**python之pandas的基本使用（2）**

2017年01月22日 18:16:02 [cxmscb](https://me.csdn.net/cxmscb) 阅读数：6861 标签： [python](http://so.csdn.net/so/search/s.do?q=python&t=blog)[pandas](http://so.csdn.net/so/search/s.do?q=pandas&t=blog) 更多

个人分类： [机器学习](https://blog.csdn.net/cxmscb/article/category/6147299)[pythonNote](https://blog.csdn.net/cxmscb/article/category/6094823)

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/cxmscb/article/details/54670113

续 [python之pandas模块的基本使用（1）](http://blog.csdn.net/cxmscb/article/details/54632492)

**一、排序和排名**

1. 排序：sort\_index和sort\_values函数

代码示例：

print 'Series排序'

x = Series(range(4), index = ['b', 'a', 'c', 'd'])

print x.sort\_index() # Series按索引排序

'''

a 1

b 0

c 2

d 3

'''

print x.sort\_values() # Series按值排序

'''

b 0

a 1

c 2

d 3

'''

print 'DataFrame按索引排序'

frame = DataFrame(numpy.arange(8).reshape((2, 4)),

index = ['b', 'a'],

columns = list('ABDC'))

print frame

'''

A B D C

b 0 1 2 3

a 4 5 6 7

'''

print frame.sort\_index() # 根据行索引来排序

'''

A B D C

a 4 5 6 7

b 0 1 2 3

'''

print frame.sort\_index(axis = 1) #根据列索引来排序

'''

A B C D

b 0 1 3 2

a 4 5 7 6

'''

print frame.sort\_index(axis = 1, ascending = False) # 设置降序排序

'''

D C B A

b 2 3 1 0

a 6 7 5 4

'''

print 'DataFrame按列的值排序'

frame = DataFrame({'b':[4, 7, -3, 2], 'a':[0, 1, 0, 1]})

print frame

'''

a b

0 0 4

1 1 7

2 0 -3

3 1 2

'''

print frame.sort\_values(by = 'b') # 指定b这列的值进行排序

'''

a b

2 0 -3

3 1 2

0 0 4

1 1 7

'''

print frame.sort\_values(by = ['a', 'b']) #先a后b进行列的值排序

'''

a b

2 0 -3

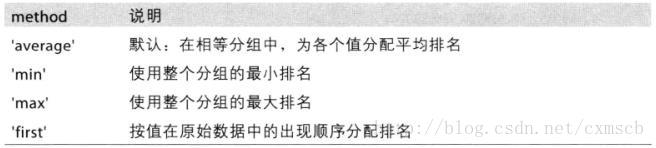
0 0 4

3 1 2

1 1 7

'''

1. 排名：根据值的大小/出现次数来进行排名，得到一组排名值：rank函数



print 'rank：默认升序，排名值从1开始'

obj = Series([4, 2, 0, 4],index = ['a','b','c','d'])

# 以值从小到大来赋排名值：c:0(1) b:2(2) a:4(3) d:4(4)

print obj.rank()

'''

a 3.5 求平均值(4+3)/2

b 2.0

c 1.0

d 3.5

'''

print obj.rank(method = 'first') # 按出现顺序排名，不求平均值。

'''

a 3.0

b 2.0

c 1.0

d 4.0

'''

print obj.rank(ascending = False, method = 'max') # 逆序，并取排名值最大值。所以-5的rank是7

# a:4(1) d:4(2) b:2(3) c:0(4)

'''

dtype: float64

a 2.0

b 3.0

c 4.0

d 2.0

'''

frame = DataFrame({'b':[4.3, 7, -3, 2],

'a':[0, 1, 0, 1],

'c':[-2, 5, 8, -2.5]})

print frame

'''

a b c

0 0 4.3 -2.0

1 1 7.0 5.0

2 0 -3.0 8.0

3 1 2.0 -2.5

'''

print frame.rank(axis = 1) # 按行进行排名，默认升序

'''

a b c

0 2.0 3.0 1.0

1 1.0 3.0 2.0

2 2.0 1.0 3.0

3 2.0 3.0 1.0

'''

**二、索引重复的情况**

代码示例：

print '重复索引:进行两次索引'

obj = Series([0,1,2,3,4], index = ['a', 'a', 'b', 'b', 'c'])

print obj.index.is\_unique # 判断是非有重复索引

# False

print obj['a'][0]

# 0

print obj.a[1]

# 1

df = DataFrame(numpy.arange(12).reshape(4, 3), index = ['a', 'a', 'b', 'b'])

print df

'''

