RiceQuant 量化教程

来源：https://www.ricequant.com/courses

Python 编程

基本的数据类型¶

在Python中，能够直接处理的数据类型有以下几种：

整数¶

Python可以处理任意大小的整数，包括负整数，写程序的时候表述与数学上的方法一样，例如：99，-3，6666 等等。

浮点数¶

浮点数也可以称为小数。叫做浮点数是因为按照科学记数法表示时，一个浮点数的小数点位置是可变的。比如，$1.11x10^6$和$11.1x10^5$是完全相等的。对于很大或很小的浮点数，必须用科学计数法表示，把10用e替代，$1.11x10^6$就是1.11e6，或者11.1e5，0.0000011可以写成1.1e-6。负数的话直接在前面加负号即可，例如：-1.1e-6。

需要注意的一点是，整数和浮点数在计算机内部存储的方式是不同的， 整数 之间的运算永远是精确的，而 浮点数 运算则可能会有四舍五入的误差。

字符串¶

字符串是以单引号 ' 或双引号 " 括起来的任意文本' '或" "只是一种表示方式，不是字符串的一部分，所以字符串'rice'只有r，i，c，e这4个字符。如果要把'本身也包括进字符里，那就可以用" "括起来。如果字符串内部既包含 ' 又包含 " 怎么办呢？我们可以用转义字符 \ 来标识，例如：I\'m \"OK\"!表示的内容为：I'm "OK"!

转义字符\可以转义很多字符，比如\n表示换行，\t表示制表符，字符\本身也要转义，所以\表示的字符就是\，用print()打印字符串看看结果如何：

In [1]:

```
print('I\'m \"OK\"!')
```

```
I'm "OK"!
```

In [2]:

```
print('Ricequant is\ngreat')
```

```
Ricequant is
great
```

In [3]:

```
print('\\\t\\')
```

```
\	\
```

布尔值¶

布尔值和布尔代数的表示完全一致，一个布尔值只有True、False两种值。Python中，既可以直接用True、False表示（一定要大写），也可以通过布尔运算计算出来：

In [4]:

```
True
```

Out[4]:

```
True
```

In [5]:

```
False
```

Out[5]:

```
False
```

In [6]:

```
1>0
```

Out[6]:

```
True
```

布尔值可以用and、or和not即与、或和非进行运算。

In [7]:

```
True and False
```

Out[7]:

```
False
```

In [8]:

```
1 > 2 or 3 < 10
```

Out[8]:

```
True
```

In [9]:

```
not 3 < 10
```

Out[9]:

```
False
```

列表¶

Python内嵌的数据类型主要包括以下两类：

有序：¶

List（列表），是有序集合，没有固定大小，可以通过对偏移量以及其他方法修改列表大小。列表的基本形式如：[1,2,3,4]

Tuple（元组），是有序集合，是不可变的，可以进行组合和复制运算后会生成一个新的元组。元组的基本形式比如：(1,3,6,10)

String（字符串），也是有序集合，字符串的基本形式比如：'hello'，这里不进行具体介绍。

无序：¶

Set（集合），是一个无序不重复元素的集。基本功能包括关系运算和消除重复元素。集合的基本形式如：set('abracadabra')

Dictionary（字典）是无序的键：值对 (key:value 对)集合，键必须是互不相同的(在同一个字典之内)。字典的基本形式如：{'jack': 4098, 'sape': 4139} 首先对列表进行介绍。

List（列表） 是 Python 中最通用的序列。列表是一个任意类型对象位置的相关有序集合，它没有固定大小。不像字符串，其大小是可以变的，通过对偏移量进行赋值以及其他各种列表的方法进行调用，可以修改列表大小。

索引是从0开始而非1开始！！¶

列表中值的分割用变量[头下标:尾下标]，就可以截取相应的列表，从左到右索引默认“0”开始的，从右到左索引默认-1开始，下标可以为空表示取到头或尾。可以对列表进行索引、切片等操作，看下面例子。

列出一周5个工作日可以使用list：

In [12]:

```
week = ['Monday','Tuesday','Wednesday','Thursday','Friday']
week
```

Out[12]:

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

week是一个list，可以用len()获取其元素个数：

In [13]:

```
len(week)
```

Out[13]:

```
5
```

可以使用索引来访问list中的每一个位置上的元素，不过一定要记住索引是从0开始的！！

In [14]:

```
week[0]
```

Out[14]:

```
'Monday'
```

In [15]:

```
week[3]
```

Out[15]:

```
'Thursday'
```

In [16]:

```
week[5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-16-aab97e4d2f9f> in <module>()
----> 1 week[5]

IndexError: list index out of range
```

当索引超出了范围时，Python会报一个IndexError错误。由于索引是从0开始的，最后一个元素的索引是列表长度 - 1。

python支持从列表最后取元素，如果要取最后一个元素，可以用-1做索引，直接获取最后一个元素：

In [17]:

```
week[-1]
```

Out[17]:

```
'Friday'
```

In [18]:

```
week[-4]
```

Out[18]:

```
'Tuesday'
```

由于list是一个可变的有序表，所以我们可以往list中追加元素到末尾：

In [20]:

```
week.append('Saturday')
week
```

Out[20]:

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
```

还可以吧元素插入到指定的位置，比如索引号为6的位置：

In [21]:

```
week.insert(6,'Sunday')
week
```

Out[21]:

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

删除list末尾的元素，使用pop()方法：

In [22]:

```
week.pop()
week
```

Out[22]:

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
```

