



# pandas



覃雄派

# 提纲

- pandas介绍
- pandas实例



pandas





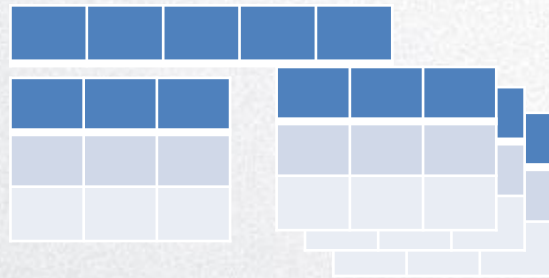
# pandas

- pandas介绍: Pandas是目前最流行的Python数据分析类库
  - Pandas → Python Data Analysis Library
- pandas是开源的(BSD-licensed)Python库, 提供易于使用的数据结构和数据分析工具
- 开发者对pandas进行了优化, 使得pandas的执行速度得到了保证
  - pandas基于numpy库开发, 和其它第三方库可以无缝地集成
    - pandas对时间序列分析提供了很好的支持, 它已经在金融数据分析领域中得到广泛应用
  - pandas可以处理如下不同类型的数据, 包括,
    - (1) **表格数据**: 表格的各个列, 可以具有不同的类型, 类似于SQL数据库的表格或者Excel电子表格
    - (2) **时间序列数据**: pandas支持有序、和无序的(Ordered/ unordered)时间序列数据的处理。时间序列数据无需是固定频率(Fixed-Frequency)的数据
    - (3) **矩阵**: pandas支持异构数据类型的矩阵, **可以设定行和列的标签**(Label, 即行、列的名称)
    - (4) 以及其它各类统计数据集(Statistical Dataset)

# pandas

- pandas介绍

- 为了表示、和分析现实世界中各类真实数据集，pandas提供了基本的数据结构
- pandas支持的数据结构，主要有，
  - (1) **Series**：一维数组，与Numpy中的一维数组array类似，二者与Python基本的数据结构List也很相近
    - 区别在于，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数据类型的元素，这样可以更有效的使用内存，提高运算效率
  - (2) **Time Series**：以时间为索引的Series
  - (3) **DataFrame**：二维的表格型数据结构
    - 可以将DataFrame理解为Series的容器
  - (4) **Panel**：三维数组
    - 可以理解为DataFrame的容器
- 这些数据结构，可以表示和处理金融、统计、社会科学、以及众多的工程领域的大多数应用场景用到的数据集







# pandas

- pandas的功能，包括，
  - (1) 处理数据的缺失值(Missing Data)，包括浮点数和非浮点数的缺失值
  - (2) 动态扩展性，用户可以插入或者删除DataFrame数据结构的列
  - (3)数据对齐(Data Alignment)：用户可以把数据对象对齐到标签(Label)，或者由Series、DataFrame等数据结构自行对齐
  - (3) 分组聚集功能(Group by and Aggregation)
  - (4) 把其它numpy等第三方库的数据结构转换成DataFrame的功能
  - (5) 数据转换(Transformation)
  - (6) 数据集的合并(Merging)和连接(Joining)



# pandas



- pandas的功能, 包括,
  - (7) 从CSV文件、Excel文件、数据库进行装载数据, 把数据保存到HDF5格式的文件, 以及从HDF5格式的文件装载数据
  - (8) 坐标轴的层次标签(Hierarchical Labeling)
  - (9) 数据透视表的旋转(Pivoting)、改变形状 (Reshaping)
  - (10) 对大数据集进行基于标签的(Label based)数据切片(Slicing)、提取子集(Sub Setting)、建立和使用索引(Fancy Indexing)
  - (11) pandas还提供面向时间序列数据处理的一些特殊功能, 比如时间频率转换、移动窗口上的统计值计算、移动窗口上的线性回归、序列的前移和后移(Shifting and Lagging)、生成数据范围比如生成时间范围(Date Range Generation)等

pandas





# pandas

- Pandas实例

- 创建Series
- 下面的实例，创建了一个Series，然后把它打印出来
- NaN表示Not a Number，即不是一个有效数值

标签

数据

0	1.0
1	3.0
2	5.0
3	NaN
4	6.0
5	8.0

dtype: float64



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

s = pd.Series([1, 3, 5, np.nan, 6, 8])
print (s)
```



# pandas

## • Pandas实例

- **创建DataFrame**
- 对于DataFrame来说，每一列的数据类型都是相同的
  - 而不同的列可以是不同的数据类型
  - 以SQL数据库表格进行类比，DataFrame中的每一行是一个记录，每一列则为一个字段，即记录的一个属性
- 下面的实例，首先创建了一个时间范围，然后基于这个时间范围创建了一个DataFrame，DataFrame的行标签即刚刚创建的时间范围，而列标签为A、B、C、D

行标签

列标签

	A	B	C	D
2013-01-01	0.124843	-0.724578	-0.848188	0.342412
2013-01-02	-2.182618	1.431102	0.769300	-0.431331
2013-01-03	0.385823	1.274385	0.057503	-0.315762
2013-01-04	-1.916555	0.747218	-0.091645	-0.787509
2013-01-05	-3.080642	0.280116	-0.250593	-0.214843
2013-01-06	-0.225550	0.451910	-0.152252	-2.027657

数据

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

dates = pd.date_range('20130101', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=['A', 'B', 'C', 'D'])
print (df)
```

# pandas

- Pandas实例

- 创建DataFrame

- 我们还可以以另外一种方式，创建DataFrame，输入的参数是一个字典，字典的每个key-value对的Value可以转换成一个Series

行标签

列标签

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

数据

```
df2 = pd.DataFrame({ 'A' : 1.,  
                      'B' : pd.Timestamp('20130102'),  
                      'C' : pd.Series(1,index=range(4),dtype='float32'),  
                      'D' : np.array([3] * 4,dtype='int32'),  
                      'E' : pd.Categorical(["test","train","test","train"]),  
                      'F' : 'foo' })  
print (df2)
```



