Pandas (http://pandas.pydata.org/)是python的一个数据分析包，最初由AQR Capital Management (http://www.aqr.com/)于2008年4月开发，并于2009年底开源出来，目前由专注于Python数据包开发的PyData (http://pydata.org/)开发team继续开发和维护，属于PyData项目的一部分。Pandas最初被作为金融数据分析工具而开发出来，因此，pandas为时间序列分析提供了很好的支持。 Pandas的名称来自于面板数据（panel data）和python数据分析（data analysis）。panel data是经济学中关于多维数据集的一个术语，在Pandas中也提供了panel的数据类型。 这篇文章会介绍一些Pandas的基本知识，偷了些懒其中采用的例子大部分会来自官方的10分钟学Pandas (http://pandas.pydata.org/pandas-docs/stable/10min.html)。我会加上个人的理解，帮助大家记忆和学习。

Pandas中的数据结构

Series：一维数组，与Numpy中的一维array类似。二者与Python基本的数据结构List也很相近，其区别是：List中的元素可以是不同的数据类型，而Array和Series中则只允许存储相同的数据类型，这样可以更有效的使用内存，提高运算效率。

Time- Series：以时间为索引的Series。

DataFrame：二维的表格型数据结构。很多功能与R中的data.frame类似。可以将DataFrame理解为Series的容器。以下的内容主要以DataFrame为主。

Panel ：三维的数组，可以理解为DataFrame的容器。<!-- more -->

创建DataFrame

首先引入Pandas及Numpy：

```
import pandas as pd
import numpy as np
```

官方推荐的缩写形式为pd，你可以选择其他任意的名称。 DataFrame是二维的数据结构，其本质是Series的容器，因此，DataFrame可以包含一个索引以及与这些索引联合在一起的Series，由于一个Series中的数据类型是相同的，而不同Series的数据结构可以不同。因此对于DataFrame来说，每一列的数据结构都是相同的，而不同的列之间则可以是不同的数据结构。或者以数据库进行类比，DataFrame中的每一行是一个记录，名称为Index的一个元素，而每一列则为一个字段，是这个记录的一个属性。 创建DataFrame有多种方式：

- 以字典的字典或Series的字典的结构构建DataFrame，这时候的最外面字典对应的是DataFrame的列，内嵌的字典及Series则是其中每个值。

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

可以看到d是一个字典，其中one的值为Series有3个值，而two为Series有4个值。由d构建的为一个4行2列的DataFrame。其中one只有3个值，因此d行one列为NaN（Not a Number）--Pandas默认的缺失值标记。

- 从列表的字典构建DataFrame，其中嵌套的每个列表（List）代表的是一个列，字典的名字则是列标签。这里要注意的是每个列表中的元素数量应该相同。否则会报错：

```
ValueError: arrays must all be same length
```

- 从字典的列表构建DataFrame，其中每个字典代表的是每条记录（DataFrame中的一行），字典中每个值对应的是这条记录的相关属性。

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

以上的语句与以Series的字典形式创建的DataFrame相同，只是思路略有不同，一个是以列为单位构建，将所有记录的不同属性转化为多个Series，行标签冗余，另一个是以行为单位构建，将每条记录转化为一个字典，列标签冗余。使用这种方式，如果不通过columns指定列的顺序，那么列的顺序会是随机的。

个人经验是对于从一些已经结构化的数据转化为DataFrame似乎前者更方便，而对于一些需要自己结构化的数据（比如解析Log文件，特别是针对较大数据量时），似乎后者更方便。创建了DataFrame后可以通过index.name属性为DataFrame的索引指定名称。

DataFrame转换为其他类型

```
df.to_dict(outtype='dict')
```

outtype的参数为‘dict’、‘list’、‘series’和‘records’。 dict返回的是dict of dict；list返回的是列表的字典；series返回的是序列的字典；records返回的是字典的列表

查看数据

head和tail方法可以显示DataFrame前N条和后N条记录，N为对应的参数，默认值为5。这通常是拿到DataFrame后的第一个命令，可以方便的了解数据内容和含义。

```
df.head()
```

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

4 rows × 2 columns

R中的对应函数：

```
head(df)
```

```
df.tail()
```

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

4 rows × 2 columns

index（行）和columns（列）属性，可以获得DataFrame的行和列的标签。这也是了解数据内容和含义的重要步骤。

```
df.index
```

```
Index([u'a', u'b', u'c', u'd'], dtype='object')
```

