

numpy.linalg拥有许多常用函数，比如inv(), qr()用来获得逆矩阵和QR分解。

```
from numpy.linalg import inv, qr
x_ = inv(x)
x_
```

```
'''
Out:
array([[ 0.5, -1.5],
       [ 0. ,  1. ]])
'''
mat = np.random.randn(3, 3)
q, r = qr(mat)
r
'''
Out:
array([[ 0.79869179,  1.18877944,  0.19382584],
       [ 0.        , -2.24303773,  0.08872422],
       [ 0.        ,  0.        ,  0.86608074]])
'''
```

trace()获得矩阵的迹，而diag()对于二维矩阵可以获取对角线元素，对于一维数组则会创建对应的对角阵。

```
x.trace()
# Out: 3
np.diag(x)
# Out: array([2, 1])
np.diag(np.diag(x))
'''
Out:
array([[2, 0],
       [0, 1]])
'''
```

linalg.det()用来计算array的行列式，但要求最后两个维度大小相同。

linalg.eig()用来计算矩阵的特征值和特征向量。

```
from numpy.linalg import eig
w, v = eig(np.diag((1, 2, 3)))
w
# Out: array([1., 2., 3.])
v
'''
Out:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
'''
```

linalg.pinv()用来计算Moore-Penrose广义逆矩阵。

linalg.svd()对矩阵进行奇异值分解，返回三个分解矩阵。

linalg.solve()解线性方程组。

linalg.lstsq()求最小二乘解，返回解向量，残差和，秩，奇异值。

```
x = np.array([[1, 2, 1],
              [1, 0, 0],
              [0, 4, 0]])
pinv(x)
'''
Out:
array([[ 2.06415723e-16,  1.00000000e+00, -2.86157572e-16],
       [ 1.18467981e-16, -1.07459905e-16,  2.50000000e-01],
       [ 1.00000000e+00, -1.00000000e+00, -5.00000000e-01]])
'''
svd(x)
'''
Out:
(array([[-0.4856289 , -0.70136375, -0.52177912],
        [-0.02497309, -0.58551396,  0.81027757],
```

```
        [-0.87380828,  0.40652464,
0.26682728]]),
 array([4.52173541, 1.48251813, 0.59669834]),
 array([[-0.11292169, -0.98778246, -0.10739879],
        [-0.86803506,  0.15067004, -0.4730895 ],
        [ 0.48349129,  0.03980385,
-0.87444373]]))
'''
solve(x, np.array([1, 1, 1]))
# Out: array([ 1.  ,  0.25, -0.5 ])
x = np.array([[1, 2],
              [1, 0],
              [0, 4]])
lstsq(x, np.array([1, 1, 1]), rcond=None)
'''
Out:
(array([0.77777778, 0.22222222]),
 array([0.11111111]),
 2,
 array([4.49661478, 1.33433712]))
'''
```

```
# ----- Day 12 -----
```

Random库

NumPy有许多的随机数生成器，比如可以用normal()来生成正态分布。

```
sample = np.random.normal(size=(3, 3))
```

通过normal()生成的一系列随机数速度很快，比逐个生成要快很多。

```
from random import normalvariate
N = 1000000
%timeit samples = [normalvariate(0, 1) for _ in range(N)]
%timeit np.random.normal(size=N)
'''
Out:
882 ms ± 24.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
38.1 ms ± 449 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
'''
```

另外可以通过seed()改变随机种子，RandomState()来创建局部随机生成器。

```
np.random.seed(123)
sample = np.random.RandomState(123)
sample.randn(5)
# Out: array([-1.0856306 ,  0.99734545,  0.2829785 , -1.50629471, -0.57860025])
```

permutation()和shuffle()都可以用来打乱array，多维数组则只打乱第一维。

但前者还能传入int，相当于随机打乱arange()。而且前者返回一个打乱后的副本，后者会直接改变array并且返回None。

```
sample = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
np.random.permutation(sample)
'''
Out:
array([[4, 5, 6],
       [7, 8, 9],
       [1, 2, 3]])
```

```
'''
sample
'''
Out:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
'''
np.random.shuffle(sample)
sample
'''
Out:
array([[4, 5, 6],
       [1, 2, 3],
       [7, 8, 9]])
'''
sample = np.random.permutation(5)
sample
# Out: array([0, 4, 2, 3, 1])
```

rand()返回$[0,1)$上的均匀随机变量。而randint()得到$[0,low)$或者$[low,high)$的随机整数。randn返回标准正态随机变量。

