量化分析师的Python日记【第5天：数据处理的瑞士军刀pandas】

207 克隆 29 赞 ☆25 收藏

发布于2015-03-11 13:58:06

Python数据处理的瑞士军刀：pandas

####第一篇：基本数据结构介绍

####一、Pandas介绍

终于写到了作者最想介绍，同时也是Python在数据处理方面功能最为强大的扩展模块了。在处理实际的金融数据时，一个条数据通常包含了多种类型的数据，例如，股票的代码是字符串，收盘价是浮点型，而成交量是整型等。在C++中可以实现为一个给定结构体作为单元的容器，如向量（vector，C++中的特定数据结构）。在Python中，pandas包含了高级的数据结构Series和DataFrame，使得在Python中处理数据变得非常方便、快速和简单。

pandas不同的版本之间存在一些不兼容性，为此，我们需要清楚使用的是哪一个版本的pandas。现在我们就查看一下量化实验室的pandas版本：

```
1 import pandas as pd
2 pd.__version__
```

查看全部 ()

```
'0.14.1'
```

pandas主要的两个数据结构是Series和DataFrame，随后两节将介绍如何由其他类型的数据结构得到这两种数据结构，或者自行创建这两种数据结构，我们先导入它们以及相关模块：

```
1 import numpy as np
2 from pandas import Series, DataFrame
```

查看全部 ()

####二、Pandas数据结构：Series

从一般意义上来讲，Series可以简单地被认为是一维的数组。Series和一维数组最主要的区别在于Series类型具有索引（index），可以和另一个编程中常见的数据结构哈希（Hash）联系起来。

#####2.1 创建Series

创建一个Series的基本格式是s = Series(data, index=index, name=name)，以下给出几个创建Series的例子。首先我们从数组创建Series：

```
1 a = np.random.randn(5)
2 print "a is an array:"
```

```
print a
s = Series(a)
print "s is a Series:"
print s
```

查看全部 ()

```
a is an array:
[-1.24962807 -0.85316907  0.13032511 -0.19088881  0.40475505]
s is a Series:
0   -1.249628
1   -0.853169
2    0.130325
3   -0.190889
4    0.404755
dtype: float64
```

可以在创建Series时添加index，并可使用Series.index查看具体的index。需要注意的一点是，当从数组创建Series时，若指定index，那么index长度要和data的长度一致：

```
s = Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
print s
s.index
```

查看全部 ()

```
a    0.509906
b   -0.764549
c    0.919338
d   -0.084712
e    1.896407
dtype: float64
Index([u'a', u'b', u'c', u'd', u'e'], dtype='object')
```

创建Series的另一个可选项是name，可指定Series的名称，可用Series.name访问。在随后的DataFrame中，每一列的列名在该列被单独取出来时就成了Series的名称：

```
s = Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'], name='my_series')
print s
print s.name
```

查看全部 ()

```
a   -1.898245
b    0.172835
c    0.779262
d    0.289468
e   -0.947995
Name: my_series, dtype: float64
```

```
my_series
```

Series还可以从字典（dict）创建：

```
d = {'a': 0., 'b': 1, 'c': 2}
print "d is a dict:"
print d
s = Series(d)
print "s is a Series:"
print s
```

查看全部 ()

```
d is a dict:
{'a': 0.0, 'c': 2, 'b': 1}
s is a Series:
a    0
b    1
c    2
dtype: float64
```

让我们来看看使用字典创建Series时指定index的情形（index长度不必和字典相同）：

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

查看全部 ()

```
b     1
c     2
d   NaN
a     0
dtype: float64
```

我们可以观察到两点：一是字典创建的Series，数据将按index的顺序重新排列；二是index长度可以和字典长度不一致，如果多了的话，pandas将自动为多余的index分配NaN（not a number，pandas中数据缺失的标准记号)，当然index少的话就截取部分的字典内容。

如果数据就是一个单一的变量，如数字4，那么Series将重复这个变量：

```
Series(4., index=['a', 'b', 'c', 'd', 'e'])
```

查看全部 ()

```
a    4
b    4
c    4
d    4
e    4
```

```
dtype: float64
```

#####2.2 Series数据的访问

访问Series数据可以和数组一样使用下标，也可以像字典一样使用索引，还可以使用一些条件过滤：

```
1 s = Series(np.random.randn(10),index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'])
2 s[0]
```

查看全部 ()

```
1.4328106520571824
```

```
1 s[:2]
```

查看全部 ()

```
a    1.432811
b    0.120681
dtype: float64
```

```
1 s[[2,0,4]]
```

查看全部 ()

```
c    0.578146
a    1.432811
e    1.327594
dtype: float64
```

```
1 s[['e', 'i']]
```

查看全部 ()

```
e    1.327594
i   -0.634347
dtype: float64
```

```
1 s[s > 0.5]
```

查看全部 ()

```
a    1.432811
c    0.578146
e    1.327594
g    1.850783
dtype: float64
```

```
1 'e' in s
```

