Pandas 基础简介

（source：《Python金融大数据分析》——第六章 金融时间序列）

1. 使用DataFream类的第一步

首先导入：

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

创建DataFrame对象：

```
In [3]: df = pd.DataFrame([10, 20 ,30, 40],columns=['numbers'],index=["a", 'b', 'c', 'd'])

In [4]: df
Out[4]:
   numbers
a       10
b       20
c       30
d       40
```

> *数据*:
>
> 数据本身可以用不同组成及类型提供（列表、元组、narray和字典对象都是候选者）
>
> *标签*:
>
> 数据组织为列，可以自定义列名
>
> *索引*:
>
> 索引可采用不同的格式。

常见操作方式：

```
In [5]: df.index # 查看index的值
Out[5]: Index(['a', 'b', 'c', 'd'], dtype='object')

In [6]: df.columns  # 查看columns名称
Out[6]: Index(['numbers'], dtype='object')

In [7]: df.ix['c']  # 通过index进行检索(目前使用.loc 方法进行index检索，.iloc方法进行位置检索)
C:\ProgramData\Anaconda3\Scripts\ipython:1: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
```

```
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
Out[7]:
numbers    30
Name: c, dtype: int64

In [8]: df.loc[['a', 'b']]  # 通过index进行多列检索
Out[8]:
   numbers
a       10
b       20

In [9]: df.loc[df.index[1:3]]  # 通过index Object检索(切片规则：左开右闭)
Out[9]:
   numbers
b       20
c       30

In [11]: df.sum()  # 所有列求和
Out[11]:
numbers    100
dtype: int64

In [13]: df.apply(lambda x:x**2)  # 对于每个元素求平方(使用隐函数表达式)
Out[13]:
   numbers
a      100
b      400
c      900
d     1600
```

通常可以在DataFrame对象上进行和Numpy ndarray对象相同的向量化操作：

```
In [15]: df **  2  # 对于每个元素求平方(使用向量化操作方法)
Out[15]:
   numbers
a      100
b      400
c      900
d     1600
```

注：上述两种方法均不会改变df变量的值

```
In [16]: df
Out[16]:
   numbers
a       10
b       20
c       30
d       40
```

在两个维度上同时扩增DataFrame对象是可能的：

```
In [17]: df['floats'] = (1.5, 2.5, 3.5, 4.5)  # 生成一个新列

In [18]: df
Out[18]:
   numbers  floats
a       10     1.5
b       20     2.5
c       30     3.5
d       40     4.5
```

也可以取整个DataFrame对象来定义一个新列，在这种情况下索引自动分配：

```
In [19]: df['names'] = pd.DataFrame(['Yves', 'Fuido', 'Felix', 'Francesc'],index=['d',
'a'
    ...: , 'b', 'c'])

In [20]: df
Out[20]:
   numbers  floats     names
a       10     1.5     Fuido
b       20     2.5     Felix
c       30     3.5  Francesc
d       40     4.5      Yves
```

增加一行数据(会导致索引发生变化)

```
In [21]: df.append({'numbers':100, 'floats': 5.75, "names":"Henry"}, ignore_index=True)  #
临时增加信息，df不会改变
Out[21]:
   numbers  floats     names
0       10    1.50     Fuido
1       20    2.50     Felix
2       30    3.50  Francesc
3       40    4.50      Yves
4      100    5.75     Henry
```

不改变索引的方式（并且对df取值进行变化）：

```
In [23]: df.append(pd.DataFrame({'numbers':100, 'floats': 5.75, "names":"Henry"}, index=['
    ...: z',]))
Out[23]:
   numbers  floats     names
a       10    1.50     Fuido
b       20    2.50     Felix
c       30    3.50  Francesc
d       40    4.50      Yves
z      100    5.75     Henry
```

Pandas处理缺漏的的信息：

例如添加新列，但是用不同的索引，使用`join`方法：

```
In [29]: df = df.join(pd.DataFrame([1,4, 9, 16, 25],index=['a', 'b', 'c', 'd', 'y'],
colum
    ...: ns=['squares', ]))

In [30]: df
Out[30]:
   numbers  floats     names  squares
a       10    1.50     Fuido      1.0
b       20    2.50     Felix      4.0
c       30    3.50  Francesc      9.0
d       40    4.50      Yves     16.0
z      100    5.75     Henry      NaN