# pandas

- Pandas实例
- 查看数据和元信息
  - **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
  - 通过**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
  - **describe**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分位差等
  - 通过**dtypes**属性，可以获得各列的数据类型
  - **values**属性则以列表的列表的方式，保存DataFrame的具体数据



# pandas

## • Pandas实例

### – 查看数据和元信息

```

      A      B      C      D      E
2013-01-01  0.471913  0.011896 -1.511653 -1.097910  one
2013-01-02  0.946843  1.889557  0.427803  1.376170  one
2013-01-03 -0.092205 -0.021862 -1.026872  0.699465  two
2013-01-04  0.469188  0.087677 -0.019453 -1.467458  three
2013-01-05  0.756042  0.066811 -0.344194  0.914210  four
2013-01-06 -0.053143 -0.126425 -0.982280  0.480497  three
      A      B      C      D      E
2013-01-02  0.946843  1.889557  0.427803  1.376170  one
2013-01-03 -0.092205 -0.021862 -1.026872  0.699465  two
2013-01-04  0.469188  0.087677 -0.019453 -1.467458  three
2013-01-05  0.756042  0.066811 -0.344194  0.914210  four
2013-01-06 -0.053143 -0.126425 -0.982280  0.480497  three
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
                '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')

```

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

```

A    float64
B    float64
C    float64
D    float64
E     object
dtype: object

```

```

[0.47191330459 0.011895751050175877 -1.51165343439111
 -1.0979096002092823  'one']
[0.9468430922633378  1.8895572949988484  0.4278033676304299
  1.376170268224617  'one']
[-0.0922054207728702 -0.021861692089023527 -1.0268718880930021
  0.6994647536811662  'two']
[0.46918812069005955  0.08767729454337776 -0.019452521535829985
 -1.4674580412818565  'three']
[0.756041963038969  0.06681086312248163 -0.34419439719104195
  0.9142099721088133  'four']
[-0.053143168966417326 -0.12642470013886903 -0.9822802693748913
  0.4804974830372463  'three']]

```

```

print (df.head())# 前5行
print (df.tail())# 后5行
df.describe()# 描述信息

```

```

print (df.index)# 行标签
print (df.columns)# 列标签
print (df.dtypes)# 各列的数据类型
print (df.values)# DataFrame的值

```

	A	B	C	D	E
2013-01-01	0.471913	0.011896	-1.511653	-1.097910	one
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-03	-0.092205	-0.021862	-1.026872	0.699465	two
2013-01-04	0.469188	0.087677	-0.019453	-1.467458	three
2013-01-05	0.756042	0.066811	-0.344194	0.914210	four
2013-01-06	-0.053143	-0.126425	-0.982280	0.480497	three

原Data frame



# pandas

## • Pandas实例

### – 转置与排序

- 对数据进行转置，就是把DataFrame的行变成列，列变成行，比如原来的二维表是6行3列，那么转置以后的二维表是3行6列

```
print (df.T)
```

	2013-01-01	2013-01-02	2013-01-03	2013-01-04	2013-01-05	2013-01-06
A	0. 471913	0. 946843	-0. 092205	0. 469188	0. 756042	-0. 053143
B	0. 011896	1. 889557	-0. 021862	0. 087677	0. 066811	-0. 126425
C	-1. 511653	0. 427803	-1. 026872	-0. 019453	-0. 344194	-0. 98228
D	-1. 09791	1. 37617	0. 699465	-1. 467458	0. 91421	0. 480497
E	one	one	two	three	four	three

	A	B	C	D	E
2013-01-01	0. 471913	0. 011896	-1. 511653	-1. 097910	one
2013-01-02	0. 946843	1. 889557	0. 427803	1. 376170	one
2013-01-03	-0. 092205	-0. 021862	-1. 026872	0. 699465	two
2013-01-04	0. 469188	0. 087677	-0. 019453	-1. 467458	three
2013-01-05	0. 756042	0. 066811	-0. 344194	0. 914210	four
2013-01-06	-0. 053143	-0. 126425	-0. 982280	0. 480497	three

原Data frame

# pandas

- Pandas实例

- 标签排序

- 可以对DataFrame按照坐标轴进行排序

- 下面的代码，对DataFrame按照第1个坐标轴进行排序，就是沿着列方向，对各个列标签即 Column Name进行排序；按照第0个坐标轴进行排序，是沿着行方向，对行标签进行排序

```
df = df.sort_index(axis=1, ascending=False) #沿着列方向对col name进行降序排序
print (df)
```

	E	D	C	B	A
2013-01-01	one	1.097910	1.511653	0.011896	0.471913
2013-01-02	one	1.376170	0.427803	1.889557	0.946843
2013-01-03	two	0.699465	-1.026872	-0.021862	-0.092205
2013-01-04	three	-1.467458	-0.019453	0.087677	0.469188
2013-01-05	four	0.914210	-0.344194	0.066811	0.756042
2013-01-06	three	0.480497	-0.982280	-0.126425	-0.053143






# pandas

- Pandas实例

- 数据排序
- 我们还可以基于某个列，对DataFrame进行排序
- 下面的代码，对DataFrame根据B数据列进行排序

```
df = df.sort_values(by='B')  
print (df)
```