查看全部 ()

```
True
```

####三、Pandas数据结构：DataFrame

在使用DataFrame之前，我们说明一下DataFrame的特性。DataFrame是将数个Series按列合并而成的二维数据结构，每一列单独取出来是一个Series，这和SQL数据库中取出的数据是很类似的。所以，按列对一个DataFrame进行处理更为方便，用户在编程时注意培养按列构建数据的思维。DataFrame的优势在于可以方便地处理不同类型的列，因此，就不要考虑如何对一个全是浮点数的DataFrame求逆之类的问题了，处理这种问题还是把数据存成NumPy的matrix类型比较便利一些。

#####3.1 创建DataFrame

首先来看如何从字典创建DataFrame。DataFrame是一个二维的数据结构，是多个Series的集合体。我们先创建一个值是Series的字典，并转换为DataFrame：

```
1 d = {'one': Series([1., 2., 3.], index=['a', 'b', 'c']), 'two': Series([1., 2., 3., 4.],
  index=['a', 'b', 'c', 'd'])}
2 df = DataFrame(d)
3 print df
```

查看全部 ()

```
   one  two
a    1    1
b    2    2
c    3    3
d  NaN    4
```

可以指定所需的行和列，若字典中不含有对应的元素，则置为NaN：

```
1 df = DataFrame(d, index=['r', 'd', 'a'], columns=['two', 'three'])
2 print df
```

查看全部 ()

```
   two three
r  NaN   NaN
d    4   NaN
a    1   NaN
```

可以使用dataframe.index和dataframe.columns来查看DataFrame的行和列，dataframe.values则以数组的形式返回DataFrame的元素：

```
1 print "DataFrame index:"
2 print df.index
3 print "DataFrame columns:"
4 print df.columns
```

```
print "DataFrame values:"
print df.values
```

查看全部 ()

```
DataFrame index:
Index([u'alpha', u'beta', u'gamma', u'delta', u'eta'], dtype='object')
DataFrame columns:
Index([u'a', u'b', u'c', u'd', u'e'], dtype='object')
DataFrame values:
[[  0.   0.   0.   0.   0.]
 [  1.   2.   3.   4.   5.]
 [  2.   4.   6.   8.  10.]
 [  3.   6.   9.  12.  15.]
 [  4.   8.  12.  16.  20.]]
```

DataFrame也可以从值是数组的字典创建，但是各个数组的长度需要相同：

```
d = {'one': [1., 2., 3., 4.], 'two': [4., 3., 2., 1.]}
df = DataFrame(d, index=['a', 'b', 'c', 'd'])
print df
```

查看全部 ()

```
   one  two
a    1    4
b    2    3
c    3    2
d    4    1
```

值非数组时，没有这一限制，并且缺失值补成NaN：

```
d= [{'a': 1.6, 'b': 2}, {'a': 3, 'b': 6, 'c': 9}]
df = DataFrame(d)
print df
```

查看全部 ()

```
     a  b    c
0  1.6  2  NaN
1  3.0  6    9
```

在实际处理数据时，有时需要创建一个空的DataFrame，可以这么做：

```
df = DataFrame()
print df
```

查看全部 ()

```
Empty DataFrame
Columns: []
Index: []
```

另一种创建DataFrame的方法十分有用，那就是使用concat函数基于Serie或者DataFrame创建一个DataFrame

```
1 a = Series(range(5))
2 b = Series(np.linspace(4, 20, 5))
3 df = pd.concat([a, b], axis=1)
4 print df
```

查看全部 ()

```
   0   1
0  0   4
1  1   8
2  2  12
3  3  16
4  4  20
```

其中的axis=1表示按列进行合并，axis=0表示按行合并，并且，Series都处理成一列，所以这里如果选axis=0的话，将得到一个10×1的DataFrame。下面这个例子展示了如何按行合并DataFrame成一个大的DataFrame：

```
1 df = DataFrame()
2 index = ['alpha', 'beta', 'gamma', 'delta', 'eta']
3 for i in range(5):
4     a = DataFrame([np.linspace(i, 5*i, 5)], index=[index[i]])
5     df = pd.concat([df, a], axis=0)
6 print df
```

查看全部 ()

```
        0  1   2   3   4
alpha   0  0   0   0   0
beta    1  2   3   4   5
gamma   2  4   6   8  10
delta   3  6   9  12  15
eta     4  8  12  16  20
```

#####3.2 DataFrame数据的访问

首先，再次强调一下DataFrame是以列作为操作的基础的，全部操作都想象成先从DataFrame里取一列，再从这个Series取元素即可。可以用datafrae.column_name选取列，也可以使用dataframe[]操作选取列，我们可以马上发现前一种方法只能选取一列，而后一种方法可以选择多列。若DataFrame没有列名，[]可以使用非负整数，也就是“下标”选取列；若有列名，则必须使用列名选取，另外datafrae.column_name在没有列名的时候是无效的：

```
print df[1]
print type(df[1])
df.columns = ['a', 'b', 'c', 'd', 'e']
print df['b']
print type(df['b'])
print df.b
print type(df.b)
print df[['a', 'd']]
print type(df[['a', 'd']])
```