还有一些常见的分布函数如下：

```
# uniform(low=0.0, high=1.0, size=None)
# 均匀分布
# binomial(n, p, size=None)
# 二项分布
# negative_binomial(n, p, size=None)
# 负二项分布
# multinomial(n, pvals, size=None)
# 多项分布
# normal(loc=0.0, scale=1.0, size=None)
# 正态分布
# standard_normal(size=None)
# 标准正态分布
```

```
# lognormal(mean=0.0, sigma=1.0, size=None)
# 对数正态分布
# multivariate_normal(mean, cov[, size,
check_valid, tol])
# 多元正态分布
# wald(mean, scale, size=None)
# 逆高斯分布
# poisson(lam=1.0, size=None)
# 泊松分布
# exponential(scale=1.0, size=None)
# 指数分布
# standard_exponential(size=None)
# 标准指数分布
# beta(a, b, size=None)
# β分布
# gamma(shape, scale=1.0, size=None)
# Γ分布
# standard_gamma(shape, size=None)
# 标准Γ分布
# chisquare(df, size=None)
# 卡方分布
# standard_t(df, size=None)
# 标准t分布
# f(dfnum, dfden, size=None)
# F分布
# noncentral_chisquare(df, nonc, size=None)
# 非中心卡方分布
# noncentral_f(dfnum, dfden, nonc, size=None)
# 非中心F分布
# geometric(p, size=None)
# 几何分布
# hypergeometric(ngood, nbad, nsample, size=None)
# 超几何分布
# pareto(a, size=None)
# 帕累托分布
# dirichlet(alpha, size=None)
# 狄利克雷分布
```

```
# gumbel(loc=0.0, scale=1.0, size=None)
# Gumbel分布(极值分布)
# laplace(loc=0.0, scale=1.0, size=None)
# 拉普拉斯分布
# logistic(loc=0.0, scale=1.0, size=None)
# Logistic分布
# rayleigh(scale=1.0, size=None)
# 瑞利分布
# standard_cauchy(size=None)
# 标准柯西分布
# triangular(left, mode, right, size=None)
# 三角分布
# weibull(a, size=None)
# 韦布尔分布
# zipf(a, size=None)
# ζ分布
```

```
# ----- Day 13 -----
```

Pandas库

Series

import加载库，同时加载最常使用的数据结构。

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

Series是一维的object，类似于一维数组。

```
obj = pd.Series([2, 4, -1, 0])
obj
```

```
'''
Out:
0    2
1    4
2   -1
3    0
dtype: int64
'''
obj.values
# Out: array([ 2,  4, -1,  0], dtype=int64)
obj.index
# Out: RangeIndex(start=0, stop=4, step=1)
```

但除了默认range之外，还可以自由指定index参数。同理可以通过自定义的index访问。

```
obj2 = pd.Series([2, 4, -1, 0], index=['a', 'c', 'b', 'd'])
obj2
'''
Out:
a    2
c    4
b   -1
d    0
dtype: int64
'''
obj2['b']
# Out: -1
obj2['c'] = 5
obj2[['a', 'b', 'c']]
'''
Out:
a    2
b   -1
c    5
dtype: int64
```

```
'''
```

pandas使用了很多NumPy和SciPy的功能，因而对于Series我们可以使用类NumPy的功能。

```
obj2[obj2 > 0]
'''
Out:
a    2
c    5
dtype: int64
'''
obj2 * 2
'''
Out:
a     4
c    10
b    -2
d     0
dtype: int64
'''
np.exp(obj2)
'''
Out:
a      7.389056
c    148.413159
b      0.367879
d      1.000000
dtype: float64
'''
```