```

pandas 默认只接受索引已经存在的值。 我们丢失了索引为 y 的值，在 索引位置 z 可以看到

NaN (也就是"不是一个数字")值。 为了保留这两个索引，我们可以提供一个附加参数，告诉

pandas 如何连接。 例子中的 bow="outer"表示使用两个 索引中所有值的并集:

尽管有丢失值，但大部分方法还是有效的:

```
In [37]: df[['numbers', 'squares']].mean()
Out[37]:
numbers    40.0
squares     7.5
dtype: float64

In [38]: df[['numbers', 'squares']].std()
Out[38]:
numbers    35.355339
squares     6.557439
dtype: float64
```

2. 使用DataFrame类的第二步

模拟数据集，生成一个9行4列的标准正态分布伪随机数（使用`numpy.ndarray`）：

```
In [39]: a = np.random.standard_normal((9, 4))

In [40]: a
Out[40]:
array([[ 0.38974586,  0.47384537,  1.86890137,  1.54942867],
       [ 0.87136206, -1.87414605, -2.12197595,  1.85668107],
       [-1.94217651, -0.73924633, -0.19380932, -0.62429293],
       [ 0.55394133, -0.54342501,  1.69892628, -0.96965967],
       [-0.44823896,  0.04674674,  0.25364913,  0.35342898],
       [ 1.18889508, -0.71665817, -0.49315751, -0.27119351],
       [ 1.26516132,  0.54755319,  1.00269772, -1.13059427],
       [-0.60650053, -0.26050869,  0.49611401,  0.05673249],
       [ 0.33265246,  0.2685178 , -1.18724769,  0.9033508 ]])

In [41]: a.round(6)
Out[41]:
array([[ 0.389746,  0.473845,  1.868901,  1.549429],
       [ 0.871362, -1.874146, -2.121976,  1.856681],
       [-1.942177, -0.739246, -0.193809, -0.624293],
       [ 0.553941, -0.543425,  1.698926, -0.96966 ],
       [-0.448239,  0.046747,  0.253649,  0.353429],
       [ 1.188895, -0.716658, -0.493158, -0.271194],
       [ 1.265161,  0.547553,  1.002698, -1.130594],
       [-0.606501, -0.260509,  0.496114,  0.056732],
       [ 0.332652,  0.268518, -1.187248,  0.903351]])
```

转为DataFrame格式：

```
In [43]: df = pd.DataFrame(a)

In [44]: df
Out[44]:
          0         1         2         3
0  0.389746  0.473845  1.868901  1.549429
1  0.871362 -1.874146 -2.121976  1.856681
2 -1.942177 -0.739246 -0.193809 -0.624293
3  0.553941 -0.543425  1.698926 -0.969660
4 -0.448239  0.046747  0.253649  0.353429
5  1.188895 -0.716658 -0.493158 -0.271194
6  1.265161  0.547553  1.002698 -1.130594
7 -0.606501 -0.260509  0.496114  0.056732
8  0.332652  0.268518 -1.187248  0.903351
```

DataFrame的相关函数参数：

表 6-1　DataFrame 函数参数

参数	格式	描述
data	ndarray/字典/DataFrame	DataFrame 数据；字典可以包含序列、ndarray 和列表
index	索引/类似数组	使用的索引；默认为 range(n)
columns	索引/类似数组	使用的列标题；默认为 range(n)
dtype	dtype，默认 None	使用/强制的数据类型；否则通过推导得出
copy	布尔值，默认 None	从输入拷贝数据

改变对应的列指标属性：

```
In [45]: df.columns=[['No1', 'No2', 'No3', "No4"]]

In [46]: df
Out[46]:
        No1       No2       No3       No4
0  0.389746  0.473845  1.868901  1.549429
1  0.871362 -1.874146 -2.121976  1.856681
2 -1.942177 -0.739246 -0.193809 -0.624293
3  0.553941 -0.543425  1.698926 -0.969660
4 -0.448239  0.046747  0.253649  0.353429
5  1.188895 -0.716658 -0.493158 -0.271194
6  1.265161  0.547553  1.002698 -1.130594
7 -0.606501 -0.260509  0.496114  0.056732
8  0.332652  0.268518 -1.187248  0.903351
```