查看字段名

```
df.columns
```

```
Index([u'one', u'two'], dtype='object')
```

decribe方法可以计算各个列的基本描述统计值。包含计数，平均数，标准差，最大值，最小值及4分位差。

```
df.describe()
```

	one	two
count	3.0	4.000000
mean	2.0	2.500000
std	1.0	1.290994
min	1.0	1.000000
25%	1.5	1.750000
50%	2.0	2.500000
75%	2.5	3.250000
max	3.0	4.000000

8 rows × 2 columns

R中的对应函数：

```
summary(df)
```

行列转置

```
df.T
```

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

2 rows × 4 columns

排序

DataFrame提供了多种排序方式。

```
df.sort_index(axis=1, ascending=False)
```

sort_index可以以轴的标签进行排序。axis是指用于排序的轴，可选的值有0和1，默认为0即行标签（Y轴），1为按照列标签排序。 ascending是排序方式，默认为True即降序排列。

```
df.sort(columns='two')
df.sort(columns=['one','two'],ascending=[0,1])
```

DataFrame也提供按照指定列进行排序，可以仅指定一个列作为排序标准（以单独列名作为columns的参数），也可以进行多重排序（columns的参数为一个列名的List，列名的出现顺序决定排序中的优先级），在多重排序中ascending参数也为一个List，分别与columns中的List元素对应。

读写数据

DataFrame可以方便的读写数据文件，最常见的文件为CSV或Excel。Pandas读写Excel文件需要openpyxl (http://pythonhosted.org/openpyxl/)（Excel 2007），xlrd/xlwt (http://www.python-excel.org/)（Excel 2003）。

从CSV中读取数据：

```
df = pd.read_csv('foo.csv')
```

R中的对应函数：

```
df = read.csv('foo.csv')
```

将DataFrame写入CSV：

```
df.to_csv('foo.csv')
```

R中的对应函数：

```
df.to.csv('foo.csv')
```

从Excel中读取数据：

```
xls = ExcelFile('foo.xlsx')
xls.parse('sheet1', index_col=None, na_values=['NA'])
```

先定义一个Excel文件，用xls.parse解析sheet1的内容，index_col用于指定index列，na_values定义缺失值的标识。

将DataFrame写入Excel文件：

```
df.to_excel('foo.xlsx', sheet_name='sheet1')
```

默认的sheet为sheet1，也可以指定其他sheet名。

数据切片

通过下标选取数据：

```
df['one']
df.one
```

以上两个语句是等效的，都是返回df名称为one列的数据，返回的为一个Series。

```
df[0:3]
df[0]
```

下标索引选取的是DataFrame的记录，与List相同DataFrame的下标也是从0开始，区间索引的话，为一个左闭右开的区间，即[0：3]选取的为1-3三条记录。与此等价，还可以用起始的索引名称和结束索引名称选取数据：

```
df['a':'b']
```

有一点需要注意的是使用起始索引名称和结束索引名称时，也会包含结束索引的数据。以上两种方式返回的都是DataFrame。

使用标签选取数据：

```
df.loc[行标签,列标签]
df.loc['a':'b']#选取ab两行数据
df.loc[:,'one']#选取one列的数据
```

df.loc的第一个参数是行标签，第二个参数为列标签（可选参数，默认为所有列标签），两个参数既可以是列表也可以是单个字符，如果两个参数都为列表则返回的是DataFrame，否则，则为Series。

使用位置选取数据：

```
df.iloc[行位置,列位置]
df.iloc[1,1]#选取第二行，第二列的值，返回的为单个值
df.iloc[0,2],:]#选取第一行及第三行的数据
df.iloc[0:2,:]#选取第一行到第三行（不包含）的数据
df.iloc[:,1]#选取所有记录的第一列的值，返回的为一个Series
df.iloc[1,:]#选取第一行数据，返回的为一个Series
```

PS：loc为location的缩写，iloc则为integer & location的缩写

更广义的切片方式是使用.ix，它自动根据你给到的索引类型判断是使用位置还是标签进行切片

```
df.ix[1,1]
df.ix['a':'b']
```

通过逻辑指针进行数据切片：

```
df[逻辑条件]
df[df.one >= 2]#单个逻辑条件
df[(df.one >=1 ) & (df.one < 3) ]#多个逻辑条件组合
```

这种方式获得的数据切片都是DataFrame。

基本运算

Pandas支持基本的运算及向量化运算。

```
df.mean()#计算列的平均值，参数为轴，可选值为0或1.默认为0，即按照列运算
df.sum(1)#计算行的和
df.apply(lambda x: x.max() - x.min())#将一个函数应用到DataFrame的每一列，这里使用的是匿名lambda函数，与R中apply函数类似
```