	E	D	C	B	A
2013-01-06	three	0.480497	-0.982280	-0.126425	-0.053143
2013-01-03	two	0.699465	-1.026872	-0.021862	-0.092205
2013-01-01	one	-1.097910	-1.511653	0.011896	0.471913
2013-01-05	four	0.914210	-0.344194	0.066811	0.756042
2013-01-04	three	-1.467458	-0.019453	0.087677	0.469188
2013-01-02	one	1.376170	0.427803	1.889557	0.946843



pandas




# pandas

- Pandas实例
  - 提取部分数据
  - 可以通过列名，提取一个数据列

```
df = df.sort_index(axis=1, ascending=True)
df = df.sort_index(axis=0, ascending=True)
print(df)

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



2013-01-01	0.471913
2013-01-02	0.946843
2013-01-03	-0.092205
2013-01-04	0.469188
2013-01-05	0.756042
2013-01-06	-0.053143

Name: A, dtype: float64

	A	B	C	D	E
2013-01-01	0.471913	0.011896	-1.511653	-1.097910	one
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-03	-0.092205	-0.021862	-1.026872	0.699465	two
2013-01-04	0.469188	0.087677	-0.019453	-1.467458	three
2013-01-05	0.756042	0.066811	-0.344194	0.914210	four
2013-01-06	-0.053143	-0.126425	-0.982280	0.480497	three



# pandas

## • Pandas实例

- **提取部分数据**
- 可以通过行号范围，行标签范围，提取若干数据行，示例代码如下
- 在这里需要注意，0:3为下标范围，表示提取下标为0、1、2的行，而'20130102':'20130104'为行标签范围，表示提取'20130102'、'20130103'、'20130104'三行

行下标/列下标  
包头不包尾

行标签/列标签  
包头又包尾

```
print (df[0:3])
print (df['20130102':'20130104'])
```

	A	B	C	D	E
2013-01-01	0.471913	0.011896	-1.511653	-1.097910	one
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-03	-0.092205	-0.021862	-1.026872	0.699465	two

	A	B	C	D	E
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-03	-0.092205	-0.021862	-1.026872	0.699465	two
2013-01-04	0.469188	0.087677	-0.019453	-1.467458	three

	A	B	C	D	E
2013-01-01	0.471913	0.011896	-1.511653	-1.097910	one
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-03	-0.092205	-0.021862	-1.026872	0.699465	two
2013-01-04	0.469188	0.087677	-0.019453	-1.467458	three
2013-01-05	0.756042	0.066811	-0.344194	0.914210	four
2013-01-06	-0.053143	-0.126425	-0.982280	0.480497	three



原Data frame

# pandas

## • Pandas实例

- 提取部分数据
- 提取一个数据块的实例如下，该实例提取行标签范围、和列标签列表对应的行列子集。
- 也可以通过行下标、列下标来提取行列子集。既然可以取得一个行列子集，便可以取得一个单元格的值

```
print (df.loc['20130102':'20130104', ['A', 'B'] ) #行标签范围('20130102'
print (df.iloc[3:5,0:2] ) # row=3, 4, col=0, 1
print (df.iloc[[1,2,4],[0,2]] ) # row=1, 2, 4, col=0, 2

print (df.iloc[1:3, :] ) # row=1, 2, col=all
print (df.iloc[:, 1:3] ) # row=all, col=1, 2

print (df.iloc[1,1] ) # 提取一个单元(cell)的值
print (df.iat[1,1] ) # 提取一个单元(cell)的值
```

行下标/列下标  
包头不包尾

行标签/列标签  
包头又包尾

	A	B			
2013-01-02	0.946843	1.889557			
2013-01-03	-0.092205	-0.021862			
2013-01-04	0.469188	0.087677			
	A	B			
2013-01-04	0.469188	0.087677			
2013-01-05	0.756042	0.066811			
	A	B			
2013-01-02	0.946843	0.427803			
2013-01-03	-0.092205	-1.026872			
2013-01-05	0.756042	0.066811			
	A	B	C	D	E
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-03	-0.092205	-0.021862	-1.026872	0.699465	two
	B	C			
2013-01-01	0.011896	-1.511653			
2013-01-02	1.889557	0.427803			
2013-01-03	-0.021862	-1.026872			
2013-01-04	0.087677	-0.019453			
2013-01-05	0.066811	-0.344194			
2013-01-06	-0.126425	-0.982280			
1.8895572949988484					
1.8895572949988484					

	A	B	C	D	E
2013-01-01	0.471913	0.011896	-1.511653	-1.097910	one
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-03	-0.092205	-0.021862	-1.026872	0.699465	two
2013-01-04	0.469188	0.087677	-0.019453	-1.467458	three
2013-01-05	0.756042	0.066811	-0.344194	0.914210	four
2013-01-06	-0.053143	-0.126425	-0.982280	0.480497	three



原Data frame



# pandas

## • Pandas实例

- **提取部分数据**
- 可以使用条件过滤，获取部分行数据。
- 下面示例代码中，第一行语句，只把df里的A列>0的行提取出来 ( True False 列表)
- 第二行语句，把df2的E列的值为'train'的行提取出来 ( True False 列表)

```
print (df[df.A > 0])  
print (df2[df2['E'].isin(['train'])])
```

	A	B	C	D	E
2013-01-01	0.471913	0.011896	-1.511653	-1.097910	one
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-04	0.469188	0.087677	-0.019453	-1.467458	three
2013-01-05	0.756042	0.066811	-0.344194	0.914210	four

	A	B	C	D	E	F
1	1.0	2013-01-02	1.0	3	train	foo
3	1.0	2013-01-02	1.0	3	train	foo

	A	B	C	D	E
2013-01-01	0.471913	0.011896	-1.511653	-1.097910	one
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-03	-0.092205	-0.021862	-1.026872	0.699465	two
2013-01-04	0.469188	0.087677	-0.019453	-1.467458	three
2013-01-05	0.756042	0.066811	-0.344194	0.914210	four
2013-01-06	-0.053143	-0.126425	-0.982280	0.480497	three

原Data frame



