

# 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

```
1 %run
2 %load
3 %debug
4 %xmode Plain # less detailed
5 %xmode Verbose # more detailed
6 %matplotlib inline # draw in Jupyter
7 import numpy as np
8 import matplotlib.pyplot as plt
9 plt.plot(np.random.rand(50).cumsum())
```

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

## Python基础

#注释。

```
1 # 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。

```
1 def append_element(some_list, element):
2     some_list.append(element)
3     return
4
5 data = [1, 2, 3]
6 append_element(data, 4)
7 data
8 # 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。

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

print配合format输出。

```
1 a = 1; b = 1.5
2 print('a is {1}, b is {0}'.format(type(b),
3 # 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, setattr, setattr操作。其中getattr可以直接使用返回对象，setattr不改变原class。

```
1 class A(object):
2     def set(self, a, b):
3         x = a
4         a = b
5         b = x
6         print a, b
7
8 a = A()
9 c = getattr(a, 'set')
10 c(a='1', b='2')
11 # Out: 2 1
```

可以利用try()检查容器是否是顺序，同时生成list。

```
1 def isiterable(obj):
2     try:
3         iter(obj)
4         return True
5     except TypeError:
```

```
6         return False
7     return
8
9 x = 'string'
10 iterable(x)
11 # Out: True
12 iterable(5)
13 # Out: False
14
15 if not isinstance(x, list) and iterable(x):
16     x = list(x)
17 x
18 # Out: ['s', 't', 'r', 'i', 'n', 'g']
```

import和as的使用。

is和is not的使用。

```
1 a = [1, 2, 3]
2 b = a
3 c = list(a)
4 d = None
5 print(a is b)
6 # Out: True
7 print(a is not c)
8 # Out: True
9 print(a == c)
10 # Out: True
11 print(d is None)
12 # Out: True
```

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

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

```
1 # ----- 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。

```
1 c = """
2 This is a longer string that
3 spans multiple lines
4 """
5 c.count('\n')
6 # 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。

```
1 a = 'this is a string'
2 b = a.replace('string', 'longer string')
3 print(b)
4 # Out: this is a longer string
5 print(a)
6 # 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。

```
1 s = '12\\34'
2 print(s)
3 # Out: 12\34
4 s = r'this\has\no\special\characters'
5 s
6 # Out: 'this\\has\\no\\special\\characters'
```

format和{}可以进行格式化输出。

```
1 template = '{0:.2f} {1:s} are worth US${2:d}'
2 template.format(4.5560, 'Argentine Pesos', 1)
3 # 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。

```
1 val = "español"
2 val_utf8 = val.encode('utf-8')
3 val_utf8
4 # Out: b'espa\xc3\xb1ol'
5 type(val_utf8)
6 # Out: bytes
7 val_utf8.decode('utf-8')
8 # Out: 'español'
9 bytes_val = b'this is bytes'
10 bytes_val
11 # Out: b'this is bytes'
12 decoded = bytes_val.decode('utf8')
13 decoded
14 # Out: 'this is bytes'
```

None是一个单独的类型，常见于函数的默认参数中。

```
1 def add_and_maybe_multiply(a, b, c=None):
2     result = a + b
3     if c is not None:
4         result = result * c
5     return result
6
7 type(None)
8 # Out: NoneType
```

and和or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。

```
1 from datetime import datetime, date, time
2 dt = datetime(2011, 10, 29, 20, 30, 21)
3 dt.day
4 # Out: 29
5 dt.date()
6 # Out: datetime.date(2011, 10, 29)
7 dt.strftime('%m/%d/%Y %H:%M')
8 # Out: '10/29/2011 20:30'
9 datetime.strptime('20091031', '%Y%m%d')
10 # Out: datetime.datetime(2009, 10, 31, 0, 0)
11 dt.replace(minute=0, second=0)
12 # Out: datetime.datetime(2011, 10, 29, 20, 0)
```

时间差用timedelta表示，单位是天和秒。

```
1 dt2 = datetime(2011, 11, 15, 22, 30)
2 delta = dt2 - dt
3 delta
4 # Out: datetime.timedelta(17, 7179)
5 type(delta)
6 # Out: datetime.timedelta
7 dt + delta
8 # Out: datetime.datetime(2011, 11, 15, 22, 30)
```

时间的格式符号(format specification)如下

```
1 %Y #4位年数
2 %y #2位年数
3 %m #2位月数
4 %d #2位天数
5 %H #24小时制(2位)
6 %I #12小时制(2位)
7 %M #2位分钟数
8 %S #2位秒数，由于闰秒存在，范围为[00, 61]
```