0 1 2

a 0 1 2

a 3 4 5

b 6 7 8

b 9 10 11

'''

print df.ix['b'].ix[0] // 两次行索引

'''

0 6

1 7

2 8

Name: b, dtype: int32

'''

print df.ix['b'].ix[1]

'''

0 9

1 10

2 11

Name: b, dtype: int32

'''

**三、汇总和计算描述统计**

常用方法选项：



常用汇总统计函数 I：



常用汇总统计函数 II：



代码示例：

print '求和'

df = DataFrame([[1, numpy.nan], [7, 4], [numpy.nan, numpy.nan], [0, 1]],

index = ['a', 'b', 'c', 'd'],

columns = ['one', 'two'])

print df

'''

one two

a 1.0 NaN

b 7.0 4.0

c NaN NaN

d 0.0 1.0

'''

print df.sum() # 按列求和

# 排除缺失值，skipna默认值为True

'''

one 8.0

two 5.0

dtype: float64

'''

print df.sum(skipna = False)

'''

one NaN

two NaN

'''

print df.sum(axis = 1) # 按行求和

'''

a 1.0

b 11.0

c 0.0

d 1.0

dtype: float64

'''

print '求平均数'

print df.mean(axis = 1, skipna = False)

'''

a NaN

b 5.5

c NaN

d 0.5

'''

print df.mean(axis = 1)

'''

a 1.0

b 5.5

c NaN

d 0.5

'''

print '其它函数'

print df

'''

one two

a 1.0 NaN

b 7.0 4.0

c NaN NaN

d 0.0 1.0

'''

print df.idxmax() # 计算每一列最大值的索引

'''

one b

two b

'''

print df.cumsum() # 每一列的累加和

'''

one two

a 1.0 NaN

b 8.0 4.0

c NaN NaN

d 8.0 5.0

'''

print df.describe() # 对DataFrame每列计算汇总统计

'''

one two

count 3.000000 2.00000

mean 2.666667 2.50000

std 3.785939 2.12132

min 0.000000 1.00000

25% NaN NaN

50% NaN NaN

75% NaN NaN

max 7.000000 4.00000

'''

obj = Series([2,4,8,4], index = ['a', 'a', 'b', 'c'])

print obj.describe() # 对Series计算汇总统计

'''

count 4.000000

mean 4.500000

std 2.516611

min 2.000000

25% 3.500000

50% 4.000000

75% 5.000000

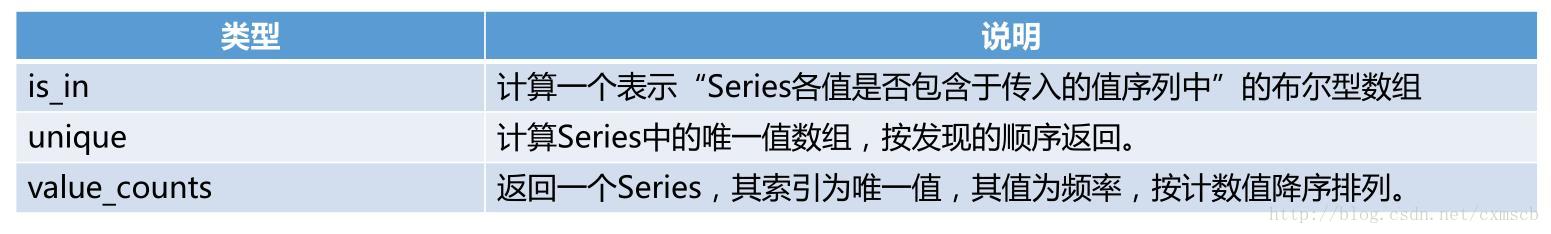
max 8.000000

dtype: float64

'''

**四、去重和成员出现计数**

主要方法：



print '去重'

obj = Series(['c', 'a', 'd','b', 'b', 'c'])

print obj.unique()

'''

['c' 'a' 'd' 'b']

'''

print obj.value\_counts()

'''

b 2

c 2

d 1

a 1

'''

print '判断元素存在'

mask = obj.isin(['b', 'c'])

print mask

'''

0 True

1 False

2 False

3 True

4 True

5 True

'''

print obj[mask] #只打印元素b和c

'''

0 c

3 b

4 b

5 c

'''

data = DataFrame({'Qu1':[1, 3, 4, 3, 4],

'Qu2':[2, 3, 1, 2, 3],

'Qu3':[1, 5, 2, 4, 4]})

print data

'''