```
print (df2)
```

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

原Data frame



# pandas

## • Pandas实例

- 设置单元格的值(Setting)
- 可以通过行标签和列标签、或者通过行下标和列下标，对单元格进行的值进行设置。也可以对整列进行设置

	A	B	C	D	E
2013-01-01	-99.000000	0.011896	-1.511653	-1.097910	one
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-03	-0.092205	-0.021862	-1.026872	0.699465	two
2013-01-04	0.469188	0.087677	-0.019453	-1.467458	three
2013-01-05	0.756042	0.066811	-0.344194	0.914210	four
	A	B	C	D	E
2013-01-01	-99.000000	3.000000	-1.511653	-1.097910	one
2013-01-02	0.946843	1.889557	0.427803	1.376170	one
2013-01-03	-0.092205	-0.021862	-1.026872	0.699465	two
2013-01-04	0.469188	0.087677	-0.019453	-1.467458	three
2013-01-05	0.756042	0.066811	-0.344194	0.914210	four
	A	B	C	D	E
2013-01-01	-99.000000	3.000000	-1.511653	5	one
2013-01-02	0.946843	1.889557	0.427803	5	one
2013-01-03	-0.092205	-0.021862	-1.026872	5	two
2013-01-04	0.469188	0.087677	-0.019453	5	three
2013-01-05	0.756042	0.066811	-0.344194	5	four
2013-01-06	-0.053143	-0.126425	-0.982280	5	three

```
df.at[dates[0], 'A'] = -99 # 通过行标签和列标签，设定单元格的值
print (df.head())
```

```
df.iat[0,1] = 3 # 通过行下标和列下标，设定单元格的值
print (df.head())
```

```
df.loc[:, 'D'] = np.array([5] * len(df)) # 对整列进行设置，
print (df)
```



pandas



# pandas

## • Pandas实例

- 对缺失值的处理(Handle Missing Data)
- 对于缺失值，我们可以把包含缺失值的整行数据从DataFrame里剔除，或者把缺失值替换成某个有意义的值

	A	B	C	D	E
2013-01-02	0.946843	1.889557	0.427803	5	one
2013-01-03	-0.092205	-0.021862	-1.026872	5	two
2013-01-04	0.469188	0.087677	-0.019453	5	three
2013-01-05	0.756042	0.066811	-0.344194	5	four
2013-01-06	-0.053143	-0.126425	-0.982280	5	three

	A	B	C	D	E
2013-01-02	False	False	False	False	False
2013-01-03	False	False	False	False	False
2013-01-04	False	False	False	False	False
2013-01-05	False	False	False	False	False
2013-01-06	False	False	False	False	False

```
df.iat[0,2] = np.nan
print (df)
df = df.dropna(how='any') # 把包含缺失值的行删除
print (df)

print (pd.isnull(df))
df.fillna(value=5) # 用5代替缺失值
```

该语句现在不起作用，  
因为有null的行已经删除



整行删除

	A	B	C	D	E
<del>2013-01-01</del>	<del>-99.000000</del>	<del>3.000000</del>	<del>NaN</del>	<del>5</del>	<del>one</del>
2013-01-02	0.946843	1.889557	0.427803	5	one
2013-01-03	-0.092205	-0.021862	-1.026872	5	two
2013-01-04	0.469188	0.087677	-0.019453	5	three
2013-01-05	0.756042	0.066811	-0.344194	5	four
2013-01-06	-0.053143	-0.126425	-0.982280	5	three



# pandas

- Pandas实例

- 计算每列的均值
- 通过DataFrame的mean方法，可以计算每个数据列的均值

```
print (df.mean())
```

```
A    0.405345  
B    0.379152  
C   -0.388999  
D    5.000000  
dtype: float64
```

	A	B	C	D	E
2013-01-02	0.946843	1.889557	0.427803	5	one
2013-01-03	-0.092205	-0.021862	-1.026872	5	two
2013-01-04	0.469188	0.087677	-0.019453	5	three
2013-01-05	0.756042	0.066811	-0.344194	5	four
2013-01-06	-0.053143	-0.126425	-0.982280	5	three

原Data frame





# pandas

- Pandas实例

- 对每列运用一个函数
- 可以对DataFrame的每个数据列，运用某个函数，比如把每列的最大值减去最小值，计算出数据的极差(Range)等

```
df = df[['A', 'B', 'C', 'D']]
df.apply(lambda x: x.max() - x.min())
```

```
A    1.039049
B    2.015982
C    1.454675
D    0.000000
dtype: float64
```

	A	B	C	D	E
2013-01-02	0.946843	1.889557	0.427803	5	one
2013-01-03	-0.092205	-0.021862	-1.026872	5	two
2013-01-04	0.469188	0.087677	-0.019453	5	three
2013-01-05	0.756042	0.066811	-0.344194	5	four
2013-01-06	-0.053143	-0.126425	-0.982280	5	three



原Data frame

# pandas

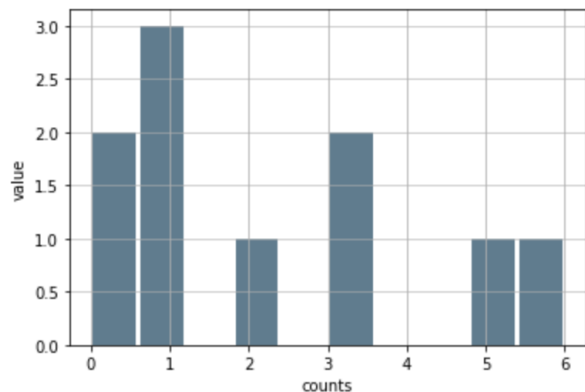
- Pandas实例

- 计算直方图

- 所谓直方图，是数据集里各个取值的频率的图形表示

- 下面示例代码，计算了s序列的每个值的频率

```
1  3
0  2
3  2
2  1
5  1
6  1
dtype: int64
```



```
s = pd.Series(np.random.randint(0, 7, size=10)) # 生成10个
print (s.value_counts())
```

```
import matplotlib.pyplot as plt
s.plot.hist( grid=True, rwidth=0.9, color='#607c8e')
plt.xlabel('counts')
plt.ylabel('value')
plt.grid(axis='y', alpha=0.75)
plt.show()
```





# pandas

- Pandas实例

- 字符串处理

- 我们可以以向量化的处理方式(一次处理若干个元素), 对序列进行字符串处理, 比如把所有的字符串都变成小写形式