可以使用Pandas生成时间索引，利用`date_range`生成一个`DatetimeIndex`对象

```
In [56]: dates = pd.date_range('2015-1-1', periods=9, freq="m")

In [57]: dates
Out[57]:
DatetimeIndex(['2015-01-31', '2015-02-28', '2015-03-31', '2015-04-30',
               '2015-05-31', '2015-06-30', '2015-07-31', '2015-08-31',
               '2015-09-30'],
              dtype='datetime64[ns]', freq='M')
```

`date_range`函数参数：

表 6-2　date_range 函数参数

参数	格式	描述
start	字符串/日期时间	生成日期的左界
end	字符串/日期时间	生成日期的右界
periods	整数/None	期数（如果 start 或者 end 空缺）
freq	字符串/日期偏移	频率字符串，例如 5D（5 天）
tz	字符串/None	本地化索引的时区名称
normalize	布尔值，默认 None	将 start 和 end 规范化为午夜
name	字符串，默认 None	结果索引名称

将新生成的DatetimeIndex作为新的Index对象，赋值给DataFrame对象：

```
In [58]: df.index = dates

In [59]: df
Out[59]:
                 No1       No2       No3       No4
2015-01-31  0.389746  0.473845  1.868901  1.549429
2015-02-28  0.871362 -1.874146 -2.121976  1.856681
2015-03-31 -1.942177 -0.739246 -0.193809 -0.624293
2015-04-30  0.553941 -0.543425  1.698926 -0.969660
2015-05-31 -0.448239  0.046747  0.253649  0.353429
2015-06-30  1.188895 -0.716658 -0.493158 -0.271194
2015-07-31  1.265161  0.547553  1.002698 -1.130594
2015-08-31 -0.606501 -0.260509  0.496114  0.056732
2015-09-30  0.332652  0.268518 -1.187248  0.903351
```

补充：date_range函数频率参数值

表 6-3　date_range 函数频率参数值

别名	描述
B	交易日
C	自定义交易日（试验性）
D	日历日
W	每周
M	每月底
BM	每月最后一个交易日
MS	月初
BMS	每月第一个交易日
Q	季度末
BQ	每季度最后一个交易日
QS	季度初
BQS	每季度第一个交易日
A	每年底
BA	每年最后一个交易日
AS	每年初
BAS	每年第一个交易日
H	每小时
T	每分钟
S	每秒
L	毫秒
U	微秒

3. 基本分析方法

pandas DataFrame类提供内建方法：

```
In [59]: df  # 原始数据
Out[59]:
                 No1       No2       No3       No4
2015-01-31  0.389746  0.473845  1.868901  1.549429
2015-02-28  0.871362 -1.874146 -2.121976  1.856681
2015-03-31 -1.942177 -0.739246 -0.193809 -0.624293
2015-04-30  0.553941 -0.543425  1.698926 -0.969660
2015-05-31 -0.448239  0.046747  0.253649  0.353429
2015-06-30  1.188895 -0.716658 -0.493158 -0.271194
2015-07-31  1.265161  0.547553  1.002698 -1.130594
2015-08-31 -0.606501 -0.260509  0.496114  0.056732
2015-09-30  0.332652  0.268518 -1.187248  0.903351

In [60]: df.sum()  # 按列总和
Out[60]:
No1    1.604842
No2   -2.797321
No3    1.324098
No4    1.723882
dtype: float64

In [61]: df.mean()  # 按列均值
Out[61]:
No1    0.178316
No2   -0.310813
No3    0.147122
No4    0.191542
dtype: float64

In [62]: df.cumsum()  # 累计总和（前行累加）
Out[62]:
                 No1       No2       No3       No4
2015-01-31  0.389746  0.473845  1.868901  1.549429
2015-02-28  1.261108 -1.400301 -0.253075  3.406110
2015-03-31 -0.681069 -2.139547 -0.446884  2.781817
2015-04-30 -0.127127 -2.682972  1.252042  1.812157
2015-05-31 -0.575366 -2.636225  1.505691  2.165586
2015-06-30  0.613529 -3.352883  1.012534  1.894393
2015-07-31  1.878690 -2.805330  2.015232  0.763798
2015-08-31  1.272190 -3.065839  2.511346  0.820531
2015-09-30  1.604842 -2.797321  1.324098  1.723882