查看全部 ()

```
alpha    0
beta     2
gamma    4
delta    6
eta      8
Name: 1, dtype: float64
<class 'pandas.core.series.Series'>
alpha    0
beta     2
gamma    4
delta    6
eta      8
Name: b, dtype: float64
<class 'pandas.core.series.Series'>
alpha    0
beta     2
gamma    4
delta    6
eta      8
Name: b, dtype: float64
<class 'pandas.core.series.Series'>
       a   d
alpha  0   0
beta   1   4
gamma  2   8
delta  3  12
eta    4  16
<class 'pandas.core.frame.DataFrame'>
```

以上代码使用了dataframe.columns为DataFrame赋列名，并且我们看到单独取一列出来，其数据结构显示的是Series，取两列及两列以上的结果仍然是DataFrame。访问特定的元素可以如Series一样使用下标或者是索引:

```
print df['b'][2]
print df['b']['gamma']
```

查看全部 ()

```
4.0
4.0
```

若需要选取行，可以使用dataframe.iloc按下标选取，或者使用dataframe.loc按索引选取：

```
print df.iloc[1]
print df.loc['beta']
```

查看全部 ()

```
a    1
b    2
c    3
d    4
e    5
Name: beta, dtype: float64
a    1
b    2
c    3
d    4
e    5
Name: beta, dtype: float64
```

选取行还可以使用切片的方式或者是布尔类型的向量：

```
print "Selecting by slices:"
print df[1:3]
bool_vec = [True, False, True, True, False]
print "Selecting by boolean vector:"
print df[bool_vec]
```

查看全部 ()

```
Selecting by slices:
       a  b  c  d   e
beta   1  2  3  4   5
gamma  2  4  6  8  10
Selecting by boolean vector:
       a  b  c   d   e
alpha  0  0  0   0   0
gamma  2  4  6   8  10
delta  3  6  9  12  15
```

行列组合起来选取数据：

```
print df[['b', 'd']].iloc[[1, 3]]
print df.iloc[[1, 3]][['b', 'd']]
```

```
print df[['b', 'd']].loc[['beta', 'delta']]
print df.loc[['beta', 'delta']][['b', 'd']]
```

查看全部 ()

```
       b   d
beta   2   4
delta  6  12
       b   d
beta   2   4
delta  6  12
       b   d
beta   2   4
delta  6  12
       b   d
beta   2   4
delta  6  12
```

如果不是需要访问特定行列，而只是某个特殊位置的元素的话，dataframe.at和dataframe.iat是最快的方式，它们分别用于使用索引和下标进行访问：

```
print df.iat[2, 3]
print df.at['gamma', 'd']
```

查看全部 ()

```
8.0
8.0
```

dataframe.ix可以混合使用索引和下标进行访问，唯一需要注意的地方是行列内部需要一致，不可以同时使用索引和标签访问行或者列，不然的话，将会得到意外的结果：

```
print df.ix['gamma', 4]
print df.ix[['delta', 'gamma'], [1, 4]]
print df.ix[[1, 2], ['b', 'e']]
print "Unwanted result:"
print df.ix[['beta', 2], ['b', 'e']]
print df.ix[[1, 2], ['b', 4]]
```

查看全部 ()

```
10.0
       b   e
delta  6  15
gamma  4  10
       b   e
beta   2   5
gamma  4  10
Unwanted result:
```

```
        b   e
beta    2   5
2     NaN NaN
        b   4
beta    2 NaN
gamma   4 NaN
```

####参考文献

1. http://pandas.pydata.org/pandas-docs/version/0.14.1 (http://pandas.pydata.org/pandas-docs/version/0.14.1)

薛昆Kelvin (community/user/54806783f9f06c8e773366fd/shares) 关注

优矿 001 号员工

- 量化分析师的Python日记【第1天：谁来给我讲讲Python？】 (community/share/54c89443f9f06c27…
- 量化分析师的Python日记【第14天：如何在优矿上做Alpha对冲模型】 (community/share/55e662f9f…
- 量化分析师的Python日记【第3天：一大波金融Library来袭之numpy篇】 (community/share/54ca15f…
- 量化分析师的Python日记【第4天：一大波金融Library来袭之scipy篇】 (community/share/54d83bb…
- 量化分析师的Python日记【第5天：数据处理的瑞士军刀pandas】 (community/share/54ffd96ef9f06…

热门分享

- 量化分析师的Python日记【第1天：谁来给我讲讲Python？】 (community/share/54c89443f9f06c27…
- 优矿社区导读——量化投资门派梳理 (community/share/56749fca228e5bab38c977d1)
- 钟摆理论的简单实现——完美躲过股灾和精准抄底 (community/share/562cdabef9f06c4ca72fb6f8)
- 羊驼策略 (community/share/554b46aaf9f06cb1e2915294)
- 互联网+量化投资 大数据指数手把手 (community/share/55263359f9f06c8f3390457b)