```
s = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA', 'dog', 'cat'])  
s=s.str.lower()  
print (s)
```

```
0      a  
1      b  
2      c  
3    aaba  
4    baca  
5     NaN  
6    caba  
7     dog  
8     cat  
dtype: object
```



pandas



# pandas

- Pandas实例

- **DataFrame的合并(Concatenation)**
- 我们可以把若干个DataFrame(模式相同), 合并起来, 构成一个大的DataFrame
- 示例代码如下, 该实例把一个DataFrame横向切割成三个子集, 然后用concat进行合并, 构成一个大的DataFrame, 其内容和原来的DataFrame是一样的

```
df = pd.DataFrame(np.random.randn(10, 4))  
print (df)  
pieces = [df[:3], df[3:7], df[7:]]  
print (pd.concat(pieces))
```

	0	1	2	3
0	-1.955903	-0.598691	0.431124	-0.640989
1	-1.293307	0.160340	0.620917	-1.181500
2	0.417922	-0.849744	0.670945	0.068590
3	-0.099992	-1.598207	1.834540	-0.474502
4	-1.640941	1.205256	0.824748	-0.621626
5	0.691440	-1.074141	-0.094086	0.206462
6	-0.409218	1.097222	0.040145	0.252508
7	2.485014	-0.181712	-1.189108	1.851442
8	1.512801	-1.350204	-2.062768	-1.627815
9	0.027076	-0.535742	0.599989	-1.256684



	0	1	2	3
0	-1.955903	-0.598691	0.431124	-0.640989
1	-1.293307	0.160340	0.620917	-1.181500
2	0.417922	-0.849744	0.670945	0.068590
3	-0.099992	-1.598207	1.834540	-0.474502
4	-1.640941	1.205256	0.824748	-0.621626
5	0.691440	-1.074141	-0.094086	0.206462
6	-0.409218	1.097222	0.040145	0.252508
7	2.485014	-0.181712	-1.189108	1.851442
8	1.512801	-1.350204	-2.062768	-1.627815
9	0.027076	-0.535742	0.599989	-1.256684



# pandas

- Pandas实例

- **DataFrame的连接(Join)**

- 所谓连接操作，是根据一定的条件，把两个DataFrame数据集的各行，合起来构成目标DataFrame的一行

- 示例代码如下，该实例把left数据集和right数据集的每一行，根据名称为key的列的值相同，合起来构成目标DataFrame的一行

```
left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [1, 2]})
right = pd.DataFrame({'key': ['foo', 'bar'], 'rval': [4, 5]})
print (left)
print (right)
print (pd.merge(left, right, on='key'))
```



	key	lval	rval
0	foo	1	4
1	bar	2	5

	key	lval		key	rval
0	foo	1	→	0	foo 4
1	bar	2	→	1	bar 5

# pandas

## • Pandas实例

- 添加数据行(Append)
- DataFrame是一个可变的数据集，我们可以添加新的数据行
- 示例代码如下，该实例把一个DataFrame的第三行切下来，然后添加到原来的DataFrame的末尾，构成新的DataFrame

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

s = df.iloc[3]
print (s)
df = df.append(s, ignore_index=True)
print (df)
```

	A	B	C	D
0	-0.170713	-0.428259	-1.462121	-0.998988
1	0.483623	1.463841	1.475985	1.697572
2	-2.738456	-1.373551	-0.416422	-0.229570
3	0.182678	0.526500	0.342883	-1.175639
4	-0.243443	-0.975875	-0.329509	2.441080
5	-1.409846	0.139154	1.025720	-1.147729
6	0.109028	-0.200910	0.152663	-1.432073
7	-0.170260	1.421573	0.781451	-1.461284
8	0.182678	0.526500	0.342883	-1.175639



	A	B	C	D
0	-0.170713	-0.428259	-1.462121	-0.998988
1	0.483623	1.463841	1.475985	1.697572
2	-2.738456	-1.373551	-0.416422	-0.229570
3	0.182678	0.526500	0.342883	-1.175639
4	-0.243443	-0.975875	-0.329509	2.441080
5	-1.409846	0.139154	1.025720	-1.147729
6	0.109028	-0.200910	0.152663	-1.432073
7	-0.170260	1.421573	0.781451	-1.461284
A	0.182678			
B	0.526500			
C	0.342883			
D	-1.175639			
Name: 3, dtype: float64				

# pandas

## • Pandas实例

- **分组与聚集(grouping & aggregation)**
- 我们可以对DataFrame进行分组和聚集；分组是根据某个列的值把所有的行，分成一组一组的；而聚集，则是进行求和、最小值、最大值、平均值等的计算

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

print (df)

print (df.groupby('A').sum()) # 按照列A的值进行分组，计算其它数值字段的总和
print (df.groupby(['A', 'B']).sum()) # 按照列A、列B的值进行分组，计算其它数值字段的总和
```

	C	D
A		
bar	0.298821	0.779223
foo	1.229322	1.564485

A	B	C	D
bar	one	-0.919129	-0.652357
	three	0.192065	0.712106
	two	1.025886	0.719473
foo	one	0.753182	-0.168997
	three	0.198845	0.871971
	two	0.277295	0.861511



	A	B	C	D
0	foo	one	0.214199	-0.212185
1	bar	one	-0.919129	-0.652357
2	foo	two	-0.654375	0.764399
3	bar	three	0.192065	0.712106
4	foo	two	0.931670	0.097112
5	bar	two	1.025886	0.719473
6	foo	one	0.538983	0.043189
7	foo	three	0.198845	0.871971

原Data frame



# pandas

## • Pandas实例

- 数据透视表
- 我们可以为DataFrame创建数据透视表
- 下面的示例代码，创建了以A列、B列为行变量，以C列为列变量，以D列为单元格的值的数据透视表

		C	
		bar	foo
one	A	-2.077585	0.092560
	B	-1.940225	0.954845
	C	-1.141519	-1.587448
three	A	1.014101	NaN
	B	NaN	-0.129777
	C	0.240874	NaN
two	A	NaN	0.713681
	B	-0.470552	NaN
	C	NaN	-0.275911