```

数值数据集统计数字的捷径方法：`describe()`

```
In [65]: df.describe()
Out[65]:
             No1       No2       No3       No4
count  9.000000  9.000000  9.000000  9.000000
mean   0.178316 -0.310813  0.147122  0.191542
std    1.024538  0.763615  1.308306  1.069097
min   -1.942177 -1.874146 -2.121976 -1.130594
25%   -0.448239 -0.716658 -0.493158 -0.624293
50%    0.389746 -0.260509  0.253649  0.056732
75%    0.871362  0.268518  1.002698  0.903351
max    1.265161  0.547553  1.868901  1.856681

```

DataFrame对象可应用大部分Numpy通用函数：

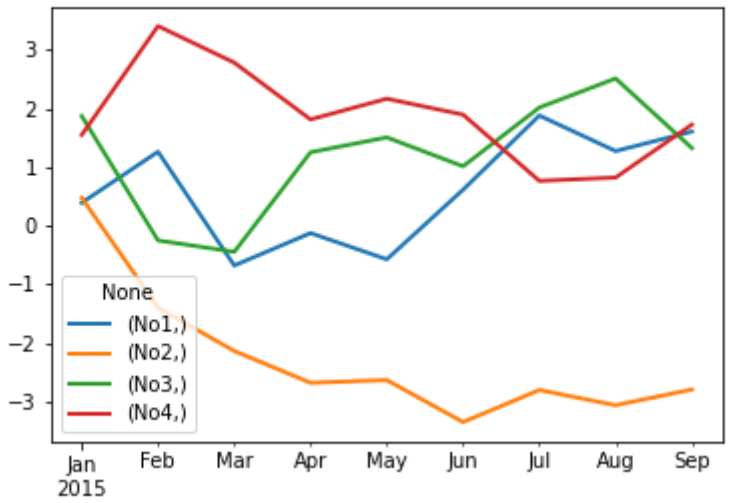
```
In [66]: np.sqrt(df)
C:\ProgramData\Anaconda3\Scripts\ipython:1: RuntimeWarning: invalid value encountered in
sqrt
Out[66]:
                 No1       No2       No3       No4
2015-01-31  0.624296  0.688364  1.367078  1.244760
2015-02-28  0.933468       NaN       NaN  1.362601
2015-03-31       NaN       NaN       NaN       NaN
2015-04-30  0.744272       NaN  1.303429       NaN
2015-05-31       NaN  0.216210  0.503636  0.594499
2015-06-30  1.090365       NaN       NaN       NaN
2015-07-31  1.124794  0.739968  1.001348       NaN
2015-08-31       NaN       NaN  0.704354  0.238186
2015-09-30  0.576760  0.518187       NaN  0.950448
```

不完整的数据集也可以进行数据统计（自动忽略）：

```
In [67]: np.sqrt(df).sum()
C:\ProgramData\Anaconda3\Scripts\ipython:1: RuntimeWarning: invalid value encountered in
sqrt
Out[67]:
No1    5.093955
No2    2.162730
No3    4.879844
No4    4.390494
dtype: float64
```

同时Pandas对matplotlib进行了封装：

```
In [74]: df.cumsum().plot(lw=2.0)
Out[74]: <AxesSubplot:>
```