```
9 %w #星期，范围为[0, 6]
10 %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
11 %W #同上，以周一为标准
12 %z #用+HHMM或-HHMM表示相对于格林威治的时区偏移
13 %Z #时区名称，默认为空
14 %F #等价于%Y-%m-%d
15 %D #等价于%m/%d/%y
```

if, elif, else语句。单行if。

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

for in语句。多变量for循环。

```
1 starts = [0,1,2,3,4]
2 ends = [5,6,7,8,9]
3 for start, end in zip(starts, ends):
4     print((start, end))
5 '''
6 Out:
7 (0, 5)
8 (1, 6)
9 (2, 7)
10 (3, 8)
11 (4, 9)
12 '''
```

while语句。

range()函数。

```
1 # ----- 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。

```
1 tup = 3, (4, 5, 6), (7, 8)
2 tup
3 # Out: (3, (4, 5, 6), (7, 8))
4 tuple([4, 0, 2])
5 # Out: (4, 0, 2)
6 tuple('string')
7 # Out: ('s', 't', 'r', 'i', 'n', 'g')
8 tuple(['foo', [1, 2], True])
9 # Out: ('foo', [1, 2], True)
10 tuple(range(10))
11 # 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。

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

也可以配合多变量for循环。

```

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

```

用\*可以代指tuple的剩余部分。

```

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

```

count是tuple的常用方法。

```

1 a = (1, 2, 2, 2, 3, 4, 2)
2 a.count(2)
3 # 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。

```

1 tup = ('foo', 'bar', 'baz')
2 b_list = list(tup)
3 b_list
4 # Out: ['foo', 'bar', 'baz']
5 b_list.append('dwarf')
6 b_list
7 # Out: ['foo', 'bar', 'baz', 'dwarf']
8 b_list.insert(1, 'red')

```

```

9 b_list
10 # Out: ['foo', 'red', 'bar', 'baz', 'dwarf']
11 b_list.extend([7, 8, (2, 3)])
12 b_list
13 # Out: ['foo', 'red', 'bar', 'baz', 'dwarf', 7,
14         8, (2, 3)]
15 b_list.pop(2)
16 b_list
17 # Out: ['foo', 'red', 'baz', 'dwarf', 7, 8, (2,
18         3)]
19 b_list.append('foo')
20 b_list.remove('foo')
21 b_list
22 # Out: ['red', 'baz', 'dwarf', 7, 8, (2, 3),
23         '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。

```

1 import bisect
2 a = [7, 2, 2, 5, 1, 3]
3 a.sort()
4 a
5 # Out: [1, 2, 2, 3, 5, 7]
6 b = ['saw', 'small', 'He', 'foxes', 'six']
7 b.sort(key=len)
8 b
9 # Out: ['He', 'saw', 'six', 'small', 'foxes']
10 import bisect
11 bisect.bisect(a, 2)
12 # Out: 3
13 bisect.bisect(a, 6)
14 # Out: 5
15 bisect.insort(a, 6)
16 a
17 # Out: [1, 2, 2, 3, 5, 6, 7]
18 sorted('horse race')

```

```
19 # Out: [' ', 'a', 'c', 'e', 'e', 'h', 'o', 'r',  
    'r', 's']
```

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。

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

enumerate和zip可以用于生成变量组合。

```
1 some_list = ['foo', 'bar', 'baz']  
2 mapping = {}  
3 for i, v in enumerate(some_list):  
4     mapping[v] = i  
5 mapping  
6 # Out: {'foo': 0, 'bar': 1, 'baz': 2}  
7 seq1 = ['foo', 'bar', 'baz']  
8 seq2 = ['one', 'two', 'three']  
9 zipped = zip(seq1, seq2)  
10 list(zipped)  
11 # Out: [('foo', 'one'), ('bar', 'two'), ('baz',  
    'three')]  
12 seq3 = [False, True]  
13 list(zip(seq1, seq2, seq3))  
14 # Out: [('foo', 'one', False), ('bar', 'two',  
    True)]  
15 for i, (a, b) in enumerate(zip(seq1, seq2)):  
16     print('{0}: {1}, {2}'.format(i, a, b))  
17 '''  
18 Out:  
19 0: foo, one  
20 1: bar, two  
21 2: baz, three  
22 '''
```

zip也可以反过来进行解压。

```
1 | pitchers = [('No1an', 'Ryan'), ('Roger',  
2 |             'Clemens'),  
3 |             ('Schilling', 'Curt')]  
4 | first_names, last_names = zip(*pitchers)  
5 | first_names  
# Out: ('No1an', 'Roger', 'Schilling')
```

```
1 | # ----- Day 3 -----
```

## Numpy库

---