删除list中指定位置的元素，使用pop(i)方法，i是对应的索引位置：

In [24]:

```
week.pop(1)
week
```

Out[24]:

```
['Monday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
```

把某个元素替换成别的元素，可以直接赋值给对应的索引位置：

In [25]:

```
week[0] = 'Tuesday'
week
```

Out[25]:

```
['Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
```

Python 的列表数据类型包含更多的方法。¶

list.append(x) 把一个元素添加到列表的结尾。

list.extend(L) 将一个给定列表中的所有元素都添加到另一个列表中。

list.insert(i, x) 在指定位置插入一个元素。第一个参数是准备插入到其前面的那个元素的索引，例如 a.insert(0, x) 会插入到整个列表之前，而 a.insert(len(a), x) 相当于 a.append(x)。

list.remove(x) 删除列表中值为 x 的第一个元素。如果没有这样的元素，就会返回一个错误。

list.pop([i]) 从列表的指定位置删除元素，并将其返回。如果没有指定索引，a.pop() 返回最后一个元素。元素随即从链表中被删除。(方法中 i 两边的方括号表示这个参数是可选的，而不是要求你输入一对方括号，这个经常会在 Python 库参考手册中遇到这样的标记。)

list.index(x) 返回列表中第一个值为 x 的元素的索引。如果没有匹配的元素就会返回一个错误。

list.count(x) 返回 x 在链表中出现的次数。

list.sort(cmp=None, key=None, reverse=False) 对列表中的元素进行排序（参数可以用来自定义排序方法，参考 sorted() 的更详细的解释）。

list.reverse() 就地倒排链表中的元素

del list[i] 有个方法可以从列表中按给定的索引而不是值来删除一个子项：del 语句。它不同于有返回值的 pop() 方法。语句 del 还可以从列表中删除切片或清空整个列表(我们以前介绍过一个方法是将空列表赋值给列表的切片)。

In [ ]:

```

```

字典（dictionary ）¶

字典在某些语言中可能称为 联合内存 (associative memories) 或 联合数组 (associative arrays)。序列是以连续的整数为索引，与此不同的是，字典以"关键字"为索引，关键字可以是任意不可变类型，通常用字符串或数值。如果元组中只包含字符串和数字，它可以作为关键字，如果它直接或间接地包含了可变对象，就不能当做关键字。不能用列表做关键字，因为列表可以用索引、切割或者 append() 和 extend() 等方法改变。

字典是无序的键：值对 (key:value 对)集合，键必须是互不相同的(在同一个字典之内)。使用大括号创建一个空的字典：{}。初始化列表时，在大括号内放置一组逗号分隔的键：值对，这也是字典输出的方式。

字典的主要操作是依据键来存储和取值。也可以用 del 来删除键：值对(key:value)，从一个不存在的键中取值会导致错误。

In [2]:

```
d = {'rice':35, 'wheat':101, 'corn':67}
print(d)
print(d['rice'])
```

```
{'rice': 35, 'wheat': 101, 'corn': 67}
35
```

把数据放入dict还可以直接通过key放入：

In [3]:

```
d['egg'] = 33
d
```

Out[3]:

```
{'corn': 67, 'egg': 33, 'rice': 35, 'wheat': 101}
```

一个key只能对应一个value，多次对一个key放入value，后面的值会把前面的值冲掉：

In [4]:

```
d['corn'] = 88
d
```

Out[4]:

```
{'corn': 88, 'egg': 33, 'rice': 35, 'wheat': 101}
```

如果key不存在，dict就会报错。要避免key不存在的错误，有两种办法，一是通过 in 判断key是否存在，二是通过dict提供的 get 方法，如果key不存在，可以返回None（返回None的时候Python的交互式命令行不显示结果），或者自己指定的value：

In [5]:

```
d['meat']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-5-d3e7ffd8023e> in <module>()
----> 1 d['meat']

KeyError: 'meat'
```

In [6]:

```
'meat' in d
```

Out[6]:

```
False
```

In [7]:

```
d.get('meat')
```

In [8]:

```
d.get('meat',45)
```

Out[8]:

```
45
```

删除一个key，使用pop(key)方法，对应的value也会从dict中删除：

In [9]:

```
d.pop('rice')
d
```

Out[9]:

```
{'corn': 88, 'egg': 33, 'wheat': 101}
```

有下面几点需要注意：

1. dict内部存放的顺序和key放入的顺序是没有关系的
2. dict查找和插入的速度极快，不会随着key的增加而增加，但是需要占用大量的内存，内存浪费多
3. dict的key必须是不可变对象。字符串、整数等都是不可变的，可以放心地作为key，而list是可变的，就不能作为key：

In [11]:

```
k = [3,4,5]
d[k] = 6
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-725fccd6cc45> in <module>()
      1 k = [3,4,5]
----> 2 d[k] = 6

TypeError: unhashable type: 'list'
```

常见字典操作方法¶

D.clear()删除字典内所有元素

D.copy()返回一个字典的复制

D.fromkeys(seq,val)创建一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值

D.get(key, default=None)返回指定键的值，如果值不在字典中返回default值

D.has_key(key)如果键在字典dict里返回true，否则返回false

D.items()以列表返回可遍历的(键, 值) 元组数组

D.keys()以列表返回一个字典所有的键

D.setdefault(key, default=None)和get()类似, 但如果键不存在于字典中，将会添加键并将值设为default

D.update(dict2)把字典dict2的键/值对更新到dict里

D.values()以列表返回字典中的所有值

D.pop(key)删除一个键并返回它的值，类似于列表的pop,只不过删除的是一个键不是一个可选的位置

del D[key]删除键

D[key] = 42新增或修改键

字典用法注意事项：¶

1. 序列运算无效，字典元素间是没有顺序的概念
2. 对新索引赋值会添加项
3. 键不一定总是字符串

多种构造字典方式¶

dict() 构造函数可以直接从 key-value 对中创建字典

In [15]:

```
dict([('frank', 4139), ('hacken', 4127), ('maxwell', 4098)])
```

Out[15]:

```
{'frank': 4139, 'hacken': 4127, 'maxwell': 4098}
```

In [16]:

```
dict.fromkeys(['a','b'],0) #创建一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值
```