Qu1 Qu2 Qu3

0 1 2 1

1 3 3 5

2 4 1 2

3 3 2 4

4 4 3 4

'''

print data.apply(pd.value\_counts).fillna(0)

# 计算每列中各个数字出现的次数，缺失值为0

'''

Qu1 Qu2 Qu3

1 1.0 1.0 1.0

2 0.0 2.0 1.0

3 2.0 2.0 0.0

4 2.0 0.0 2.0

5 0.0 0.0 1.0

'''

print data.apply(pd.value\_counts, axis = 1).fillna(0)

# 计算每行中各个数字出现的次数，缺失值为0

'''

1 2 3 4 5

0 2.0 1.0 0.0 0.0 0.0

1 0.0 0.0 2.0 0.0 1.0

2 1.0 1.0 0.0 1.0 0.0

3 0.0 1.0 1.0 1.0 0.0

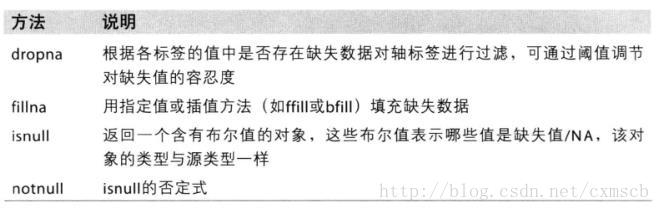
4 0.0 0.0 1.0 2.0 0.0

'''

**五、处理缺失数据**

• NaN（Not a Number）表示浮点数和非浮点数组中的缺失数据，None也被当作NA处理。

处理缺失数据函数：



• dropna 函数：DatFrame默认丢弃任何含有缺失值的行。how参数控制行为，axis参数选择轴，thresh参数控制NaN数量的要求。

• fillna函数： inplace参数决定返回新对象还是就地修改

代码示例：

print '作为null处理的值'

string\_data = Series(['a', 'b', numpy.nan, 'd'])

print string\_data

'''

0 a

1 b

2 NaN

3 d

'''

print string\_data.isnull()

'''

0 False

1 False

2 True

3 False

'''

string\_data[0] = None

print string\_data

'''

0 None

1 b

2 NaN

3 d

'''

# None也被当作NA处理

print string\_data.isnull()

'''

0 True

1 False

2 True

3 False

'''

from numpy import nan as NA

print '丢弃缺失数据NaN'

data = Series([1, NA, 3.5, NA, 7])

print data.dropna()

'''

0 1.0

2 3.5

4 7.0

'''

print 'DataFrame对丢弃NA的处理'

data = DataFrame([[1., 6.5, 3.], [1., NA, NA],

[NA, NA, NA], [NA, 6.5, 3.]])

print data

'''

0 1 2

0 1.0 6.5 3.0

1 1.0 NaN NaN

2 NaN NaN NaN

3 NaN 6.5 3.0

'''

print data.dropna() # 默认只要某行有NA就全部删除

'''

0 1 2

0 1.0 6.5 3.0

'''

print data.dropna(how = 'all') # 某行全部为NA才删除

'''

0 1 2

0 1.0 6.5 3.0

1 1.0 NaN NaN

3 NaN 6.5 3.0

'''

data[0] = NA

print data.dropna(axis = 1, how = 'all') #某行有NA就全部删除

'''

1 2

0 6.5 3.0

1 NaN NaN

2 NaN NaN

3 6.5 3.0

'''

data = DataFrame(numpy.arange(21).reshape(7, 3))

data.ix[:4, 1] = NA

data.ix[:2, 2] = NA

print data

'''

0 1 2

0 0 NaN NaN

1 3 NaN NaN

2 6 NaN NaN

3 9 NaN 11.0

4 12 NaN 14.0

5 15 16.0 17.0

6 18 19.0 20.0

'''

print data.dropna(thresh = 2) # 每行至少要有2个非NA元素则删除

'''

0 1 2

3 9 NaN 11.0

4 12 NaN 14.0

5 15 16.0 17.0

6 18 19.0 20.0

'''

print '填充0'

df = DataFrame(numpy.arange(9).reshape(3, 3))

df.ix[:1, 1] = NA

df.ix[:2, 2] = NA

print df.fillna(0) # 默认inplace为False

'''

0 1 2

0 0 0.0 0.0

1 3 0.0 0.0

2 6 7.0 0.0

'''

print df

'''

0 1 2

0 0 NaN NaN

1 3 NaN NaN

2 6 7.0 NaN

'''

df.fillna(0, inplace = True) # 就地修改

print df

'''

