

来源于10分钟学习pandas，非常经典的教程，使用pandas通常会导入以下包，本环境使用的是python2.7

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

一、创建对象方法

1、通过传递一个list来创建一个Series，pandas会默认创建整型索引

In [2]:

```
s = pd.Series([5,4,np.nan,3,2,1])
```

In [3]:

```
s
```

Out[3]:

```
0    5.0
1    4.0
2    NaN
3    3.0
4    2.0
5    1.0
dtype: float64
```

2、传递一个numpy array，时间索引以及标签来创建一个DataFrame

In [4]:

```
dates = pd.date_range(start='20180821',end='20180826')
```

In [5]:

```
dates
```

Out[5]:

```
DatetimeIndex(['2018-08-21', '2018-08-22', '2018-08-23', '2018-08-24',
               '2018-08-25', '2018-08-26'],
              dtype='datetime64[ns]', freq='D')
```

In [6]:

```
dates = pd.date_range(start='20180821',periods=6)
```

In [7]:

```
dates
```

Out[7]:

```
DatetimeIndex(['2018-08-21', '2018-08-22', '2018-08-23', '2018-08-24',  
              '2018-08-25', '2018-08-26'],  
              dtype='datetime64[ns]', freq='D')
```

In [8]:

```
df = pd.DataFrame(np.random.randn(6,4),index = dates ,columns = list('ABCD'))
```

In [9]:

```
df
```

Out[9]:

	A	B	C	D
2018-08-21	1.199214	-0.005610	-0.968173	0.418314
2018-08-22	0.522341	0.795916	-0.576385	-0.014663
2018-08-23	1.664533	0.160990	0.212496	0.638565
2018-08-24	-1.933720	-0.229028	0.387028	0.890340
2018-08-25	0.790387	0.786251	-0.364618	-0.925611
2018-08-26	-1.118799	-0.104908	-2.210427	-0.767139

3、可以通过一个字典来创建dataframe，通常以列作为key

In [10]:

```
df2 = pd.DataFrame({'A':1,'B':'foo','C':np.array([3]*4,dtype='int32')})
```

In [11]:

```
df2
```

Out[11]:

	A	B	C
0	1	foo	3
1	1	foo	3
2	1	foo	3
3	1	foo	3

In [12]:

```
df2.dtypes
```

Out[12]:

```
A      int64
B      object
C      int32
dtype: object
```

二、查看数据

1、查看dataframe中头部和尾部的行

In [13]:

```
df.head()
```

Out[13]:

	A	B	C	D
2018-08-21	1.199214	-0.005610	-0.968173	0.418314
2018-08-22	0.522341	0.795916	-0.576385	-0.014663
2018-08-23	1.664533	0.160990	0.212496	0.638565
2018-08-24	-1.933720	-0.229028	0.387028	0.890340
2018-08-25	0.790387	0.786251	-0.364618	-0.925611

In [14]:

```
df.tail(3)
```

Out[14]:

	A	B	C	D
2018-08-24	-1.933720	-0.229028	0.387028	0.890340
2018-08-25	0.790387	0.786251	-0.364618	-0.925611
2018-08-26	-1.118799	-0.104908	-2.210427	-0.767139

2、查看索引、列和内部的numpy数据:

In [15]:

```
df.index
```

Out[15]:

```
DatetimeIndex(['2018-08-21', '2018-08-22', '2018-08-23', '2018-08-24',  
               '2018-08-25', '2018-08-26'],  
              dtype='datetime64[ns]', freq='D')
```

In [16]:

```
df.columns
```

Out[16]:

```
Index([u'A', u'B', u'C', u'D'], dtype='object')
```

In [17]:

```
df.values
```

Out[17]:

```
array([[ 1.19921415, -0.00560977, -0.96817293,  0.41831362],  
       [ 0.52234075,  0.79591573, -0.5763854 , -0.01466342],  
       [ 1.6645334 ,  0.1609904 ,  0.21249639,  0.63856502],  
       [-1.93372012, -0.22902838,  0.38702828,  0.89034042],  
       [ 0.79038668,  0.78625103, -0.36461796, -0.92561118],  
       [-1.11879889, -0.10490838, -2.21042675, -0.76713886]])
```

3、describe()函数可对数据快速统计汇总

In [18]:

```
df.describe()
```

Out[18]:

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	0.187326	0.233935	-0.586680	0.039968
std	1.406185	0.450051	0.940032	0.749846
min	-1.933720	-0.229028	-2.210427	-0.925611
25%	-0.708514	-0.080084	-0.870226	-0.579020
50%	0.656364	0.077690	-0.470502	0.201825
75%	1.097007	0.629936	0.068218	0.583502
max	1.664533	0.795916	0.387028	0.890340

4、对数据进行转置

In [19]:

```
df.T
```

Out[19]:

	2018-08-21 00:00:00	2018-08-22 00:00:00	2018-08-23 00:00:00	2018-08-24 00:00:00	2018-08-25 00:00:00	2018-08-26 00:00:00
A	1.199214	0.522341	1.664533	-1.933720	0.790387	-1.118799
B	-0.005610	0.795916	0.160990	-0.229028	0.786251	-0.104908
C	-0.968173	-0.576385	0.212496	0.387028	-0.364618	-2.210427
D	0.418314	-0.014663	0.638565	0.890340	-0.925611	-0.767139

5、对数据按轴排序，行为0，列为1

In [20]:

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

Out[20]:

	D	C	B	A
2018-08-21	0.418314	-0.968173	-0.005610	1.199214
2018-08-22	-0.014663	-0.576385	0.795916	0.522341
2018-08-23	0.638565	0.212496	0.160990	1.664533
2018-08-24	0.890340	0.387028	-0.229028	-1.933720
2018-08-25	-0.925611	-0.364618	0.786251	0.790387
2018-08-26	-0.767139	-2.210427	-0.104908	-1.118799

In [21]:

```
df.sort_index(axis=0,ascending=False)
```

Out[21]:

	A	B	C	D
2018-08-26	-1.118799	-0.104908	-2.210427	-0.767139
2018-08-25	0.790387	0.786251	-0.364618	-0.925611
2018-08-24	-1.933720	-0.229028	0.387028	0.890340
2018-08-23	1.664533	0.160990	0.212496	0.638565
2018-08-22	0.522341	0.795916	-0.576385	-0.014663
2018-08-21	1.199214	-0.005610	-0.968173	0.418314

6、按值排序，只能按照列排序

In [22]:

```
df.sort_values('B',ascending=False)
```

Out[22]:

	A	B	C	D
2018-08-22	0.522341	0.795916	-0.576385	-0.014663
2018-08-25	0.790387	0.786251	-0.364618	-0.925611
2018-08-23	1.664533	0.160990	0.212496	0.638565
2018-08-21	1.199214	-0.005610	-0.968173	0.418314
2018-08-26	-1.118799	-0.104908	-2.210427	-0.767139
2018-08-24	-1.933720	-0.229028	0.387028	0.890340

三、选择

获取数据

1、选择单独的一个列，这将会返回一个Series，等同于df.A

In [23]:

```
df.A
```

Out[23]:

```
2018-08-21    1.199214
2018-08-22    0.522341
2018-08-23    1.664533
2018-08-24   -1.933720
2018-08-25    0.790387
2018-08-26   -1.118799
Freq: D, Name: A, dtype: float64
```

In [24]:

```
df['A']
```

Out[24]:

```
2018-08-21    1.199214
2018-08-22    0.522341
2018-08-23    1.664533
2018-08-24   -1.933720
2018-08-25    0.790387
2018-08-26   -1.118799
Freq: D, Name: A, dtype: float64
```

2、通过 [] 选择，默认是对行进行切片，也就是选择行，注意，一个是能取到最后一个元素，一个不能取到最后元素。第一个没有语意，第二个有语意

In [25]:

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

Out[25]:

	A	B	C	D
2018-08-21	1.199214	-0.005610	-0.968173	0.418314
2018-08-22	0.522341	0.795916	-0.576385	-0.014663
2018-08-23	1.664533	0.160990	0.212496	0.638565

In [26]:

```
df['20180821':'20180823']
```

Out[26]:

	A	B	C	D
2018-08-21	1.199214	-0.005610	-0.968173	0.418314
2018-08-22	0.522341	0.795916	-0.576385	-0.014663
2018-08-23	1.664533	0.160990	0.212496	0.638565

通过标签选择

1、使用标签来获取一个交叉的区域,dataframe中loc是选行

In [27]:

```
df.loc[:]
```

Out[27]:

	A	B	C	D
2018-08-21	1.199214	-0.005610	-0.968173	0.418314
2018-08-22	0.522341	0.795916	-0.576385	-0.014663
2018-08-23	1.664533	0.160990	0.212496	0.638565
2018-08-24	-1.933720	-0.229028	0.387028	0.890340
2018-08-25	0.790387	0.786251	-0.364618	-0.925611
2018-08-26	-1.118799	-0.104908	-2.210427	-0.767139

In [28]:

```
df.loc['20180825']
```

Out[28]:

```
A    0.790387
B    0.786251
C   -0.364618
D   -0.925611
Name: 2018-08-25 00:00:00, dtype: float64
```

In [29]:

```
df.loc[dates[0]]
```

Out[29]:

```
A    1.199214
B   -0.005610
C   -0.968173
D    0.418314
Name: 2018-08-21 00:00:00, dtype: float64
```

2、通过标签来在多个轴上进行选则，非常好用

In [30]:

```
df.loc[:,['A','B']]
```

Out[30]:

	A	B
2018-08-21	1.199214	-0.005610
2018-08-22	0.522341	0.795916
2018-08-23	1.664533	0.160990
2018-08-24	-1.933720	-0.229028
2018-08-25	0.790387	0.786251
2018-08-26	-1.118799	-0.104908

3、标签切片

In [31]:

```
df.loc['20180821':'20180823',['C','D']]
```

Out[31]:

	C	D
2018-08-21	-0.968173	0.418314
2018-08-22	-0.576385	-0.014663
2018-08-23	0.212496	0.638565

4、对返回的对象进行维度缩减

In [32]:

```
df.loc['20180821',['C','D']]
```

Out[32]:

```
C    -0.968173
D      0.418314
Name: 2018-08-21 00:00:00, dtype: float64
```

5、获取一个标量

In [33]:

```
df.loc[dates[0],'A']
```

Out[33]:

```
1.199214149363109
```

6、快速访问一个标量

In [34]:

```
df.at[dates[0],'A']
```

Out[34]:

```
1.199214149363109
```

通过位置选择

1、通过传递数值进行位置选择（选择的是行）

In [35]:

```
df.iloc[1]
```

Out[35]:

```
A    0.522341
B    0.795916
C   -0.576385
D   -0.014663
Name: 2018-08-22 00:00:00, dtype: float64
```