Out[16]:

```
{'a': 0, 'b': 0}
```

In [17]:

```
dict(zip(['a','b','c'],[1,2,3]))
```

Out[17]:

```
{'a': 1, 'b': 2, 'c': 3}
```

In [18]:

```
{k:v for (k,v) in zip(['a','b','c'],[1,2,3])}
```

Out[18]:

```
{'a': 1, 'b': 2, 'c': 3}
```

此外，字典推导式可以从任意的键值表达式中创建字典:

In [20]:

```
{x: x**2 for x in (2, 4, 6)}
```

Out[20]:

```
{2: 4, 4: 16, 6: 36}
```

如果关键字都是简单的字符串，有时通过关键字参数指定 key-value 对更为方便:

In [21]:

```
D = dict(a=1,b=2,c=3)
D
```

Out[21]:

```
{'a': 1, 'b': 2, 'c': 3}
```

In [26]:

```
{c:c*4 for c in 'RiceQuant'}#默认是集合
```

Out[26]:

```
{'Q': 'QQQQ',
 'R': 'RRRR',
 'a': 'aaaa',
 'c': 'cccc',
 'e': 'eeee',
 'i': 'iiii',
 'n': 'nnnn',
 't': 'tttt',
 'u': 'uuuu'}
```

In [30]:

```
{c:c*4 for c in ['RiceQuant']}
```

Out[30]:

```
{'RiceQuant': 'RiceQuantRiceQuantRiceQuantRiceQuant'}
```

In [29]:

```
{c.lower():c*4+'!' for c in 'RiceQuant'}
```

Out[29]:

```
{'a': 'aaaa!',
 'c': 'cccc!',
 'e': 'eeee!',
 'i': 'iiii!',
 'n': 'nnnn!',
 'q': 'QQQQ!',
```

```
 'r': 'RRRR!',
 't': 'tttt!',
 'u': 'uuuu!'}
```

元组（tuple）¶

tuple是另一种有序的数据类型，与list比较类似。主要不同的一点是tuple被创建后就不能对其进行修改。所以，tuple与list不同，没有append(),pop(),insert()这些方法可以使用。获取元素的方法和list是一样的，可以通过索引来访问（也是从0开始的），只不过不能赋值成为其他的元素。

因为tuple不可变，所以代码更安全。如果可以的话，我们尽量使用tuple代替list。

创造元组¶

定义一个空的tuple，使用 () ：

In [31]:

```
t = ()
t
```

Out[31]:

```
()
```

只有1个元素的元组在进行定义的时候，需要加一个逗号 , 来消除歧义，否则定义的就不是一个元组而是元素本身：

In [33]:

```
t = (3)
t
```

Out[33]:

```
3
```

In [35]:

```
t = (3,)
t
```

Out[35]:

```
(3,)
```

元组的连接¶

如前面所说，元组是不可改的，但是可以连接

可以使用 + 对元组进行连接，例如：

In [36]:

```
t1 = (2,3,5)
t2 = ('ricequant','python')
t3 = t1 + t2
t3
```

Out[36]:

```
(2, 3, 5, 'ricequant', 'python')
```

元组的删除¶

元组中的元素不能被删除，但是我们可以使用 del 删除整个元组，删除后可以重新定义：

In [38]:

```
t4 = ('a',2,'b')
t4
```

Out[38]:

```
('a', 2, 'b')
```

In [39]:

```
del t4
t4
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-39-00422465996d> in <module>()
      1 del t4
----> 2 t4

NameError: name 't4' is not defined
```

事实上，上面所说的元素不变是指每个元素的指向永远不变。即指向1，就不能改成指向2，指向一个list，就不能改成指向其他对象，但指向的这个list本身是可变的，例如：

In [41]:

```
tup = (1,2,[3,4,5])
tup
```

Out[41]:

```
(1, 2, [3, 4, 5])
```

In [42]:

```
tup[2][0] = 7
tup[2][1] = 11
tup
```

Out[42]:

```
(1, 2, [7, 11, 5])
```

Python 的元组数据类型包含更多的方法。¶

tup.index(x, [start, [stop]])) 返回元组中start到stop索引中第一个值为 x 的元素在整个列表中的索引。如果没有匹配的元素就会返回一个错误。

tup.count(x) 返回 x 在元组中出现的次数。

cmp(tuple1, tuple2) 比较元组中两个元素。

len(tuple) 计算元组元素个数。

max(tuple) 返回元组中元素最大值。

min(tuple) 返回元组中元素最小值。

tuple(seq) 将列表转换为元组。

元组不提供字符串、列表和字典中的方法。如果相对元组排序，通常先得将它转换为列表并使其成为一个可变对象，才能获得使用排序方法，或使用sorted内置方法。

集合（set）¶

与dict类似，set也是一组key的集合，但不存储value。由于key不能重复，所以，在set中，没有重复的key。创建一个set，需要提供一个list作为输入集合：