同时也可以把Series看作一个顺序的dict，因而也有相关的操作。既可以直接将整个dict转化为Series，也可以通过index查找建立。

```
'b' in obj2
# Out: True
sdata = {'A':86, 'B':54, 'XXX':98}
```

```
obj3 = pd.Series(sdata)
obj3
'''
Out:
A      86
B      54
XXX    98
dtype: int64
'''
names = ['XXX', 'A', 'C']
obj4 = pd.Series(sdata, index=names)
obj4
'''
Out:
XXX    98.0
A      86.0
C       NaN
dtype: float64
'''
```

isnull()和notnull()可以用来检查missing data。

```
pd.isnull(obj4)
'''
Out:
XXX    False
A      False
C       True
dtype: bool
'''
```

Series之间可以直接用+合并。

```
obj3 + obj4
'''
Out:
A      172.0
B        NaN
C        NaN
XXX    196.0
dtype: float64
'''
```

Series及其index有着独立的名称属性。同时index可以被直接整体替换。

```
obj4.name = 'Score'
obj4.index.name = 'name'
obj4
'''
Out:
name
XXX    98.0
A      86.0
C       NaN
Name: Score, dtype: float64
'''
obj4.index = [1, 2, 3]
obj4
'''
Out:
1    98.0
2    86.0
3     NaN
Name: Score, dtype: float64
'''
```

```
# ----- Day 14 -----
```

DataFrame

DataFrame提供了一种类似于Excel的数据结构，可以看作是一个关于Series的dict。而且虽然DataFrame是二维的数据结构，你依然可以用高级技巧来表示高维数据。

最常见的创建DataFrame的方式是用一个dict，其中包含长度相同的list或者array。

```
data = {'state': ['Ohio', 'Ohio', 'Ohio',
'Nevada', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002,
2003],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
frame
'''
Out:
    state   year    pop
0   Ohio    2000    1.5
1   Ohio    2001    1.7
2   Ohio    2002    3.6
3   Nevada  2001    2.4
4   Nevada  2002    2.9
5   Nevada  2003    3.2
'''
```

对于一个较大的DataFrame，可以用head()来显示前五项。

利用columns参数可以指定各列的顺序。

```
pd.DataFrame(data, columns=['year', 'state', 'pop'])
'''
Out:
    year    state   pop
0   2000    Ohio    1.5
1   2001    Ohio    1.7
2   2002    Ohio    3.6
3   2001    Nevada  2.4
4   2002    Nevada  2.9
5   2003    Nevada  3.2
'''
```

我们也可以在创建时指定column，但不存在的参数会被设为missing data。

```
frame2 = pd.DataFrame(data,
                      columns=['year', 'state', 'pop', 'debt'],
                      index=['one', 'two', 'three', 'four', 'five', 'six'])
frame2
'''
Out:
        year    state   pop debt
one     2000    Ohio    1.5 NaN
two     2001    Ohio    1.7 NaN
three   2002    Ohio    3.6 NaN
four    2001    Nevada  2.4 NaN
five    2002    Nevada  2.9 NaN
six     2003    Nevada  3.2 NaN
'''
```

我们可以访问DataFrame的单个列，返回Series。

```
frame2['pop']
'''
Out:
one      1.5
two      1.7
three    3.6
four     2.4
five     2.9
six      3.2
Name: pop, dtype: float64
'''
type(frame2.year)
# Out: pandas.core.series.Series
```

要注意的是frame[column]对任何类型都合法，而frame.column只对合法的Python标识符有效。

利用loc[row]可以得到某行数据。

```
frame2.loc['three']
'''
Out:
year     2002
state    Ohio
pop       3.6
debt      NaN
Name: three, dtype: object
'''
```