2、通过数值进行切片，与numpy/python中情况类似

In [36]:

```
df.iloc[0:2,0:2]
```

Out[36]:

	A	B
2018-08-21	1.199214	-0.005610
2018-08-22	0.522341	0.795916

3、通过指定一个位置的列表，与numpy/python中的情况类似

In [37]:

```
df.iloc[[1,2,4],[0,2]]
```

Out[37]:

	A	C
2018-08-22	0.522341	-0.576385
2018-08-23	1.664533	0.212496
2018-08-25	0.790387	-0.364618

4、对行进行切片

In [38]:

```
df.iloc[1:3,:]
```

Out[38]:

	A	B	C	D
2018-08-22	0.522341	0.795916	-0.576385	-0.014663
2018-08-23	1.664533	0.160990	0.212496	0.638565

5、对列进行切片

In [39]:

```
df.iloc[:,1:3]
```

Out[39]:

	B	C
2018-08-21	-0.005610	-0.968173
2018-08-22	0.795916	-0.576385
2018-08-23	0.160990	0.212496
2018-08-24	-0.229028	0.387028
2018-08-25	0.786251	-0.364618
2018-08-26	-0.104908	-2.210427

6、获取特定的值

In [40]:

```
df.iloc[1,1]
```

Out[40]:

0.79591573356929468

In [41]:

```
df.iat[1,1]
```

Out[41]:

0.79591573356929468

布尔索引

1、使用单独的列来选择数据

In [42]:

```
df[df.A>0]
```

Out[42]:

	A	B	C	D
2018-08-21	1.199214	-0.005610	-0.968173	0.418314
2018-08-22	0.522341	0.795916	-0.576385	-0.014663
2018-08-23	1.664533	0.160990	0.212496	0.638565
2018-08-25	0.790387	0.786251	-0.364618	-0.925611

2、使用where操作来选择数据,将所有不满足条件的置为NaN

In [43]:

```
df[df>0]
```

Out[43]:

	A	B	C	D
2018-08-21	1.199214	NaN	NaN	0.418314
2018-08-22	0.522341	0.795916	NaN	NaN
2018-08-23	1.664533	0.160990	0.212496	0.638565
2018-08-24	NaN	NaN	0.387028	0.890340
2018-08-25	0.790387	0.786251	NaN	NaN
2018-08-26	NaN	NaN	NaN	NaN

3、使用isin()来过滤

In [44]:

```
df2 = df.copy()
```

In [45]:

```
df2['E']=['a','b','c','a','b','c']
```

In [46]:

```
df2
```

Out[46]:

	A	B	C	D	E
2018-08-21	1.199214	-0.005610	-0.968173	0.418314	a
2018-08-22	0.522341	0.795916	-0.576385	-0.014663	b
2018-08-23	1.664533	0.160990	0.212496	0.638565	c
2018-08-24	-1.933720	-0.229028	0.387028	0.890340	a
2018-08-25	0.790387	0.786251	-0.364618	-0.925611	b
2018-08-26	-1.118799	-0.104908	-2.210427	-0.767139	c

In [47]:

```
df2[df2['E'].isin(['a','b'])]
```

Out[47]:

	A	B	C	D	E
2018-08-21	1.199214	-0.005610	-0.968173	0.418314	a
2018-08-22	0.522341	0.795916	-0.576385	-0.014663	b
2018-08-24	-1.933720	-0.229028	0.387028	0.890340	a
2018-08-25	0.790387	0.786251	-0.364618	-0.925611	b

设置

1、设置一个新的列，按照行索引来合并

In [48]:

```
s1 = pd.Series([1,2,3,4,5,6],index=pd.date_range('20180821',periods=6))
```

In [49]:

```
df['F'] = s1
```

In [50]:

```
df
```

Out[50]:

	A	B	C	D	F
2018-08-21	1.199214	-0.005610	-0.968173	0.418314	1
2018-08-22	0.522341	0.795916	-0.576385	-0.014663	2
2018-08-23	1.664533	0.160990	0.212496	0.638565	3
2018-08-24	-1.933720	-0.229028	0.387028	0.890340	4
2018-08-25	0.790387	0.786251	-0.364618	-0.925611	5
2018-08-26	-1.118799	-0.104908	-2.210427	-0.767139	6

2、通过标签来设置新值

In [51]:

```
df.at[dates[0], 'A'] = 0
```

In [52]:

```
df
```

Out[52]:

	A	B	C	D	F
2018-08-21	0.000000	-0.005610	-0.968173	0.418314	1
2018-08-22	0.522341	0.795916	-0.576385	-0.014663	2
2018-08-23	1.664533	0.160990	0.212496	0.638565	3
2018-08-24	-1.933720	-0.229028	0.387028	0.890340	4
2018-08-25	0.790387	0.786251	-0.364618	-0.925611	5
2018-08-26	-1.118799	-0.104908	-2.210427	-0.767139	6

3、通过位置设置新值

In [53]:

```
df.at[dates[0], 'A'] = 1.0
```

In [54]:

```
df
```

Out[54]:

	A	B	C	D	F
2018-08-21	1.000000	-0.005610	-0.968173	0.418314	1
2018-08-22	0.522341	0.795916	-0.576385	-0.014663	2
2018-08-23	1.664533	0.160990	0.212496	0.638565	3
2018-08-24	-1.933720	-0.229028	0.387028	0.890340	4
2018-08-25	0.790387	0.786251	-0.364618	-0.925611	5
2018-08-26	-1.118799	-0.104908	-2.210427	-0.767139	6

In [55]:

```
df.iat[0,1]=0
```

In [56]:

```
df
```

Out[56]:

	A	B	C	D	F
2018-08-21	1.000000	0.000000	-0.968173	0.418314	1
2018-08-22	0.522341	0.795916	-0.576385	-0.014663	2
2018-08-23	1.664533	0.160990	0.212496	0.638565	3
2018-08-24	-1.933720	-0.229028	0.387028	0.890340	4
2018-08-25	0.790387	0.786251	-0.364618	-0.925611	5
2018-08-26	-1.118799	-0.104908	-2.210427	-0.767139	6

4、通过numpy数值设置一组新值

In [57]:

```
df.loc[:, 'D'] = np.array([5]*len(df))
```

In [58]:

```
df
```

Out[58]:

	A	B	C	D	F
2018-08-21	1.000000	0.000000	-0.968173	5	1
2018-08-22	0.522341	0.795916	-0.576385	5	2
2018-08-23	1.664533	0.160990	0.212496	5	3
2018-08-24	-1.933720	-0.229028	0.387028	5	4
2018-08-25	0.790387	0.786251	-0.364618	5	5
2018-08-26	-1.118799	-0.104908	-2.210427	5	6

5、通过where条件来设置新值

In [59]:

```
df2 = df.copy()
```

In [60]:

```
df2[df2>0] = -df2
```

In [61]:

```
df2
```

Out[61]:

	A	B	C	D	F
2018-08-21	-1.000000	0.000000	-0.968173	-5	-1
2018-08-22	-0.522341	-0.795916	-0.576385	-5	-2
2018-08-23	-1.664533	-0.160990	-0.212496	-5	-3
2018-08-24	-1.933720	-0.229028	-0.387028	-5	-4
2018-08-25	-0.790387	-0.786251	-0.364618	-5	-5
2018-08-26	-1.118799	-0.104908	-2.210427	-5	-6

In [62]:

```
-df2
```

Out[62]:

	A	B	C	D	F
2018-08-21	1.000000	-0.000000	0.968173	5.0	1.0
2018-08-22	0.522341	0.795916	0.576385	5.0	2.0
2018-08-23	1.664533	0.160990	0.212496	5.0	3.0
2018-08-24	1.933720	0.229028	0.387028	5.0	4.0
2018-08-25	0.790387	0.786251	0.364618	5.0	5.0
2018-08-26	1.118799	0.104908	2.210427	5.0	6.0

四、缺失值处理

在pandas中，使用np.nan来替代缺失值，这些值将默认不会包含在计算中

In [63]:

```
df.iat[0,2]=np.nan
```


In [64]:

```
df
```

Out[64]:

	A	B	C	D	F
2018-08-21	1.000000	0.000000	NaN	5	1
2018-08-22	0.522341	0.795916	-0.576385	5	2
2018-08-23	1.664533	0.160990	0.212496	5	3
2018-08-24	-1.933720	-0.229028	0.387028	5	4
2018-08-25	0.790387	0.786251	-0.364618	5	5
2018-08-26	-1.118799	-0.104908	-2.210427	5	6

1、reindex()方法可以对指定轴上的索引进行改变/增加/删除操作，并返回原始数据的一个拷贝

In [65]:

```
df1 = df.reindex(index=dates[0:4],columns=list(df.columns)+['E'])
```

In [66]:

```
df1.loc[dates[0]:dates[2], 'E']=1
```

In [67]:

```
df1
```

Out[67]:

	A	B	C	D	F	E
2018-08-21	1.000000	0.000000	NaN	5	1	1.0
2018-08-22	0.522341	0.795916	-0.576385	5	2	1.0
2018-08-23	1.664533	0.160990	0.212496	5	3	1.0
2018-08-24	-1.933720	-0.229028	0.387028	5	4	NaN

2、去掉包含缺失值的行

In [68]:

```
df1.dropna(how='any')
```

Out[68]:

	A	B	C	D	F	E
2018-08-22	0.522341	0.795916	-0.576385	5	2	1.0
2018-08-23	1.664533	0.160990	0.212496	5	3	1.0

3、对缺失值进行填充

In [69]:

```
df1.fillna(value=5)
```

Out[69]:

	A	B	C	D	F	E
2018-08-21	1.000000	0.000000	5.000000	5	1	1.0
2018-08-22	0.522341	0.795916	-0.576385	5	2	1.0
2018-08-23	1.664533	0.160990	0.212496	5	3	1.0
2018-08-24	-1.933720	-0.229028	0.387028	5	4	5.0

4、对数据进行布尔填充

In [70]:

```
pd.isnull(df1)
```

Out[70]:

	A	B	C	D	F	E
2018-08-21	False	False	True	False	False	False
2018-08-22	False	False	False	False	False	False
2018-08-23	False	False	False	False	False	False
2018-08-24	False	False	False	False	False	True

五、相关操作

1、执行描述性统计，默认按列统计

In [71]:

```
df.mean()
```

Out[71]:

```
A    0.154124
B    0.234870
C   -0.510381
D    5.000000
F    3.500000
dtype: float64
```