In [43]:

```
s = set([3,4,5])
s
```

Out[43]:

```
{3, 4, 5}
```

重复元素在set中会被自动被过滤，通过add(key)方法往set中添加元素，重复添加不会有效果，通过remove(key)方法可以删除元素,例如：

In [44]:

```
s = set([3,3,4,4,5,5,6,6,7])
s
```

Out[44]:

```
{3, 4, 5, 6, 7}
```

In [45]:

```
s.add(1)
s
```

Out[45]:

```
{1, 3, 4, 5, 6, 7}
```

In [46]:

```
s.add(1)
s
```

Out[46]:

```
{1, 3, 4, 5, 6, 7}
```

In [47]:

```
s.remove(5)
s
```

Out[47]:

```
{1, 3, 4, 6, 7}
```

由于set是无序和无重复元素的集合，所以两个set可以做数学意义上的交并集等操作：

In [48]:

```
s1 = set([3,4,5,6])
s2 = set([5,6,7,8,9])
s1 & s2
```

Out[48]:

```
{5, 6}
```

In [49]:

```
s1 | s2
```

Out[49]:

```
{3, 4, 5, 6, 7, 8, 9}
```

与dict一样，set同样不可以放入可变对象，因为无法判断两个可变对象是否相等，也就无法保证set内部不会有重复元素。所以把list放入set，会报错。

In [50]:

```
s = set([1,2,[1,2]])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-806dee2b7dcf> in <module>()
----> 1 s = set([1,2,[1,2]])

TypeError: unhashable type: 'list'
```

条件与循环：if、while 、for¶

一、 条件if¶

格式：¶

if 判断条件：

```
执行语句……

```

else：

```
执行语句……
```

当if有多个条件时可使用括号来区分判断的先后顺序，括号中的判断优先执行，此外 and 和 or 的优先级低于>（大于）、<（小于）等判断符号，即大于和小于在没有括号的情况下会比与或要优先判断。

由于 python 并不支持 switch 语句，所以多个条件判断，只能用 elif 来实现，如果判断需要多个条件需同时判断时，可以使用 or （或），表示两个条件有一个成立时判断条件成功；使用 and （与）时，表示只有两个条件同时成立的情况下，判断条件才成功。

if 判断条件1：

```
执行语句1……
```

elif 判断条件2：

```
执行语句2……
```

elif 判断条件3：

```
执行语句3……
```

else：

```
执行语句4……
```

In [2]:

```
num = 5
if num == 3:            # 判断num的值
    print('3')
elif num == 2:
    print ('2')
elif num == 1:
    print ('1')
elif num < 0:           # 值小于零时输出
    print ('error')
else:
    print ('roadman')     # 条件均不成立时输出
```

```
roadman
```

1.1 and、or¶

对or而言，Python会由左到右求算操作对象，然后返回第一个为真的操作对象。

Python会在其找到的第一个真值操作数的地方停止，通常叫短路计算。

In [3]:

```
2 < 3, 3 < 2
```

Out[3]:

```
(True, False)
```

In [3]:

```
2 or 3, 3 or 2
```

Out[3]:

```
(2, 3)
```

如果左边的操作数为假（空对象），Python只会计算右边的操作数并将其返回

In [4]:

```
[] or 3
```

Out[4]:

```
3
```

In [5]:

```
[] or {}
```

Out[5]:

```
{}
```

and 会停在第一个为假的对象上

In [6]:

```
2 and 3, 3 and 2
```

Out[6]:

```
(3, 2)
```

In [7]:

```
[] and {} #空 list 本身等同于 False
```

Out[7]:

```
[]
```

In [6]:

```
#由于一个空 list 本身等同于 False，所以可以直接：
mylist=[]
if mylist:
  # Do something with my list
  pass
else:
  # The list is empty
  pass
```

1.2 神奇的布尔值¶

从一个固定大小的集合中选择非空的对象，只要将其串在一个or表达式即可

In [20]:

```
X = 'A' or 'B' or 'C' or None #会把X设为A、B以及C中第一个非空（为真）的对象，或者所有对象都为空就设为None
```

In [21]:

```
X = 'A' or default # 如果A为真（或非空）的话将X设置为A，否则，将X设置为default
```

In [ ]:

```
if f1() or f2():...# 如果f1返回真值（非空），Python将不再执行f2，若要保证两个函数都执行，需要在or之前调用他们，如

tmp1, tmp2 = f1(), f2()
if tmp1 or tmp2:...
```

二、循环¶

稍后会介绍更加奇特的迭代工具，如生成器、filter和reduce。现在先从最基础的学起。

Python提供了for循环和while循环（在Python中没有do...while循环），for循环一般比while计数器循环运行得更快

break语句，在语句块执行过程中终止循环，并且跳出整个循环

continue语句，在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环。

pass语句，是空语句，是为了保持程序结构的完整性。不做任何事情，一般用做占位语句。

循环else块，只有当循环正常离开时才会执行（也就是没有碰到break语句）

2.1 while¶

In [8]:

```
count = 0
while (count < 9):
   print ('The count is:', count)
   count = count + 1
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
```

2.2 for¶

是一个通用的序列迭代器，可以遍历任何有序的序列对象内的元素。可用于字符串、列表、元组、其他内置可迭代对象等

In [ ]:

```
for iterating_var in sequence:
   statements(s)
```

In [9]:

```
for letter in 'Python':     # 第一个实例
   print ('当前字母 :', letter)

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # 第二个实例
   print ('当前字母 :', fruit)
```

```
当前字母 : P
当前字母 : y
当前字母 : t
当前字母 : h
当前字母 : o
当前字母 : n
当前字母 : banana
当前字母 : apple
当前字母 : mango
```

In [10]:

```
T = [(1,2), (3,4), (5,6)]
for (a,b) in T:
    print (a,b)
```

```
1 2
3 4
5 6
```

In [11]:

```
D = {'a':1,'b':2,'c':3}
for key in D:
    print (key,'=>',D[key])
```

```
b => 2
a => 1
c => 3
```

In [12]:

```
D.items()
```

Out[12]:

```
dict_items([('b', 2), ('a', 1), ('c', 3)])
```

In [13]:

```
for (key,value) in D.items():
    print (key,'=>',value)
```

```
b => 2
a => 1
c => 3
```

嵌套的结构可以自动解包¶

In [14]:

```
for ((a,b),c) in [((1,2),3),((4,5),6)]: print (a,b,c)
```

```
1 2 3
4 5 6
```