我们可以对整一个column进行直接赋值。也可以导入Series，但导入Series后原本存在的数据也会被修改产生missing data。

```
frame2['debt'] = 1
frame2
'''
Out:
        year    state   pop debt
```

```
one     2000    Ohio    1.5 1
two     2001    Ohio    1.7 1
three   2002    Ohio    3.6 1
four    2001    Nevada  2.4 1
five    2002    Nevada  2.9 1
six     2003    Nevada  3.2 1
'''
frame2['debt'] = np.arange(6.)
frame2
'''
Out:
        year    state   pop debt
one     2000    Ohio    1.5 0.0
two     2001    Ohio    1.7 1.0
three   2002    Ohio    3.6 2.0
four    2001    Nevada  2.4 3.0
five    2002    Nevada  2.9 4.0
six     2003    Nevada  3.2 5.0
'''
val = pd.Series([-1, 5, 4], index=['one', 'five',
'three'])
frame2['debt'] = val
frame2
'''
Out:
        year    state   pop debt
one     2000    Ohio    1.5 -1.0
two     2001    Ohio    1.7 NaN
three   2002    Ohio    3.6 4.0
four    2001    Nevada  2.4 NaN
five    2002    Nevada  2.9 5.0
six     2003    Nevada  3.2 NaN
'''
```

用del可以删除column，也可以使用drop()。

drop()默认用来删除行，需要使用axis=1来删除列。drop()返回副本，如果要在原DataFrame上操作，则使用参数inplace=True。

```
del frame2['debt']
frame2
'''
Out:
        year    state   pop
one     2000    Ohio    1.5
two     2001    Ohio    1.7
three   2002    Ohio    3.6
four    2001    Nevada  2.4
five    2002    Nevada  2.9
six     2003    Nevada  3.2
'''
frame2.drop('state', axis=1)
'''
Out:
        year    pop
one     2000    1.5
two     2001    1.7
three   2002    3.6
four    2001    2.4
five    2002    2.9
six     2003    3.2
'''
frame2.drop('six', inplace=True)
frame2
'''
Out:
        year    state   pop
one     2000    Ohio    1.5
two     2001    Ohio    1.7
three   2002    Ohio    3.6
four    2001    Nevada  2.4
five    2002    Nevada  2.9
'''
```

```
# ----- Day 15 -----
```

我们也可以使用dict的dict来创建DataFrame。也可以在创建过程中指定index。

也可以使用Series的dict来创建。

```
pop = {'Nevada': {2001: 2.4, 2002: 2.9},
       'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
frame3 = pd.DataFrame(pop)
frame3
'''
Out:
        Nevada  Ohio
2000    NaN     1.5
2001    2.4     1.7
2002    2.9     3.6
'''
pd.DataFrame(pop, index=pd.Series([2001, 2002, 2003]))
'''
Out:
        Nevada  Ohio
2001    2.4     1.7
2002    2.9     3.6
2003    NaN     NaN
'''
```

对于DataFrame我们可以进行转置。

```
frame3.T
'''
Out:
        2000    2001    2002
Nevada  NaN     2.4     2.9
Ohio    1.5     1.7     3.6
'''
```

我们可以设定DataFrame的index和columns的name。

```
frame3.index.name = 'year'
frame3.columns.name = 'state'
frame3
'''
Out:
state   Nevada  Ohio
year
2000    NaN     1.5
2001    2.4     1.7
2002    2.9     3.6
'''
```

values则会返回一个二维的array。

```
frame3.values
'''
Out:
array([[nan, 1.5],
       [2.4, 1.7],
       [2.9, 3.6]])
'''
```

```
# ----- Day 16 -----
```

index对象

pandas库的index是一种单独的数据类型，能够进行许多操作。

```
obj = pd.Series(range(3), index=['a', 'b', 'c'])
obj
'''
Out:
a    0
b    1
c    2
dtype: int64
'''
index = obj.index
index
# Out: Index(['a', 'b', 'c'], dtype='object')
```

index类型属于顺序容器，可以区间读取。但不能进行修改，确保了index类型的安全性。

```
index[:2]
# Out: Index(['a', 'b'], dtype='object')
```