2、在其他轴上统计

In [72]:

```
df.mean(axis=1)
```

Out[72]:

```
2018-08-21    1.750000
2018-08-22    1.548374
2018-08-23    2.007604
2018-08-24    1.444856
2018-08-25    2.242404
2018-08-26    1.513173
Freq: D, dtype: float64
```

3、对于拥有不同维度，需要对齐的对象进行操作，pandas会自动的沿着指定的维度进行广播

In [73]:

```
dates
```

Out[73]:

```
DatetimeIndex(['2018-08-21', '2018-08-22', '2018-08-23', '2018-08-24',
               '2018-08-25', '2018-08-26'],
              dtype='datetime64[ns]', freq='D')
```

In [74]:

```
s = pd.Series([1,2,3,np.nan,5,6],index=dates).shift(2)
```

In [75]:

```
s
```

Out[75]:

```
2018-08-21    NaN
2018-08-22    NaN
2018-08-23    1.0
2018-08-24    2.0
2018-08-25    3.0
2018-08-26    NaN
Freq: D, dtype: float64
```

Apply

1、对数据应用函数，默认是对列操作

In [76]:

```
df
```

Out[76]:

	A	B	C	D	F
2018-08-21	1.000000	0.000000	NaN	5	1
2018-08-22	0.522341	0.795916	-0.576385	5	2
2018-08-23	1.664533	0.160990	0.212496	5	3
2018-08-24	-1.933720	-0.229028	0.387028	5	4
2018-08-25	0.790387	0.786251	-0.364618	5	5
2018-08-26	-1.118799	-0.104908	-2.210427	5	6

In [77]:

```
df.apply(np.cumsum)
```

Out[77]:

	A	B	C	D	F
2018-08-21	1.000000	0.000000	NaN	5	1
2018-08-22	1.522341	0.795916	-0.576385	10	3
2018-08-23	3.186874	0.956906	-0.363889	15	6
2018-08-24	1.253154	0.727878	0.023139	20	10
2018-08-25	2.043541	1.514129	-0.341479	25	15
2018-08-26	0.924742	1.409220	-2.551905	30	21

In [78]:

```
df.apply(lambda x: x.max()-x.min())
```

Out[78]:

```
A    3.598254
B    1.024944
C    2.597455
D    0.000000
F    5.000000
dtype: float64
```

直方图

In [79]:

```
s = pd.Series(np.random.randint(0,7,size=10))
```

In [80]:

```
s
```

Out[80]:

```
0    1
1    2
2    5
3    2
4    0
5    1
6    2
7    5
8    4
9    3
dtype: int64
```

In [81]:

```
s.value_counts()
```

Out[81]:

```
2    3
5    2
1    2
4    1
3    1
0    1
dtype: int64
```

字符串方法，Series对象在其str属性中包含大量字符串处理方法，可以很容易应用到数组中的每一个元素

In [82]:

```
s = pd.Series(['A','B',np.nan,'cat','Aaba'])
```

In [83]:

```
s.str.lower()
```

Out[83]:

```
0      a
1      b
2     NaN
3     cat
4     aaba
dtype: object
```

六、合并

pandas提供大量的方法能够轻松对Series，DataFrame和Panel对象进行各种符号逻辑关系的合并操作

In [84]:

```
df = pd.DataFrame(np.random.randn(10,4))
```

In [85]:

```
df
```

Out[85]:

	0	1	2	3
0	1.002085	-0.913897	-2.109344	-0.342600
1	-1.550781	-0.814409	0.277290	0.014148
2	-1.390003	-0.678331	-0.075444	-2.099395
3	0.482018	0.642555	-0.963368	-0.368277
4	-0.027364	0.076906	-0.642373	-0.070943
5	0.311142	0.228270	0.159489	0.090280
6	0.951503	0.211551	0.000848	-0.475100
7	-0.175005	0.323663	-2.272497	0.159793
8	1.049522	0.075842	-0.813033	1.335427
9	-1.040715	0.403349	2.251924	0.702020

In [86]:

```
pieces = [df[:3],df[3:7],df[7:]]
```

In [87]:

```
pieces
```

Out[87]:

```
[          0          1          2          3
0  1.002085 -0.913897 -2.109344 -0.342600
1 -1.550781 -0.814409  0.277290  0.014148
2 -1.390003 -0.678331 -0.075444 -2.099395,
          0          1          2          3
3  0.482018  0.642555 -0.963368 -0.368277
4 -0.027364  0.076906 -0.642373 -0.070943
5  0.311142  0.228270  0.159489  0.090280
6  0.951503  0.211551  0.000848 -0.475100,
          0          1          2          3
7 -0.175005  0.323663 -2.272497  0.159793
8  1.049522  0.075842 -0.813033  1.335427
9 -1.040715  0.403349  2.251924  0.702020]
```

In [88]:

```
pd.concat(pieces)
```

Out[88]:

	0	1	2	3
0	1.002085	-0.913897	-2.109344	-0.342600
1	-1.550781	-0.814409	0.277290	0.014148
2	-1.390003	-0.678331	-0.075444	-2.099395
3	0.482018	0.642555	-0.963368	-0.368277
4	-0.027364	0.076906	-0.642373	-0.070943
5	0.311142	0.228270	0.159489	0.090280
6	0.951503	0.211551	0.000848	-0.475100
7	-0.175005	0.323663	-2.272497	0.159793
8	1.049522	0.075842	-0.813033	1.335427
9	-1.040715	0.403349	2.251924	0.702020