2.3 break¶

In [15]:

```
for letter in 'Python':     # First Example
   if letter == 'h':
      break
   print ('Current Letter :', letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
```

2.4 continue ¶

In [17]:

```
for letter in 'Python':     # 第一个实例
   if letter == 'h':
      continue
   print ('当前字母 :', letter)
```

```
当前字母 : P
当前字母 : y
当前字母 : t
当前字母 : o
当前字母 : n
```

2.5 pass¶

In [19]:

```
for letter in 'Python':
   if letter == 'h':
      pass
      print ('这是 pass 块')
   print ('当前字母 :', letter)
```

```
当前字母 : P
当前字母 : y
当前字母 : t
这是 pass 块
当前字母 : h
当前字母 : o
当前字母 : n
```

In [ ]:

```

```

函数¶

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。函数能提高应用的模块性，和代码的重复利用率。

一、调用函数¶

Python内置了很多有用的函数，我们可以直接调用。

要调用一个函数，需要知道函数的名称和参数，比如求绝对值的函数`abs`，只有一个参数。可以直接从Python的官方网站查看文档：

http://docs.python.org/2/library/functions.html#abs

也可以在交互式命令行通过`help(abs)`查看`abs`函数的帮助信息。

调用`abs`函数：

In [4]:

```
abs(-500)
```

Out[4]:

```
500
```

调用函数的时候，如果传入的参数数量不对，会报`TypeError`的错误，并且Python会明确地告诉你：abs()有且仅有1个参数，但给出了两个：

In [5]:

```
abs(1, 5)
```

```
TypeErrorTraceback (most recent call last)
<ipython-input-5-aad0fc61c90d> in <module>()
----> 1 abs(1, 5)

TypeError: abs() takes exactly one argument (2 given)
```

如果传入的参数数量是对的，但参数类型不能被函数所接受，也会报`TypeError`的错误，并且给出错误信息：`str`是错误的参数类型：

In [6]:

```
abs('s')
```

```
TypeErrorTraceback (most recent call last)
<ipython-input-6-62d1aa8a5ee7> in <module>()
----> 1 abs('s')

TypeError: bad operand type for abs(): 'str'
```

二、定义函数¶

在Python中，定义一个函数要使用`def`语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用`return`n语句返回。

我们以自定义一个求绝对值的`my_abs`函数为例：

In [7]:

```
def my_abs(x):
    if x >= 0:
        return x
    else:
        return -x
```

测试并调用my_abs看看返回结果是否正确:

In [9]:

```
my_abs(-500)
```

Out[9]:

```
500
```

请注意，函数体内部的语句在执行时，一旦执行到`return`时，函数就执行完毕，并将结果返回。因此，函数内部通过条件判断和循环可以实现非常复杂的逻辑。

如果没有`return`语句，函数执行完毕后也会返回结果，只是结果为`None`。

`return None`可以简写为`return`。

三、函数的参数¶

定义函数的时候，我们把参数的名字和位置确定下来，函数的接口定义就完成了。对于函数的调用者来说，只需要知道如何传递正确的参数，以及函数将返回什么样的值就够了，函数内部的复杂逻辑被封装起来，调用者无需了解。

Python的函数定义非常简单，但灵活度却非常大。除了正常定义的必选参数外，还可以使用默认参数、可变参数和关键字参数，使得函数定义出来的接口，不但能处理复杂的参数，还可以简化调用者的代码。

3.1 默认参数¶

我们仍以具体的例子来说明如何定义函数的默认参数。先写一个计算$ x^{2} $的函数：

In [10]:

```
def power(x):
    return x * x
```

当我们调用 `power` 函数时，必须传入有且仅有的一个参数 `x`：

In [12]:

```
power(9)
```

Out[12]:

```
81
```

现在，如果我们要计算 $ x^{3} $ 怎么办？可以再定义一个power3函数，但是如果要计算 $ x^{4} $、$ x^{5} $……怎么办？我们不可能定义无限多个函数。

你也许想到了，可以把 power(x) 修改为 power(x, n)，用来计算 xn ，说干就干：

In [18]:

```
def power(x, n):
    s = 1
    while n > 0:
        n = n - 1
        s = s * x
    return s
```

对于这个修改后的 power 函数，可以计算任意 n 次方：

In [14]:

```
power(3, 5)
```

Out[14]:

```
243
```

但是，旧的调用代码失败了，原因是我们增加了一个参数，导致旧的代码无法正常调用：

In [19]:

```
power(9)
```

```
TypeErrorTraceback (most recent call last)
<ipython-input-19-2317c49f27d0> in <module>()
----> 1 power(9)

TypeError: power() missing 1 required positional argument: 'n'
```

这个时候，默认参数就排上用场了。由于我们经常计算 x^2，所以，完全可以把第二个参数 n 的默认值设定为 2：

In [20]:

```
def power(x, n=2):
    s = 1
    while n > 0:
        n = n - 1
        s = s * x
    return s
```

这样，当我们调用 `power(5)` 时，相当于调用 `power(5, 2)`：

In [21]:

```
power(9)
```

Out[21]:

```
81
```

In [22]:

```
power(5, 2)
```

Out[22]:

```
25
```

而对于 `n > 2` 的其他情况，就必须明确地传入 `n`，比如 `power(5, 3)`。

从上面的例子可以看出，默认参数可以简化函数的调用。设置默认参数时，有几点要注意：

一是必选参数在前，默认参数在后，否则Python 的解释器会报错（思考一下为什么默认参数不能放在必选参数前面）；

二是如何设置默认参数。

当函数有多个参数时，把变化大的参数放前面，变化小的参数放后面。变化小的参数就可以作为默认参数。

使用默认参数有什么好处？最大的好处是能降低调用函数的难度。

3.2 可变参数¶

在Python函数中，还可以定义可变参数。顾名思义，可变参数就是传入的参数个数是可变的，可以是1个、2个到任意个，还可以是0个。

我们以数学题为例子，给定一组数字a，b，c……，请计算 $a^2 + b^2 + c^2 + ……$。

要定义出这个函数，我们必须确定输入的参数。由于参数个数不确定，我们首先想到可以把a，b，c……作为一个 list 或 tuple 传进来，这样，函数可以定义如下：

