pandas为我们提供了多种切片方法，而要是不太了解这些方法，就会经常容易混淆。下面举例对这些切片方法进行说明。

## 数据介绍

先随机生成一组数据：

In [5]: rnd\_1 = [random.randrange(1,20) for x in xrange(1000)]

...: rnd\_2 = [random.randrange(1,20) for x in xrange(1000)]

...: rnd\_3 = [random.randrange(1,20) for x in xrange(1000)]

...: fecha = pd.date\_range('2012-4-10', '2015-1-4')

...:

...: data = pd.DataFrame({'fecha':fecha, 'rnd\_1': rnd\_1, 'rnd\_2': rnd\_2, 'rnd\_3': rnd\_3})

In [6]: data.describe()

Out[6]:

rnd\_1 rnd\_2 rnd\_3

count 1000.000000 1000.000000 1000.000000

mean 9.946000 9.825000 9.894000

std 5.553911 5.559432 5.423484

min 1.000000 1.000000 1.00000025% 5.000000 5.000000 5.00000050% 10.000000 10.000000 10.00000075% 15.000000 15.000000 14.000000

max 19.000000 19.000000 19.000000

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18

## []切片方法

使用方括号能够对DataFrame进行切片，有点类似于python的列表切片。按照索引能够实现行选择或列选择或区块选择。

# 行选择

In [7]: data[1:5]

Out[7]:

fecha rnd\_1 rnd\_2 rnd\_31 2012-04-11 1 16 32 2012-04-12 7 6 13 2012-04-13 2 16 74 2012-04-14 4 17 7

# 列选择

In [10]: data[['rnd\_1', 'rnd\_3']]

Out[10]:

rnd\_1 rnd\_30 8 121 1 32 7 13 2 74 4 75 12 86 2 127 9 88 13 179 4 710 14 1411 19 1612 2 1213 15 1814 13 1815 13 1116 17 717 14 1018 9 619 11 1520 16 1321 18 922 1 1823 4 324 6 1125 2 1326 7 1727 11 828 3 1229 4 2

.. ... ...970 8 14971 19 5972 13 2973 8 10974 8 17975 6 16976 3 2977 12 6978 12 10979 15 13980 8 4981 17 3982 1 17983 11 5984 7 7985 13 14986 6 19987 13 9988 3 15989 19 6990 7 11991 11 7992 19 12993 2 15994 10 4995 14 13996 12 11997 11 15998 17 14999 3 8

[1000 rows x 2 columns]

# 区块选择

In [11]: data[:7][['rnd\_1', 'rnd\_2']]

Out[11]:

rnd\_1 rnd\_20 8 171 1 162 7 63 2 164 4 175 12 196 2 7

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88

不过对于多列选择，不能像行选择时一样使用1：5这样的方法来选择。

In [12]: data[['rnd\_1':'rnd\_3']]

File "<ipython-input-13-6291b6a83eb0>", line 1

data[['rnd\_1':'rnd\_3']]

^

SyntaxError: invalid syntax

* 1
* 2
* 3
* 4
* 5

## loc

loc可以让你按照索引来进行行列选择。

In [13]: data.loc[1:5]Out[13]:

fecha rnd\_1 rnd\_2 rnd\_3

1 2012-04-11 1 16 3

2 2012-04-12 7 6 1

3 2012-04-13 2 16 7

4 2012-04-14 4 17 7

5 2012-04-15 12 19 8

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

这里需要注意的是，loc与第一种方法不同之处在于会把第5行也选择进去，而第一种方法只会选择到第4行为止。

data.loc[2:4, ['rnd\_2', 'fecha']]

Out[14]:

rnd\_2 fecha2 6 2012-04-123 16 2012-04-134 17 2012-04-14

* 1
* 2
* 3
* 4
* 5
* 6

loc能够选择在两个特定日期之间的数据，需要注意的是这两个日期必须都要在索引中。

In [15]: data\_fecha = data.set\_index('fecha')

...: data\_fecha.head()

Out[15]:

rnd\_1 rnd\_2 rnd\_3

fecha 2012-04-10 8 17 122012-04-11 1 16 32012-04-12 7 6 12012-04-13 2 16 72012-04-14 4 17 7