Join 类似于SQL类型的合并

In [89]:

```
left = pd.DataFrame({'key':['foo','foo'],'lval':[1,2]})
```

In [90]:

```
right = pd.DataFrame({'key':['foo','foo'],'rval':[4,5]})
```

In [91]:

```
left
```

Out[91]:

	key	lval
0	foo	1
1	foo	2

In [92]:

```
right
```

Out[92]:

	key	rval
0	foo	4
1	foo	5

In [93]:

```
pd.merge(left,right,on='key')
```

Out[93]:

	key	lval	rval
0	foo	1	4
1	foo	1	5
2	foo	2	4
3	foo	2	5

Append将一行连接到一个DataFrame上

In [94]:

```
df = pd.DataFrame(np.random.randn(8,4),columns=['A','B','C','D'])
```

In [95]:

```
df
```

Out[95]:

	A	B	C	D
0	-0.427729	-0.372866	-1.436830	-0.449695
1	-2.458164	0.697945	-0.032415	-0.935662
2	0.191171	0.839163	1.989867	-0.730836
3	-1.028197	1.277604	-0.766794	-0.640668
4	-0.734925	0.105059	0.401019	0.126799
5	-0.809884	-2.105929	0.223554	0.667745
6	1.176203	0.377354	-1.546621	0.159540
7	-0.637126	0.564601	0.548597	-0.163806

In [96]:

```
s = df.iloc[3]
```

In [97]:

```
s
```

Out[97]:

```
A    -1.028197
B     1.277604
C    -0.766794
D    -0.640668
Name: 3, dtype: float64
```

In [98]:

```
df.append(s, ignore_index=True)
```

Out[98]:

	A	B	C	D
0	-0.427729	-0.372866	-1.436830	-0.449695
1	-2.458164	0.697945	-0.032415	-0.935662
2	0.191171	0.839163	1.989867	-0.730836
3	-1.028197	1.277604	-0.766794	-0.640668
4	-0.734925	0.105059	0.401019	0.126799
5	-0.809884	-2.105929	0.223554	0.667745
6	1.176203	0.377354	-1.546621	0.159540
7	-0.637126	0.564601	0.548597	-0.163806
8	-1.028197	1.277604	-0.766794	-0.640668

七、分组

对于"groupby by" 操作，我们通常是指以下一个或多个操作步骤：

- (1)、Splitting 按照一些规则将数据分为不同的组
- (2)、Applying 对于每组数据分别执行一个函数
- (3)、Combining 将结果组合到一个数据结构中

In [99]:

```
df = pd.DataFrame({
    'A': ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar'],
    'B': ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
    'C': np.random.randn(8)})
```

In [100]:

```
df
```

Out[100]:

	A	B	C
0	foo	one	-1.417086
1	bar	one	0.307996
2	foo	two	-0.183475
3	bar	three	-1.212871
4	foo	two	1.153401
5	bar	two	-0.057243
6	foo	one	0.452743
7	bar	three	0.336198

1、分组并对每个分组执行sum函数：

In [101]:

```
df.groupby('A').sum()
```

Out[101]:

	C
A	
bar	-0.625920
foo	0.005583

2、通过多个列进行分组形成一个层次索引，然后执行函数

In [102]:

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

Out[102]:

		C
A	B	
bar	one	0.307996
	three	-0.876673
	two	-0.057243
foo	one	-0.964343
	two	0.969927

八、Reshaping

Stack

In [103]:

```
tuples = list(zip(*[['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'], ['one', 'two', 'one', 'two', 'one', 'two']]))
```

In [104]:

```
tuples
```

Out[104]:

```
[('bar', 'one'),  
 ('bar', 'two'),  
 ('baz', 'one'),  
 ('baz', 'two'),  
 ('foo', 'one'),  
 ('foo', 'two'),  
 ('qux', 'one'),  
 ('qux', 'two')]
```

In [105]:

```
index = pd.MultiIndex.from_tuples(tuples, names=['first', 'second'])
```

In [106]:

```
index
```

Out[106]:

```
MultiIndex(levels=[['bar', 'baz', 'foo', 'qux'], ['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two']],  
            labels=[[0, 0, 1, 1, 2, 2, 3, 3], [0, 1, 0, 1, 0, 1, 0, 1]],  
            names=[u'first', u'second'])
```

In [107]:

```
df = pd.DataFrame(np.random.randn(8,2),index=index,columns=['A','B'])
```

In [108]:

```
df
```

Out[108]:

		A	B
first	second		
bar	one	0.365162	0.794545
	two	0.343488	0.009913
baz	one	0.746508	0.380457
	two	-0.579208	0.504321
foo	one	0.493576	-0.086843
	two	0.944458	1.922707
qux	one	1.808426	0.734766
	two	1.004492	-1.118476

In [109]:

```
df2 = df[:4]
```

In [110]:

```
df2
```

Out[110]:

		A	B
first	second		
bar	one	0.365162	0.794545
	two	0.343488	0.009913
baz	one	0.746508	0.380457
	two	-0.579208	0.504321