设置索引

```
df.set_index('one')
```

重命名列

```
df.rename(columns={u'one':'1'}, inplace=True)
```

查看每个列的数据类型

```
df.dtypes
```

R中的对应函数：

```
str(df)
```

查看最大值/最小值

```
pd.Series.max()
pd.Series.idxmax()
```

重设索引

```
df.reset_index(inplace=True)
```

改变数据类型

```
df['A'].astype(float)
```

计算Series每个值的频率

```
df['A'].value_counts()
```

R的对应函数：

```
table(df['A'])
```

字符方法

pandas提供许多向量化的字符操作，你可以在str属性中找到它们

```
s.str.lower()
s.str.len()
s.str.contains(pattern)
```

DataFrame的合并

Contact：

```
ds = [{'one' : 4,'two':2},{'one' : 5,'two' : 3},{'one' : 6,'two' :
4},{'two' : 7,'three':10}]
dfs = pd.DataFrame(ds,index=['e','f','g','h'])
##构建一个新的DataFrame，dfs
df_t=pd.concat([df,dfs])#合并两个DataFrame
```

Merge（类似SQL中的Join操作）：

```
left = pd.DataFrame({'key': ['foo1', 'foo2'], 'lval': [1, 2]})
right = pd.DataFrame({'key': ['foo1', 'foo2'], 'rval': [4, 5]})
#构建了两个DataFrame
pd.merge(left, right, on='key')#按照key列将两个DataFrame join在一起
```

DataFrame中的Group by：

```
df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar','foo', 'bar',
'foo', 'foo'],
                   'B' : ['one', 'one', 'two', 'three','two', 'tw
o', 'one', 'three'],
                   'C' :randn(8), 'D' : randn(8)});
df.groupby('A').sum()#按照A列的值分组求和
df.groupby(['A','B']).sum()##按照A、B两列的值分组求和
```

对应R函数：

```
tapply()
```

在实际应用中，先定义groups，然后再对不同的指标指定不同计算方式。

```
groups = df.groupby('A')#按照A列的值分组求和
groups['B'].sum()##按照A列的值分组求B组和
groups['B'].count()##按照A列的值分组B组计数
```

默认会以groupby的值作为索引，如果不将这些值作为索引，则需要使用as_index=False

```
df.groupby(['A','B'], as_index=False).sum()
```

构建透视表

使用pivot_table和crosstab都可以创建数据透视表

```
df = pd.DataFrame({'A' : ['one', 'one', 'two', 'three'] * 3,'B' :
['A', 'B', 'C'] * 4,
                'C' : ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] *
2,
                'D' : np.random.randn(12), 'E' : np.random.randn(1
2)})
pd.pivot_table(df, values = 'D', rows = ['A', 'B'], cols = ['C'])#以
A、B为行标签，以C为列标签将D列的值汇总求和
pd.crosstab(rows = ['A', 'B'], cols = ['C'], values = 'D')#以A、B为行
标签，以C为列标签将D列的值汇总求和
```

时间序列分析

时间序列也是Pandas的一个特色。时间序列在Pandas中就是以Timestamp为索引的Series。

pandas提供to_datetime方法将代表时间的字符转化为Timestamp对象：

```
s = '2013-09-16 21:00:00'
ts = pd.to_datetime(s)
```

有时我们需要处理时区问题：

```
ts=pd.to_datetime(s,utc=True).tz_convert('Asia/Shanghai')
```

构建一个时间序列：

```
rng = pd.date_range('1/1/2012', periods=5, freq='M')
ts = pd.Series(randn(len(rng)), index=rng)
```

Pandas提供resample方法对时间序列的时间粒度进行调整：

```
ts_h=ts.resample('H', how='count')#M,5Min,1s
```

以上是将时间序列调整为小时，还可以支持月（M），分钟（Min）甚至秒（s）等。

画图

Pandas也支持一定的绘图功能，需要安装matplot模块。

比如前面创建的时间序列，通过plot()就可以绘制出折线图，也可以使用hist()命令绘制频率分布的直方图。

关于Panda作图，请查看另一篇博文：用Pandas作图 (http://cloga.info/python/2014/02/23/Plotting_with_Pandas/)

以上是关于Pandas的简单介绍，其实除了Pandas之外，Python还提供了多个科学计算包，比如Numpy，Scipy，以及数据挖掘的包：Scikit Learn，Orage，NLTK等，感兴趣的同学可以了解一下。