In [23]:

```
def calc(numbers):
    sum = 0
    for n in numbers:
        sum = sum + n * n
    return sum
```

但是调用的时候，需要先组装出一个 list 或 tuple：

In [24]:

```
calc([1, 2, 3])
```

Out[24]:

```
14
```

In [25]:

```
calc((1, 3, 5, 7))
```

Out[25]:

```
84
```

如果利用可变参数，我们把函数的参数改为可变参数:

In [26]:

```
def calc(*numbers):
    sum = 0
    for n in numbers:
        sum = sum + n * n
    return sum
```

调用函数的方式可以简化成这样：

In [27]:

```
calc(1, 2, 3)
```

Out[27]:

```
14
```

In [28]:

```
calc(1, 3, 5, 7)
```

Out[28]:

```
84
```

定义可变参数和定义 list 或 tuple 参数相比，仅仅在参数前面加了一个 * 号。在函数内部，参数 numbers 接收到的是一个 tuple，因此，函数代码完全不变。但是，调用该函数时，可以传入任意个参数，包括 0 个参数：

In [29]:

```
calc()
```

Out[29]:

```
0
```

如果已经有一个 list 或者 tuple，要调用一个可变参数怎么办？ Python允许你在 list 或 tuple 前面加一个 * 号，把 list 或 tuple 的元素变成可变参数传进去,可以这样做：

In [30]:

```
nums = [1, 2, 3]
calc(*nums)
```

Out[30]:

```
14
```

这种写法相当有用，而且很常见。

3.3 关键字参数¶

可变参数允许你传入 0 个或任意个参数，这些可变参数在函数调用时自动组装为一个 tuple。而关键字参数允许你传入 0 个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个 dict。请看示例：

In [33]:

```
def person(name, age, **kw):
    print('name:', name, 'age:', age, 'other:', kw)
```

函数person除了必选参数 name 和 age 外，还接受关键字参数 kw。在调用该函数时，可以只传入必选参数：

In [34]:

```
person('Michael', 30)
```

```
name: Michael age: 30 other: {}
```

也可以传入任意个数的关键字参数：

In [35]:

```
person('Adam', 45, gender='M', job='Engineer')
```

```
name: Adam age: 45 other: {'job': 'Engineer', 'gender': 'M'}
```

关键字参数有什么用？它可以扩展函数的功能。比如，在person函数里，我们保证能接收到name和age这两个参数，但是，如果调用者愿意提供更多的参数，我们也能收到。试想你正在做一个用户注册的功能，除了用户名和年龄是必填项外，其他都是可选项，利用关键字参数来定义这个函数就能满足注册的需求。

3.4 混合参数¶

在Python中定义函数，可以用必选参数、默认参数、可变参数和关键字参数，这4种参数都可以一起使用，或者只用其中某些，但是请注意，参数定义的顺序必须是：必选参数、默认参数、可变参数和关键字参数。

比如定义一个函数，包含上述4种参数：

In [36]:

```
def func(a, b, c=0, *args, **kw):
    print ('a =', a, 'b =', b, 'c =', c, 'args =', args, 'kw =', kw)
```

在函数调用的时候，Python解释器自动按照参数位置和参数名把对应的参数传进去。

In [37]:

```
func(1, 2, 3, 'a', 'b', x=99)
```

```
a = 1 b = 2 c = 3 args = ('a', 'b') kw = {'x': 99}
```

小结

Python的函数具有非常灵活的参数形态，既可以实现简单的调用，又可以传入非常复杂的参数。

默认参数一定要用不可变对象，如果是可变对象，运行会有逻辑错误！

要注意定义可变参数和关键字参数的语法：

*args是可变参数，args接收的是一个tuple；

**kw是关键字参数，kw接收的是一个dict。

以及调用函数时如何传入可变参数和关键字参数的语法：

可变参数既可以直接传入：func(1, 2, 3)，又可以先组装list或tuple，再通过*args传入：func(*(1, 2, 3))；

关键字参数既可以直接传入：func(a=1, b=2)，又可以先组装dict，再通过**kw传入：func(**{'a': 1, 'b': 2})。

使用*args和**kw是Python的习惯写法，当然也可以用其他参数名，但最好使用习惯用法。

三、匿名函数 lambda ¶

python 使用 lambda 来创建匿名函数。

- lambda只是一个表达式，函数体比def简单很多。
- lambda的主体是一个表达式，而不是一个代码块。仅仅能在lambda表达式中封装有限的逻辑进去。
- lambda函数拥有自己的名字空间，且不能访问自有参数列表之外或全局名字空间里的参数。
- 虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

In [1]:

```
import time
start = time.clock()
fib=lambda n,x=0,y=1:x if not n else fib(n-1,y,x+y)
print (fib(20))
end = time.clock()
print ("read: %f s" % (end - start))
```

```
6765
read: 0.000000 s
```

In [2]:

```
start = time.clock()
fib=lambda n:1 if n<=2 else fib(n-1)+fib(n-2)
print(fib(20))
end = time.clock()
print ("read: %f s" % (end - start))
```

```
6765
read: 0.000000 s
```