```
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(12)})

print (df)
print (pd.pivot_table(df, values='D', index=['A', 'B'], columns=['C']))
```



DataFrame的数据

	A	B	C	D	E
0	one	A	foo	0.092560	0.963485
1	one	B	foo	0.954845	1.119986
2	two	C	foo	-0.275911	-1.638659
3	three	A	bar	1.014101	1.660604
4	one	B	bar	-1.940225	0.858434
5	one	C	bar	-1.141519	0.626182
6	two	A	foo	0.713681	-0.176162
7	three	B	foo	-0.129777	1.201548
8	one	C	foo	-1.587448	-1.476434
9	one	A	bar	-2.077585	0.555919
10	two	B	bar	-0.470552	-0.785403
11	three	C	bar	0.240874	-0.452803

透视表效果

# pandas

## • Pandas实例

### – 数据透视表

- 数据透视表的目的是对数据进行汇总，有可能透视表的一个cell对应原来DataFrame的多行数据，那么这个cell应该取什么样的值，由pivot\_table函数的aggfunc属性决定；比如当aggfunc=np.min，那么进行聚集的时候，取其中最小值，aggfunc还可以取最大值、平均值、总和等

- 示例代码如下，该实例对多个数据行的某一行，取平均值

DataFrame 的数据行和 pivot table 的 cell 的数据关系，可以通过图 5 查看。

ix	Item	CType	USD	EU
0	Item0	Gold	1	1
1	Item0	Bronze	2	2
2	Item0	Bold	3	3
3	Item	Silver	4	4

ix=Item	Bronze	Gold	Silver
Item0	2	2=mean(1,3)	NaN
Item1	NaN	NaN	4

d.pivot\_table(index='Item', column='CType', values='USD', aggfunc=np.mean)

```
from collections import OrderedDict
from pandas import DataFrame
import pandas as pd
import numpy as np

table = OrderedDict((
    ("Item", ['Item0', 'Item0', 'Item0', 'Item1']),
    ('CType', ['Gold', 'Bronze', 'Gold', 'Silver']),
    ('USD', [1, 2, 3, 4]),
    ('EU', [1.1, 2.2, 3.3, 4.4])
))
df = DataFrame(table)
print(df)
print(pd.pivot_table(df, index='Item', columns='CType', values='USD', aggfunc=np.mean))
```

Pivot\_tale及其数据的计算关系



pandas



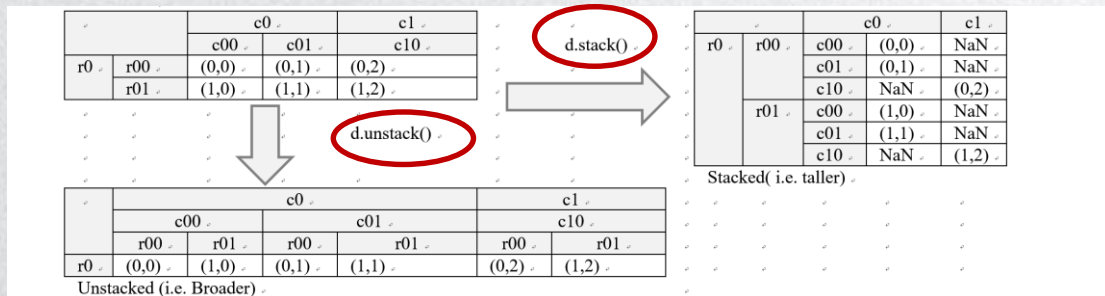


# pandas

## • Pandas实例

### – 重塑Reshape(stack&unstack)操作

- 对一个二维表进行透视表操作(Pivoting), 实际上是DataFrame的堆叠(Stacking)/反堆叠操作的特例。下图展示了堆叠/反堆叠操作的实例
  - 最初的DataFrame, 在行方向和列方向上有多个索引(MultiIndices)
- 对DataFrame进行堆叠操作, 它把列方向的最底层的索引(Innermost Column Index), 转换成行方向最底层的索引(Innermost Row Index), 当然相应的单元格数据也需要做出改变
- 堆叠操作的反操作, 是反堆叠, 它把行方向最底层的索引(Innermost Row Index), 转换成列方向最底层的索引(Innermost Column Index)



对汇总数据进行  
观察的不同行方  
向和列方向这

这个实例的行方向和列方向, 都包含二级索引。堆叠操作, 使得 DataFrame 变高了, 因为堆叠操作把数据在更少的列和更多的行上“叠加”起来。而反堆叠操作, 使得 DataFrame 变矮了、变宽了。

# pandas

## • Pandas实例

- 重塑Reshape(stack&unstack)操作
- 代码实例，**请读者在纸面上推演一下，加深理解**

这部分可选

			c0	c1
r0	r-00	c-00	(0, 0)	NaN
		c-01	(0, 1)	NaN
		c-10	NaN	(0, 2)
r-01	c-00	c-00	(1, 0)	NaN
		c-01	(1, 1)	NaN
		c-10	NaN	(1, 2)

```
from collections import OrderedDict
from pandas import DataFrame
import pandas as pd
import numpy as np

# 行方向的多级索引(row multi index)
row_idx_arr = list(zip(['r0', 'r0'], ['r-00', 'r-01'])) # r0有两个子节点r-00, r-01
row_idx = pd.MultiIndex.from_tuples(row_idx_arr)

# 列方向的多级索引(column multi index)
col_idx_arr = list(zip(['c0', 'c0', 'c1'], ['c-00', 'c-01', 'c-10'])) # c0有两个子节点, c1有一个子节点
col_idx = pd.MultiIndex.from_tuples(col_idx_arr)