0 1 2

0 0 0.0 0.0

1 3 0.0 0.0

2 6 7.0 0.0

'''

df = DataFrame(numpy.arange(9).reshape(3, 3))

df.ix[:1, 1] = NA

df.ix[:2, 2] = NA

print '不同行列填充不同的值'

print df.fillna({1:0.5, 2:-1}) # 第3列不存在

'''

0 1 2

0 0 0.5 -1.0

1 3 0.5 -1.0

2 6 7.0 -1.0

'''

print '不同的填充方式'

print df

'''

0 1 2

0 0 NaN NaN

1 3 NaN NaN

2 6 7.0 NaN

'''

print df.fillna(method = 'bfill') # 向前填充

'''

0 1 2

0 0 7.0 NaN

1 3 7.0 NaN

2 6 7.0 NaN

'''

print df.fillna(method = 'bfill', limit = 1) # 只可向前填充一步

'''

0 1 2

0 0 NaN NaN

1 3 7.0 NaN

2 6 7.0 NaN

'''

print '用统计数据填充'

data = Series([1, NA, 2, NA, 3])

print data.fillna(data.mean())

'''

0 1.0

1 2.0

2 2.0

3 2.0

4 3.0

'''

**六、多层次化索引**

1. 对Series和DataFrame进行多层次的索引MultiIndex，通过stack与unstack进行Series和DataFrame的变换。

代码示例：

from pandas import MultiIndex

print 'Series的多层次索引'

data = Series(numpy.arange(8),

index = [['a', 'a', 'b', 'b', 'c', 'c', 'd','d'],

[1, 2, 1, 2, 1, 2, 1,2]])

print data #　两层行索引

'''

a 1 0

2 1

b 1 2

2 3

c 1 4

2 5

d 1 6

2 7

'''

print data.index

'''

MultiIndex(levels=[[u'a', u'b', u'c', u'd'], [1, 2]],

labels=[[0, 0, 1, 1, 2, 2, 3], [0, 1, 0, 1, 0, 1, 0]])

'''

print data.b

'''

1 2

2 3

'''

print data['b':'c'] # 闭区间

'''

b 1 2

2 3

c 1 4

2 5

'''

print data[:2] # 数组索引不区分标签

'''

a 1 0

2 1

'''

print data.unstack() #将Series转换为DataFrame

'''

1 2

a 0 1

b 2 3

c 4 5

d 6 7

'''

print data.unstack().stack() # 将DataFrame转换回Series

'''

a 1 0

2 1

b 1 2

2 3

c 1 4

2 5

d 1 6

2 7

'''

print

print 'DataFrame的多层次化索引'

frame = DataFrame(numpy.arange(12).reshape((4, 3)),

index = [['a', 'a', 'b', 'b'], [1, 2, 1, 2]],

columns = [['A', 'A', 'B'], ['A1', 'A2', 'B1']])

print frame # 两层行索引和两层列索引

'''

A B

A1 A2 B1

a 1 0 1 2

2 3 4 5

b 1 6 7 8

2 9 10 11

'''

print frame.index

'''

MultiIndex(levels=[[u'a', u'b'], [1, 2]],

labels=[[0, 0, 1, 1], [0, 1, 0, 1]])

'''

print frame.columns

'''

MultiIndex(levels=[[u'A', u'B'], [u'A1', u'A2', u'B1']],

labels=[[0, 0, 1], [0, 1, 2]])

'''

frame.index.names = ['key1', 'key2']

frame.columns.names = ['state', 'more']

print frame

'''

state A B

more A1 A2 B1

key1 key2

a 1 0 1 2

2 3 4 5

b 1 6 7 8

2 9 10 11

'''

print frame.ix['a', 1]

'''

A A1 0

A2 1

B B1 2

'''

print frame.ix['a', 1]['B']

'''

more

B1 2

'''

print frame.ix['a', 1]['A']['A1']

'''

0

'''

print

print '直接用MultiIndex创建层次索引结构index'

print MultiIndex.from\_arrays([['A', 'A', 'B'], ['Gree', 'Red', 'Green']],

names = ['state', 'color'])

'''

MultiIndex(levels=[[u'A', u'B'], [u'Gree', u'Green', u'Red']],

labels=[[0, 0, 1], [0, 2, 1]],

names=[u'state', u'color'])

'''