In [111]:

```
stacked = df2.stack()
```

In [112]:

```
stacked
```

Out[112]:

```
first  second
bar    one    A    0.365162
          B    0.794545
          two    A    0.343488
          B    0.009913
baz    one    A    0.746508
          B    0.380457
          two    A   -0.579208
          B    0.504321
dtype: float64
```

In [113]:

```
type(stacked)
```

Out[113]:

```
pandas.core.series.Series
```

In [114]:

```
stacked.unstack()
```

Out[114]:

		A	B
first	second		
bar	one	0.365162	0.794545
	two	0.343488	0.009913
baz	one	0.746508	0.380457
	two	-0.579208	0.504321

九、时间序列

pandas 在对频率转化进行重新采样时拥有简单、强大且高效的功能（如将按秒采样的数据转化为按5分钟为单位采样的数据）。这种操作在金融非常常见

In [115]:

```
rng = pd.date_range('1/1/2012', periods=100, freq='S')
```

In [116]:

```
ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)
```

In [118]:

```
new_ts = ts.resample('5Min').sum()
```

In [119]:

```
new_ts
```

Out[119]:

```
2012-01-01    25954
Freq: 5T, dtype: int64
```

1、时区表示:

In [120]:

```
rng = pd.date_range('3/6/2012 00:00',periods=5,freq='D')
```

In [121]:

```
rng
```

Out[121]:

```
DatetimeIndex(['2012-03-06', '2012-03-07', '2012-03-08', '2012-03-09',
               '2012-03-10'],
              dtype='datetime64[ns]', freq='D')
```

In [122]:

```
ts = pd.Series(np.random.randn(len(rng)),rng)
```

In [123]:

```
ts
```

Out[123]:

```
2012-03-06    0.078082
2012-03-07   -1.027249
2012-03-08   -0.463756
2012-03-09   -0.218023
2012-03-10   -0.886765
Freq: D, dtype: float64
```

In [124]:

```
ts_utc = ts.tz_localize('UTC')
```

In [125]:

```
ts_utc
```

Out[125]:

```
2012-03-06 00:00:00+00:00    0.078082
2012-03-07 00:00:00+00:00   -1.027249
2012-03-08 00:00:00+00:00   -0.463756
2012-03-09 00:00:00+00:00   -0.218023
2012-03-10 00:00:00+00:00   -0.886765
Freq: D, dtype: float64
```

2、时区转换

In [126]:

```
ts_utc.tz_convert('US/Eastern')
```

Out[126]:

```
2012-03-05 19:00:00-05:00    0.078082
2012-03-06 19:00:00-05:00   -1.027249
2012-03-07 19:00:00-05:00   -0.463756
2012-03-08 19:00:00-05:00   -0.218023
2012-03-09 19:00:00-05:00   -0.886765
Freq: D, dtype: float64
```

3、时区跨度转换

In [127]:

```
rng = pd.date_range('1/1/2012', periods = 5, freq='M')
```

In [128]:

```
ts = pd.Series(np.random.randn(len(rng)), rng)
```

In [129]:

```
ts
```

Out[129]:

```
2012-01-31    -0.632020
2012-02-29    -1.209838
2012-03-31    -0.122362
2012-04-30     0.894776
2012-05-31     0.761503
Freq: M, dtype: float64
```

In [132]:

```
ps = ts.to_period()
```

In [133]:

```
ps.to_timestamp()
```

Out[133]:

```
2012-01-01    -0.632020
2012-02-01    -1.209838
2012-03-01    -0.122362
2012-04-01     0.894776
2012-05-01     0.761503
Freq: MS, dtype: float64
```

十、Categorical

从0.15版本开始, pandas可以在DataFrame中支持Categorical类型的数据

In [134]:

```
df = pd.DataFrame({'id':[1,2,3,4,5,6], 'raw_grade':['a','b','b','a','a','e']})
```

In [135]:

```
df
```

Out[135]:

	id	raw_grade
0	1	a
1	2	b
2	3	b
3	4	a
4	5	a
5	6	e

In [137]:

```
df['grade'] = df['raw_grade'].astype('category')
```

In [138]:

```
df
```

Out[138]:

	id	raw_grade	grade
0	1	a	a
1	2	b	b
2	3	b	b
3	4	a	a
4	5	a	a
5	6	e	e

In [139]:

```
df['grade']
```

Out[139]:

```
0    a
1    b
2    b
3    a
4    a
5    e
Name: grade, dtype: category
Categories (3, object): [a, b, e]
```


十一、画图

In [140]:

```
ts = pd.Series(np.random.randn(1000), index = pd.date_range('1/1/2000', periods = 1000))
```

In [142]:

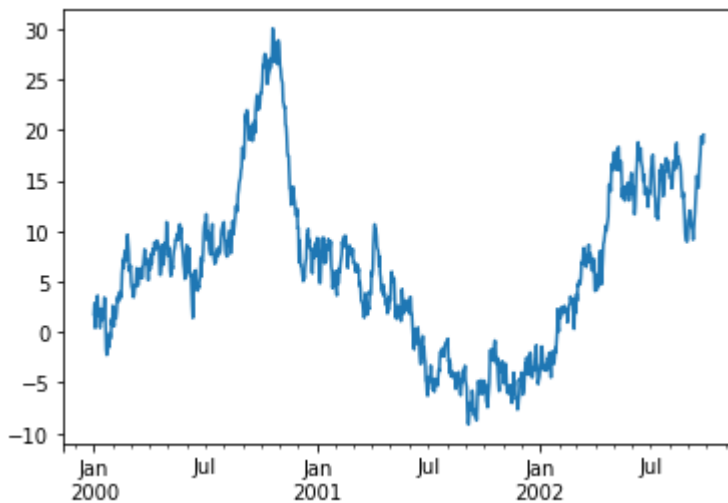
```
ts = ts.cumsum()
```

In [143]:

```
ts.plot()
```

Out[143]:

<matplotlib.axes._subplots.AxesSubplot at 0x111bba290>



对于DataFrame来说，plot是一种将所有列及标签进行绘制的简单方法：

In [145]:

```
df = pd.DataFrame(np.random.randn(1000,4), index=ts.index, columns=['A','B','C','D'])
```

In [146]:

```
df = df.cumsum()
```

In [147]:

```
df
```

Out[147]:

	A	B	C	D
2000-01-01	1.182622	-1.550789	-0.677847	-0.686247
2000-01-02	1.064192	0.058998	-0.926929	-0.256416
2000-01-03	1.120760	0.891475	-1.964664	0.237981
2000-01-04	2.203324	1.725893	-2.221881	-0.749801
2000-01-05	3.944393	1.265695	0.393940	-0.167811
2000-01-06	2.399475	2.747915	0.555871	-0.183643
2000-01-07	0.670621	3.962892	1.102575	-1.352560
2000-01-08	1.465290	2.604249	1.579050	-1.321723
2000-01-09	1.294411	2.098209	0.674846	-3.023043
2000-01-10	1.465836	2.268938	2.082882	-3.323533
2000-01-11	2.338271	3.229924	2.178011	-2.746837
2000-01-12	2.554280	3.337275	2.150721	-3.557612
2000-01-13	2.276484	2.299232	2.262648	-3.365707
2000-01-14	3.168337	3.041009	0.350698	-2.250190
2000-01-15	1.525096	3.872797	-0.036254	-1.685966
2000-01-16	0.204583	2.966500	-0.697475	-1.276493
2000-01-17	0.941978	2.338194	-0.697146	-0.050248
2000-01-18	0.151462	2.419511	-0.675964	-0.360681
2000-01-19	1.075320	2.897466	-0.717112	-0.295414
2000-01-20	0.108721	4.623138	-0.396870	-0.593116
2000-01-21	0.278526	4.626399	-1.058917	-1.079679
2000-01-22	-0.948492	4.707695	-1.036280	-0.246729
2000-01-23	-0.064785	3.445049	-0.860540	-0.223205
2000-01-24	-0.246719	3.475956	-1.142722	0.653491
2000-01-25	1.738197	2.825108	-1.243337	-0.405911
2000-01-26	2.355221	3.775056	-0.717280	2.108960
2000-01-27	2.666638	4.653226	-2.569220	2.028796
2000-01-28	3.246871	2.444634	-4.415980	1.859823
2000-01-29	1.574620	3.112912	-6.070074	1.222045
2000-01-30	1.905904	3.323519	-4.376543	0.544732
...
2002-08-28	16.590329	-23.183809	-42.979534	17.495257
2002-08-29	18.748040	-21.339509	-41.624783	16.946337

	A	B	C	D
2002-08-30	18.969827	-21.417957	-41.010790	16.153179
2002-08-31	17.501288	-20.444164	-40.787909	15.817613
2002-09-01	18.714918	-19.415917	-38.090624	14.580079
2002-09-02	16.239028	-19.166933	-37.983269	13.628032
2002-09-03	17.973330	-18.595785	-36.221094	14.122780
2002-09-04	17.183539	-18.230061	-38.055282	14.617521
2002-09-05	17.086782	-18.204624	-38.465634	13.804284
2002-09-06	18.085360	-18.618970	-39.385083	14.198803
2002-09-07	19.356489	-18.544110	-38.282765	16.077343
2002-09-08	20.867905	-17.344181	-38.669479	15.479961
2002-09-09	20.070411	-17.935920	-36.995769	15.338326
2002-09-10	19.641738	-17.647254	-37.014692	13.090233
2002-09-11	20.854340	-18.002293	-38.034404	14.752444
2002-09-12	21.608106	-17.451519	-37.723300	16.058830
2002-09-13	21.323948	-17.570505	-38.135861	16.269580
2002-09-14	20.921055	-15.981014	-36.757520	14.884650
2002-09-15	19.302019	-15.394612	-35.996095	14.915762
2002-09-16	17.875576	-13.576830	-36.998255	15.529466
2002-09-17	18.136928	-13.133025	-36.448088	14.572156
2002-09-18	18.924019	-14.224540	-35.438267	14.303393
2002-09-19	18.130387	-15.279811	-35.708854	14.660981
2002-09-20	17.948707	-15.482400	-35.846138	15.424886
2002-09-21	19.101230	-16.332950	-34.618454	14.312085
2002-09-22	18.447349	-16.304084	-33.785271	12.726596
2002-09-23	16.891263	-15.852266	-34.254771	12.965776
2002-09-24	16.141712	-16.834251	-34.089030	11.887542
2002-09-25	16.383442	-17.704636	-35.004542	12.044976
2002-09-26	17.635426	-18.405983	-32.839260	11.741913

1000 rows × 4 columns

In [148]:

```
df.plot()
```

Out[148]:

<matplotlib.axes._subplots.AxesSubplot at 0x111c7e090>