# 创建DataFrame, 2行3列
d = DataFrame(np.arange(6).reshape(2, 3), index=row_idx, columns=col_idx)
d = d.applymap(lambda x: (x // 3, x % 3))

s = d.stack() # Stack
print(s)
u = d.unstack() # Unstack
print(u)
```

	c0		c1	
	c-00	c-01	c-00	c-01
r0	r-00	r-01	r-00	r-01
r0	(0, 0)	(1, 0)	(0, 1)	(1, 1)
			(0, 2)	(1, 2)



# pandas

- Pandas实例

- 时间序列(Time Series)数据处理
- Pandas提供了resample函数, 对时间序列数据进行频率转换(Frequency Conversion)和重新采样(Resample)
- 示例代码如下, 该实例把秒级采样的数据, 进行“每5分钟”的重新采样, 每五分钟里的秒级数据, 以求和的方式进行汇总

```
rng = pd.date_range('1/1/2012', periods=10, freq='S') # 频率为秒
ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng) # 备注:
print (ts)

ts = ts.resample('5Min').sum() # 进行“每5分钟”的重新采样
#ts = ts.resample('5min', how='sum')
print (ts)
```

2012-01-01 00:00:00	275
2012-01-01 00:00:01	355
2012-01-01 00:00:02	0
2012-01-01 00:00:03	221
2012-01-01 00:00:04	265
2012-01-01 00:00:05	427
2012-01-01 00:00:06	432
2012-01-01 00:00:07	414
2012-01-01 00:00:08	227
2012-01-01 00:00:09	264

Freq: S, dtype: int32

2012-01-01	2880
------------	------

Freq: 5T, dtype: int32





# pandas

## • Pandas实例

- 时间序列(Time Series)数据处理
- 时间序列数据的时间戳，可以改变时区设定
  - 比如，由世界标准时间(Coordinated Universal Time, UTC)，转换成美国东部的时间

```
2012-03-06    0.191659
2012-03-07   -0.209573
2012-03-08   -1.123520
2012-03-09   -0.761616
2012-03-10    1.793972
```

Freq: D, dtype: float64

```
2012-03-06 00:00:00+00:00    0.191659
2012-03-07 00:00:00+00:00   -0.209573
2012-03-08 00:00:00+00:00   -1.123520
2012-03-09 00:00:00+00:00   -0.761616
2012-03-10 00:00:00+00:00    1.793972
```

Freq: D, dtype: float64

```
2012-03-05 19:00:00-05:00    0.191659
2012-03-06 19:00:00-05:00   -0.209573
2012-03-07 19:00:00-05:00   -1.123520
2012-03-08 19:00:00-05:00   -0.761616
2012-03-09 19:00:00-05:00    1.793972
```

Freq: D, dtype: float64

```
# Time zone representation
rng = pd.date_range('3/6/2012 00:00', periods=5, freq='D')# 频率为天
ts = pd.Series(np.random.randn(len(rng)), index= rng)# 备注：生成5个小数
print (ts)

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

ts_new = ts_utc.tz_convert('US/Eastern')
print (ts_new)
```



# pandas

## • Pandas实例

- 时间序列(Time Series)数据处理
- 时间序列数据，可以分为时期序列(Period)和时点序列(Point)。每个月末的外汇储备总额，就是一个时点序列。而每个月的出口额，则是一个时期序列。Pandas提供函数，在这两类时间序列之间，进行转换。示例代码如下，该实例把时点序列转换成时期序列，再转换成时点序列

```
2012-01-31    -1.480303
2012-02-29    -0.729128
2012-03-31    -0.628868
2012-04-30    -0.669048
2012-05-31     0.975180
```

```
Freq: M, dtype: float64
```

```
2012-01    -1.480303
2012-02    -0.729128
2012-03    -0.628868
2012-04    -0.669048
2012-05     0.975180
```

```
Freq: M, dtype: float64
```

```
2012-01-01    -1.480303
2012-02-01    -0.729128
2012-03-01    -0.628868
2012-04-01    -0.669048
2012-05-01     0.975180
```

```
Freq: MS, dtype: float64
```

时期序列

时点序列

```
# Converting between time span representations
rng = pd.date_range('1/1/2012', periods=5, freq='M') # 频率为月
ts = pd.Series(np.random.randn(len(rng)), index=rng) # 备注: 生成5个小数
print (ts)

ps = ts.to_period()
print (ps)

ts = ps.to_timestamp()
print (ts)
```



# pandas

## • Pandas实例

- 时间序列(Time Series)数据处理
- 在时点序列和时期序列之间进行转换，可以使我们方便地实现一些算术运算。
- 该实例把频率为季度的时间序列数据
  - 转换成每个季度最末一个月的第一天的上午9点的时点序列数据

- Q-NOV表示财年从本年12月到下年11月，四个季度分别是12/1/2月，3/4/5月，6/7/8月，以及9/10/11月
- `prng.asfreq('M', 'e') + 1`，把每个季度的最末月份加上1
- `.asfreq('H','s')+9`表示每天的开始0时加上9个小时