In [16]: # 生成两个特定日期

...: fecha\_1 = dt.datetime(2013, 4, 14)

...: fecha\_2 = dt.datetime(2013, 4, 18)

...:

...: # 生成切片数据

...: data\_fecha.loc[fecha\_1: fecha\_2]

Out[16]:

rnd\_1 rnd\_2 rnd\_3

fecha 2013-04-14 17 10 52013-04-15 14 4 92013-04-16 1 2 182013-04-17 9 15 12013-04-18 16 7 17

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25

**更新**：如果没有特殊需求，强烈建议使用loc而尽量少使用[]，因为loc在对DataFrame进行重新赋值操作时会避免chained indexing问题，使用[]时编译器很可能会给出**SettingWithCopy**的警告。具体可以参见官方文档：[http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy](http://pandas.pydata.org/pandas-docs/stable/indexing.html" \l "indexing-view-versus-copy" \t "_blank)

## iloc

如果说loc是按照索引（index）的值来选取的话，那么iloc就是按照索引的位置来进行选取。iloc不关心索引的具体值是多少，只关心位置是多少，所以使用iloc时方括号中只能使用数值。

# 行选择In [17]: data\_fecha[10: 15]Out[17]:

rnd\_1 rnd\_2 rnd\_3

fecha 2012-04-20 14 6 142012-04-21 19 14 162012-04-22 2 6 122012-04-23 15 8 182012-04-24 13 8 18

# 列选择In [18]: data\_fecha.iloc[:,[1,2]].head()Out[18]:

rnd\_2 rnd\_3

fecha 2012-04-10 17 122012-04-11 16 32012-04-12 6 12012-04-13 16 72012-04-14 17 7

# 切片选择In [19]: data\_fecha.iloc[[1,12,34],[0,2]]Out[19]:

rnd\_1 rnd\_3

fecha 2012-04-11 1 32012-04-22 2 122012-05-14 17 10

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30

## at

at的使用方法与loc类似，但是比loc有更快的访问数据的速度，而且只能访问单个元素，不能访问多个元素。

In [20]: timeit data\_fecha.at[fecha\_1,'rnd\_1']

The slowest run took 3783.11 times longer than the fastest. This could mean that an intermediate result is being cached.100000 loops, best of 3: 11.3 µs per loop

In [21]: timeit data\_fecha.loc[fecha\_1,'rnd\_1']

The slowest run took 121.24 times longer than the fastest. This could mean that an intermediate result is being cached.10000 loops, best of 3: 192 µs per loop

In [22]: data\_fecha.at[fecha\_1,'rnd\_1']

Out[22]: 17

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

## iat

iat对于iloc的关系就像at对于loc的关系，是一种更快的基于索引位置的选择方法，同at一样只能访问单个元素。

In [23]: data\_fecha.iat[1,0]Out[23]: 1

In [24]: timeit data\_fecha.iat[1,0]

The slowest run took 6.23 times longer than the fastest. This could mean that an intermediate result is being cached.100000 loops, best of 3: 8.77 µs per loop

In [25]: timeit data\_fecha.iloc[1,0]10000 loops, best of 3: 158 µs per loop

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

## ix

以上说过的几种方法都要求查询的秩在索引中，或者位置不超过长度范围，而ix允许你得到不在DataFrame索引中的数据。

In [28]: date\_1 = dt.datetime(2013, 1, 10, 8, 30)

...: date\_2 = dt.datetime(2013, 1, 13, 4, 20)

...:

...: # 生成切片数据

...: data\_fecha.ix[date\_1: date\_2]

Out[28]:

rnd\_1 rnd\_2 rnd\_3

fecha 2013-01-11 19 17 192013-01-12 10 9 172013-01-13 15 3 10

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

如上面的例子所示，2013年1月10号并没有被选择进去，因为这个时间点被看作为0点0分，比8点30分要早一些。

--------------------- 本文来自 ForeseeMark 的CSDN 博客 ，全文地址请点击：https://blog.csdn.net/wr339988/article/details/65446138?utm\_source=copy