1. 将索引层进行交换：swaplevel函数。对某个索引层进行排序：sortlevel函数

代码示例：

print '索引层交换'

frame = DataFrame(numpy.arange(12).reshape((4, 3)),

index = [['a', 'a', 'b', 'b'], [1, 2, 1, 2]],

columns = [['A', 'A', 'B'], ['A1', 'A2', 'B1']])

frame.index.names = ['key1', 'key2']

print frame

'''

A B

A1 A2 B1

key1 key2

a 1 0 1 2

2 3 4 5

b 1 6 7 8

2 9 10 11

'''

frame\_swapped = frame.swaplevel('key1', 'key2') # 交互索引层

print frame\_swapped

'''

A B

A1 A2 B1

key2 key1

1 a 0 1 2

2 a 3 4 5

1 b 6 7 8

2 b 9 10 11

'''

print frame\_swapped.swaplevel(0, 1) # 交换回来

'''

A B

A1 A2 B1

key1 key2

a 1 0 1 2

2 3 4 5

b 1 6 7 8

2 9 10 11

'''

print

print '对某个索引层进行排序'

print frame.sortlevel('key2')

'''

A B

A1 A2 B1

key1 key2

a 1 0 1 2

b 1 6 7 8

a 2 3 4 5

b 2 9 10 11

'''

print frame.swaplevel(0, 1).sortlevel(0)

'''

A B

A1 A2 B1

key2 key1

1 a 0 1 2

b 6 7 8

2 a 3 4 5

b 9 10 11

'''

1. 根据某个索引层进行统计计算

代码示例：

print '根据索引层进行统计'

print frame

'''

A B

A1 A2 B1

key1 key2

a 1 0 1 2

2 3 4 5

b 1 6 7 8

2 9 10 11

'''

print frame.sum(level = 'key2')

'''

A B

A1 A2 B1

key2

1 6 8 10

2 12 14 16

'''

1. 将某列转化为层次的行索引，列名为索引名，列的值为索引值：set\_index函数；恢复重置行索引且恢复列：reset\_index函数。

代码示例：

print '将列索引转化行层次索引'

frame = DataFrame({'a':range(7),

'b':range(7, 0, -1),

'c':['one', 'one', 'one', 'two', 'two', 'two', 'two'],

'd':[0, 1, 2, 0, 1, 2, 3]})

print frame

'''

a b c d

0 0 7 one 0

1 1 6 one 1

2 2 5 one 2

3 3 4 two 0

4 4 3 two 1

5 5 2 two 2

6 6 1 two 3

'''

print frame.set\_index(['c', 'd']) # 把c/d列索引变成行索引

'''

a b

c d

one 0 0 7

1 1 6

2 2 5

two 0 3 4

1 4 3

2 5 2

3 6 1

'''

print frame.set\_index(['c', 'd'], drop = False) # 列依然保留

'''

a b c d

c d

one 0 0 7 one 0

1 1 6 one 1

2 2 5 one 2

two 0 3 4 two 0

1 4 3 two 1

2 5 2 two 2

3 6 1 two 3

'''

frame2 = frame.set\_index(['c', 'd'])

print frame2.reset\_index() # 恢复列

'''

c d a b

0 one 0 0 7

1 one 1 1 6

2 one 2 2 5

3 two 0 3 4

4 two 1 4 3

5 two 2 5 2

6 two 3 6 1

'''

**七、整数型索引值**

Series/DataFrame的索引值的类型为整数时，使用数组索引会产生歧义：无法分清是数组类型索引还是字典类型索引。整数型索引的Series/DataFrame索引的方法：以iloc索引替代数组索引。

代码示例：

print '索引值为整数时的歧义'

ser = Series(numpy.arange(3))

print ser

'''

0 0

1 1

2 2

'''

try:

print ser[-1] # 这里会有歧义.

except:

print 'exception'

ser2 = Series(numpy.arange(3), index = ['a', 'b', 'c'])

print ser2[-1] # 索引值类型不是整数

# 2

ser3 = Series(range(3), index = [-5, 1, 3])

print ser3.iloc[2] # 使用iloc避免直接用[2]产生的歧义

# 2

print

print '对DataFrame使用整数索引'

frame = DataFrame(numpy.arange(6).reshape((3, 2)), index = [2, 0, 1])

print frame

'''

0 1

2 0 1

0 2 3

1 4 5

'''

print frame.iloc[0]

'''

0 4

1 5

'''

# print frame[2] 有歧义则会发生异常错误

# print frame['2'] 不存在'2'该索引

print frame.iloc[:, 1]

'''

2 1

0 3

1 5

'''