```
prng = pd.period_range('1990Q1', '2000Q4', freq='Q-NOV')
ts = pd.Series(np.random.randn(len(prng)), index=prng)
print (ts.head())

ts.index = (prng.asfreq('M', 'e') + 1).asfreq('H', 's') + 9
# 季度最末尾一个月第一天上午9点 M表示Month, e表示end, H表示Hour, s表示start
print (ts.head())
```

Month end hour start

1990-03-01 09:00	-0.890051
1990-06-01 09:00	0.990587
1990-09-01 09:00	1.165205
1990-12-01 09:00	0.455105
1991-03-01 09:00	-0.467067

Freq: H, dtype: float64

1990Q1	-0.890051
1990Q2	0.990587
1990Q3	1.165205
1990Q4	0.455105
1991Q1	-0.467067

Freq: Q-NOV, dtype: float64

原Data frame



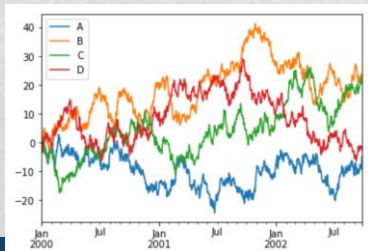
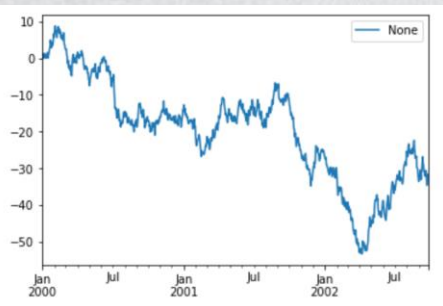


# pandas

## • Pandas实例

### – 绘图

- 我们可以使用matplotlib对pandas数据进行可视化。下面展示了两个实例，第一个实例显示了一个Series，该序列是从2000年1月1日开始1000天的随机数序列；第二个实例，显示了一个DataFrame，它使用的行标签和第一个实例的行标签是一样的



### # 可视化Series

```
import matplotlib.pyplot as plt
ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))
plt.figure();
ts = ts.cumsum() # cumulative sum即累计值
ts.plot()
plt.legend(loc='best')
plt.show()
```

### # 可视化DataFrame

```
df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=['A', 'B', 'C', 'D'])
df = df.cumsum() # cumulative sum即累计值
plt.figure();
df.plot();
plt.legend(loc='best')
plt.show()
```



# pandas

## • Pandas实例

### – 文件读/写

- 利用pandas提供的函数，我们可以把数据保存到文件中，也可以从文件中读取数据；Pandas支持的数据文件格式，包括CSV、HDF5、Excel等

### Dataframe数据，前5行

Unnamed: 0	A	B	C	D
0	1.295959	-0.122926	-0.415229	1.394466
1	1.295942	-0.061915	-0.996014	-0.919578
2	-2.047141	0.234107	-1.437758	1.409104
3	-0.604988	0.160255	-0.356097	-0.809044
4	-1.206840	-1.088032	2.711974	0.836574

### 3.恢复数据

```
1,A,B,C,D
2,0,1.2959592640036128,-0.12292629458803561,-0.4152286509860301,1.3944663589712691
3,1,1.2959423043311495,-0.06191521729049765,-0.996014136805057,-0.9195776973432487
4,2,-2.047141130780656,0.23410731393721107,-1.4377578409067382,1.4091040645378368
5,3,-0.6049875649911697,0.16025519753264553,-0.3560973735681203,-0.8090444836148311
6,4,-1.206839959149915,-1.0880318014652577,2.7119743716600104,0.836574423353176
7,5,2.1855542485843262,-0.1468512078706515,0.7499152983934357,-1.0649473841563954
8,6,0.5234072574687985,0.9896701901088677,0.42393435893441983,0.6021390454792174
9,7,0.947218920375,-2.12576279263943,0.13015059916248276,-0.8622088592832489
```

### 2.CSV文件

```
# 读写文件
df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
print (df.head())

# Writing to & read from a csv file
df.to_csv('foo.csv')
df = pd.read_csv('foo.csv')
print (df.head())
```



	A	B	C	D
0	1.295959	-0.122926	-0.415229	1.394466
1	1.295942	-0.061915	-0.996014	-0.919578
2	-2.047141	0.234107	-1.437758	1.409104
3	-0.604988	0.160255	-0.356097	-0.809044
4	-1.206840	-1.088032	2.711974	0.836574

### 1.数据

# pandas

- Pandas实例

- 文件读/写

- 利用pandas提供的函数，我们可以把数据保存到文件中，也可以从文件中读取数据。Pandas支持的数据文件格式，包括CSV、HDF5、Excel等

	A	B	C	D
0	-0.206193	1.245800	0.687127	-0.057650
1	-0.896292	2.222805	-0.088637	-0.621545
2	1.627094	0.951365	-0.200073	-0.070609
3	-0.356627	0.057124	1.066961	-0.477339
4	-1.789168	-1.052720	-0.666424	2.552375

```
# Writing to & read from a HDF5 Store
```

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

```
df.to_hdf('foo.h5', 'df')  
df = pd.read_hdf('foo.h5', 'df')  
print (df.head())
```



	A	B	C	D
0	-0.206193	1.245800	0.687127	-0.057650
1	-0.896292	2.222805	-0.088637	-0.621545
2	1.627094	0.951365	-0.200073	-0.070609
3	-0.356627	0.057124	1.066961	-0.477339
4	-1.789168	-1.052720	-0.666424	2.552375

3.恢复数据

2. HDF5文件

1.数据



# pandas

## • Pandas实例

### – 文件读/写

- 利用pandas提供的函数，我们可以把数据保存到文件中，也可以从文件中读取数据。Pandas支持的数据文件格式，包括CSV、HDF5、Excel等



```
# Writing to & read from an excel file
df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
print (df.head())

df.to_excel('foo.xlsx', sheet_name='Sheet1')
df = pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
print (df.head())
```

Unnamed: 0	A	B	C	D
0	0 -2.405966	-0.859835	-0.858694	0.427761
1	1 0.083608	1.847162	-0.764215	0.514822
2	2 -0.673479	-0.591838	-0.175998	0.652206
3	3 -0.737217	0.920646	-1.745612	-0.504261
4	4 -1.747235	0.285935	0.460740	1.598113

	A	B	C	D
0	-2.405966	-0.859835	-0.858694	0.427761
1	0.083608	1.847162	-0.764215	0.514822
2	-0.673479	-0.591838	-0.175998	0.652206
3	-0.737217	0.920646	-1.745612	-0.504261
4	-1.747235	0.285935	0.460740	1.598113

3.恢复数据

2. Excel文件

1.数据