可以使用Index()将list或Series转化为index类型。但不同情况下的index可能不同，可以使用index参数来复制index。

```
labels = pd.Index(np.arange(3))
labels
# Out: Int64Index([0, 1, 2], dtype='int64')
obj2 = pd.Series(range(3), index=range(3))
obj2.index
# Out: RangeIndex(start=0, stop=3, step=1)
obj2.index is labels
# Out: False
obj2.index = labels
obj2.index is labels
# Out: True
```

除了is之外，还可以使用in来对单独的值进行判断。

```
'b' in obj.index
# Out: True
```

另外，index类型虽然和set有很多类似操作，但却可以包含相同的值。而每次操作将会选中所有的相同值。

常用方法有:

append(), difference(), intersection(), union();

isin()返回boolean数组;

delete(), drop(), insert();其中delete()按照位置删除，drop()寻找指定元素删除。

is_monotonic()判断index是否单调不减;

is_unique();

unique()返回一个unique的index。

```
# ----- Day 17 -----
```

基本函数

reindex()可以重新定义Series的索引顺序，并且可以使用method参数来控制缺失值的填充方式。ffill和bfill分别根据index的升序(而非实际的顺序)向前或向后填充。

```
obj = pd.Series([3, 7 ,8 ,9], index=['a', 'b', 'c', 'e'])
obj
'''
Out:
a    3
b    7
```

```
c    8
e    9
dtype: int64
'''
obj.reindex(['b', 'c', 'd', 'e', 'f'])
'''
Out:
b    7.0
c    8.0
d    NaN
e    9.0
f    NaN
dtype: float64
'''
obj.reindex(['b', 'e', 'c', 'd', 'f'],
method='ffill')
'''
Out:
b    7
e    9
c    8
d    8
f    9
dtype: int64
'''
obj.reindex(['b', 'e', 'c', 'd', 'f'],
method='bfill')
'''
Out:
b    7.0
e    9.0
c    8.0
d    9.0
f    NaN
dtype: float64
'''
```

类似的，reindex可以作用于DataFrame，但需要指定index或者columns。同时指定index和columns的情况也可以使用loc()来代替reindex。

```
frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
                     index=['a', 'c', 'd'],
                     columns=['Ohio', 'Texas', 'California'])
frame
'''
Out:
    Ohio    Texas   California
a   0       1       2
c   3       4       5
d   6       7       8
'''
frame2 = frame.reindex(['a', 'b', 'c', 'd'])
frame2
'''
Out:
    Ohio    Texas   California
a   0.0     1.0     2.0
b   NaN     NaN     NaN
c   3.0     4.0     5.0
d   6.0     7.0     8.0
'''
states = ['Texas', 'Utah', 'California']
frame.reindex(index=['a', 'b', 'c', 'd'], columns=states)
'''
Out:
    Texas   Utah    California
a   1.0     NaN     2.0
b   NaN     NaN     NaN
c   4.0     NaN     5.0
d   7.0     NaN     8.0
```

```
'''
```

可以通过fill_value参数来控制missing data的值，limit参数用来限制自动填充的最大长度，tolerance则表示填充间隔。

level是reindex中最难以理解的一个参数，它表示多层索引上的简单索引匹配。

要建立多层索引表，可以直接隐式构造。也可以通过MultiIndex显式构造。

```
series1 = Series(np.random.randint(0,150,size=6),
                 index=[['a','a','b','b','c','c'],
                        ['期中','期末','期中','期末','期中','期末']])
series1
'''
Out:
a  期中    123
   期末      3
b  期中     17
   期末     80
c  期中     88
   期末    108
dtype: int32
'''
frame1 = DataFrame(np.random.randint(0,150,size=(4,6)),
                   index = list('ABCD'),
                   columns=[['python','python','math','math','En','En'],
                            ['期中','期末','期中','期末','期中','期末']])
frame1
'''
Out:
    python      math        En
```

```
    期中  期末  期中  期末  期中  期末
A   97  15    145 23    19  82
B   67  18    138 116   139 78
C   26  45    36  104   42  85
D   64  107   119 24    11  97
'''
```

不过不建议用多层索引，会带来很多不必要的麻烦。

```
# ----- Day 18 -----
```

数据分析

数据存储