注：plot方法参数

表 6-4　plot 方法参数

参数	格式	描述
x	标签/位置，默认 None	只在列值为 x 刻度时使用
y	标签/位置，默认 None	只在列值为 x 刻度时使用
subplots	布尔值，默认 False	子图中的绘图列
sharex	布尔值，默认 True	共用 x 轴
sharey	布尔值，默认 False	共用 y 轴
use_index	布尔值，默认 True	使用 DataFrame.index 作为 x 轴刻度
stacked	布尔值，默认 False	堆叠（只用于柱状图）
sort_columns	布尔值，默认 False	在绘图之前将列按字母顺序排列
title	字符串，默认 None	图表标题
grid	布尔值，默认 False	水平和垂直网格线
legend	布尔值，默认 True	标签图例
ax	matplotlib 轴对象	绘图使用的 matplotlib 轴对象
style	字符串或者列表/字典	绘图线形（每列）
kind	“line”/”bar”/”barh”/”kde”/”density”	图表类型
logx	布尔值，默认 False	x 轴的对数刻度
logy	布尔值，默认 False	y 轴的对数刻度

续表

参数	格式	描述
xticks	序列，默认 Index	x 轴刻度
yticks	序列，默认 Values	y 轴刻度
xlim	二元组，列表	x 轴界限
ylim	二元组，列表	y 轴界限
rot	整数，默认 None	旋转 x 刻度
secondary_y	布尔值/序列，默认 False	第二个 y 轴
mark_right	布尔值，默认 True	第二个 y 轴自动设置标签
colormap	字符串/颜色映射对象，默认 None	用于绘图的颜色映射
kwds	关键字	传递给 matplotlib 选项

4. Series类

在从DataFrame对象中单选一列时就得到一个Series类，DataFrame的主要方法也可用于Series对象。

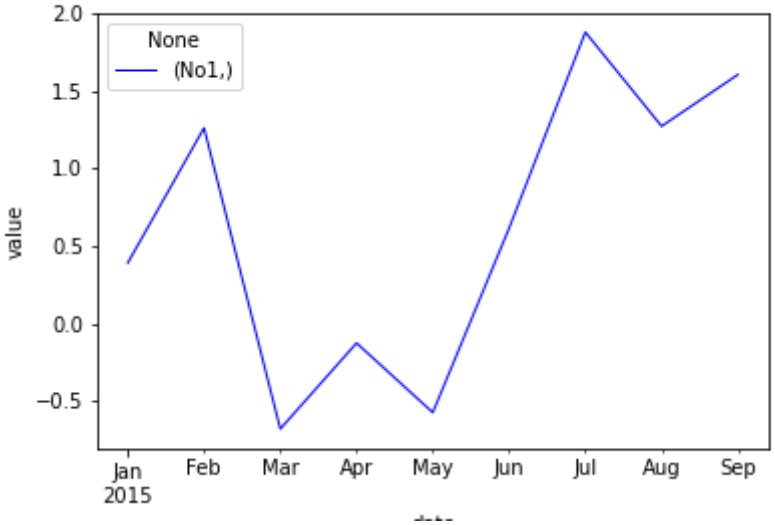
例如：

```
In [78]: df['No1'].cumsum().plot(style='b', lw=1)
Out[78]: <AxesSubplot:>

In [79]: plt.xlabel('date')
Out[79]: Text(0.5, 3.3999999999999986, 'date')

In [80]: plt.ylabel('value')
Out[80]: Text(18.625, 0.5, 'value')

In [81]: plt.show()
```



5. GroupBy操作

Pandas具有分组功能，例如：

首先添加一列季度数据，然后根据"Quarter"列分组：

```
In [82]: df["Quarter"]=["Q1", 'Q1', 'Q1', 'Q2', 'Q2', 'Q2', 'Q3', 'Q3', 'Q3']
    ...: df
Out[82]:
                 No1       No2       No3       No4 Quarter
2015-01-31  0.389746  0.473845  1.868901  1.549429      Q1
2015-02-28  0.871362 -1.874146 -2.121976  1.856681      Q1
2015-03-31 -1.942177 -0.739246 -0.193809 -0.624293      Q1
2015-04-30  0.553941 -0.543425  1.698926 -0.969660      Q2
2015-05-31 -0.448239  0.046747  0.253649  0.353429      Q2
2015-06-30  1.188895 -0.716658 -0.493158 -0.271194      Q2
2015-07-31  1.265161  0.547553  1.002698 -1.130594      Q3
2015-08-31 -0.606501 -0.260509  0.496114  0.056732      Q3
2015-09-30  0.332652  0.268518 -1.187248  0.903351      Q3
```