四、Python 函数中的多态¶

一个操作的意义取决于被操作对象的类型

In [3]:

```
def times(x,y):
    return x*y
times(2,4)
```

Out[3]:

```
8
```

In [4]:

```
# 传递了与上不同的数据类型
times('Python',4)
```

Out[4]:

```
'PythonPythonPythonPython'
```

In [5]:

```
def intersect(s1,s2):
    return [x for x in s1 if x in s2]
s1 = 'Python'
s2 = 'python'
intersect(s1,s2)
```

Out[5]:

```
['y', 't', 'h', 'o', 'n']
```

In [6]:

```
# 传递了不同的数据类型
intersect([1,2,3],(1,4))
```

Out[6]:

```
[1]
```

五、递归¶

（1）递归就是在过程或函数里调用自身；

（2）在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口。

递归算法一般用于解决三类问题：（1）数据的定义是按递归定义的。（比如Fibonacci函数）（2）问题解法按递归算法实现。（回溯）（3）数据的结构形式是按递归定义的。（比如树的遍历，图的搜索）

递归的缺点：递归算法解题的运行效率较低。在递归调用的过程当中系统为每一层的返回点、局部量等开辟了栈来存储。递归次数过多容易造成栈溢出等。

示例：斐波那契数列

斐波那契数列由十三世纪意大利数学家斐波那契发现。数列中的一系列数字常被人们称之为神奇数奇异数。具体数列为：0，1，1，2，3，5，8，13，21，34，55，89，144，233等，从该数列的第三项数字开始，每个数字等于前两个相邻数字之和。而斐波那契数列中相邻两项之商就接近黄金分割数0.618，与这一数字相关的0.191、0.382、0.5和0.809等数字就构成了股市中关于市场时间和空间计算的重要数字。

在金融市场的分析方法中，斐波那契数字频频出现。例如，在波浪理论中，一轮牛市行情可以用1个上升浪来表示，也可以用5个低一个层次的小浪来表示，还可继续细分为21个或89个小浪；在空间分析体系中，反弹行情的高度通常是前方下降趋势幅度的0.382、0.5、0.618；回调行情通常是前方上升趋势的0.382、0.5和0.618。

In [8]:

```
def fib(num):
    result=[0,1]
    for i in range(num-2):
        result.append(result[-2]+result[-1])
    return result
print (fib(15))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

In [10]:

```
import time
start = time.clock()
def fib(n):
    if n<=2:return 1
    else:
        return fib(n-1)+fib(n-2)
print (fib(20))
end = time.clock()
print ("read: %f s" % (end - start))
```

```
6765
read: 0.000000 s
```

In [ ]:

```
start = time.clock()
def fib(n):
    return 1 and n<=2 or fib(n-1)+fib(n-2)
print (fib(20))
end = time.clock()
print ("read: %f s" % (end - start))
```

NumPy数组的优势¶

NumPy数组通常是由相同种类的元素组成的，既数组中的数据项的类型必须一致。

In [1]:

```
import numpy as np
```

In [2]:

```
a = np.arange(5)
a.dtype
```

Out[2]:

```
dtype('int64')
```

以上数组的数据类型为int64，除了知道数据类型之外，还要注意其形状，这一点非常重要

In [3]:

```
a
```

Out[3]:

```
array([0, 1, 2, 3, 4])
```

In [4]:

```
a.shape
```

Out[4]:

```
(5,)
```

如你所见，该向量有5个元素，他们的值分别是从0到4，该数组的shape属性是一个元组,存放的是数组在每一个维度的长度。

创建多维数组¶

既然我们已经知道了创建向量的方法，下面开始学习如何建设多维NumPy数组。生成矩阵后，再来看它的形状，

In [5]:

```
m = np.array([np.arange(2),np.arange(2)])
m
```

Out[5]:

```
array([[0, 1],
       [0, 1]])
```

In [6]:

```
m.shape
```

Out[6]:

```
(2, 2)
```

选择NumPy数组元素¶

有时，我们可能想从数组中选择指定的元素。如何做到这一点呢？不妨从创建一个2行2列的矩阵着手：

In [7]:

```
a = np.array([[1,2],[3,4]])
```

上面的矩阵是通过向array()函数传递一个由列表组成的列表得到的，接下来，我们要逐个选择矩阵的各个元素，代码如下所示。别忘了，下标是从0开始的：

In [8]:

```
a[0,0]
```

Out[8]:

```
1
```

In [9]:

```
a[0,1]
```

Out[9]:

```
2
```

In [10]:

```
a[1,0]
```

Out[10]:

```
3
```

In [11]:

```
a[1,1]
```

Out[11]:

```
4
```

可见，选择数组元素是一件非常简单的事情，对于数组a，只要通过a[m,n]的形态，就能访问数组内的元素，其中m和n为数组元素的下标。

NumPy 的数值类型¶

Python自身虽然支持整型、浮点型和复数型，但对于科学计算来说，还远远不够。现实中，我们仍然需要更多的数据类型，来满足在精度和储存大小方面的各种不同的要求。为此，NumPy提供了更加丰富的数据类型。注意，NumPy跟数学运算有关的数据类型的名称都以数字结尾。而这个数字指示了该类型的变量所占用的二进制位数。

每一种数据类型都有相应的转换函数，如下所示：

In [12]:

```
float(42)
```

Out[12]:

```
42.0
```

In [13]:

```
int(42.0)
```

Out[13]:

```
42
```

In [14]:

```
bool(42)
```

Out[14]:

```
True
```

In [15]:

```
bool(0)
```

Out[15]:

```
False
```

In [16]:

```
bool(42.0)
```

Out[16]:

```
True
```

In [17]:

```
float(True)
```

Out[17]:

```
1.0
```

In [18]:

```
float(False)
```

Out[18]:

```
0.0
```

许多函数都带有一个指定数据类型的参数，该参数通常是可选的：

In [19]:

```
np.arange(7,dtype=float)
```

Out[19]:

```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.])
```

谨记：不允许把复数类型转化成整型，当你企图进行这种转换时，将会触发TypeError错误，就像下面这样：

In [20]:

```
float(42.0+1.j)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-cc01ef639a6a> in <module>()
----> 1 float(42.0+1.j)

TypeError: can't convert complex to float
```

同样的，也不允许把复数转化成浮点数。另外，复数的分量j是其虚部的系数。

一维数组的切片与索引¶

一维NumPy数组的切片操作与Python列表的切片一样。下面先来定义包括0、1、2，直到8的一个数组，然后通过指定下标3到7来选择该数组的部分元素，这实际上就是提取数组中值为3到6的那些元素。

In [22]:

```
a = np.arange(9)
a[3:7]
```

Out[22]:

```
array([3, 4, 5, 6])
```

可以用下标选择元素，下标范围从0到7，并且下标每次递增2

In [23]:

```
a[:7:2]
```

Out[23]:

```
array([0, 2, 4, 6])
```

恰如使用Python那样，也可用负值下标来反转数组：

In [24]:

```
 a[::-1]
```

Out[24]:

```
array([8, 7, 6, 5, 4, 3, 2, 1, 0])
```

处理数组形态¶

前面，我们已经学过了reshape()函数，实际上，除了数组形状的调整外，数组的扩充也是一个经常碰到的乏味工作。比如，可以想象一下将多维数组转换成一维数组时的情形。

In [25]:

```
print('In:b = arange(24).reshape(2,3,4)')
b = np.arange(24).reshape(2,3,4)
print('In:b')
print(b)
```

```
In:b = arange(24).reshape(2,3,4)
In:b
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

In [26]:

```
print('In:b.ravel()')
print(b.ravel())
```

```
In:b.ravel()
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

In [27]:

```
print('In:b.flatten()')
print(b.flatten())
```

```
In:b.flatten()
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

In [28]:

```
print('In:b.shape = (6,4)')
b.shape = (6,4)
print('In:b')
print(b)
```

```
In:b.shape = (6,4)
In:b
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

In [29]:

```
print('In:b.transpose()')
print(b.transpose())
```

```
In:b.transpose()
[[ 0  4  8 12 16 20]
 [ 1  5  9 13 17 21]
 [ 2  6 10 14 18 22]
 [ 3  7 11 15 19 23]]
```

In [30]:

```
print('In:b.resize((2,12))')
b.resize((2,12))
print('In:b')
print(b)
```

```
In:b.resize((2,12))
In:b
[[ 0  1  2  3  4  5  6  7  8  9 10 11]
 [12 13 14 15 16 17 18 19 20 21 22 23]]
```

拆解：可以用ravel()函数将多维数组变成一维数组

In [31]:

```
b.shape = (6,4)
b
```

Out[31]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [32]:

```
b.ravel()
```

Out[32]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23])
```

拉直：flatten()函数的名字取得非常贴切，其功能与ravel()相同。可是，flatten()返回的是真实的数组，需要分配新的内存空间；而ravel()函数返回的只是数组的视图。这意味着，我们可以像下面这样的直接操作数组：

In [33]:

```
b.flatten()
```

Out[33]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23])
```

用元组指定数组形状：除reshape()函数外，还可以用元组来轻松定义数组的形状：

转置：在线性代数中，矩阵的转置操作非常常见。转置是一种数据变换方法，对于二维表而言，转置就意味着行变成列，同时列变成行。转置也可以通过下列代码完成：

In [34]:

```
b.transpose()
```

Out[34]:

```
array([[ 0,  4,  8, 12, 16, 20],
       [ 1,  5,  9, 13, 17, 21],
       [ 2,  6, 10, 14, 18, 22],
       [ 3,  7, 11, 15, 19, 23]])
```

调整大小：函数resize()的作用类似于reshape()，但是会改变所作用的数组：

In [35]:

```
b.resize((2,12))
b
```

Out[35]:

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]])
```

堆叠数组¶

从深度看，数组既可以横向叠放，也可以竖向叠放，为此，可以使用vstack()、dstack()、hstack()、column_stack()、row_stack()和concatenate()等函数。在此之前，我们先要建立某些数组：

In [36]:

```
a = np.arange(9).reshape(3,3)
a
```

Out[36]:

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [37]:

```
b= 2*a
b
```

Out[37]:

```
array([[ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16]])
```

水平叠加：先介绍水平叠加方式，即用元组确定ndarrays数组的形状，然后交由hstack（）函数来码放这些数组。

In [38]:

```
np.hstack((a,b))
```

Out[38]:

```
array([[ 0,  1,  2,  0,  2,  4],
       [ 3,  4,  5,  6,  8, 10],
       [ 6,  7,  8, 12, 14, 16]])
```

用concatenate（）函数也能达到相同的效果。

In [39]:

```
np.concatenate((a,b),axis = 1)
```

Out[39]:

```
array([[ 0,  1,  2,  0,  2,  4],
       [ 3,  4,  5,  6,  8, 10],
       [ 6,  7,  8, 12, 14, 16]])
```

垂直叠加：使用垂直叠加方法时，先要构建一个元祖，然后将元祖交给vstack（）函数来码放数组。

In [40]:

```
np.vstack((a,b))
```

Out[40]:

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16]])
```

当参数axis置0时，concatenate（）函数也会得到相同的效果。实际上，这是该参数的缺省值：

In [41]:

```
np.concatenate((a,b),axis = 0)
```

Out[41]:

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16]])
```

深度叠加：除此之外，还有一种深度叠加方法，这要用到dstack（）函数和一个元组。这种方法是沿着第三个坐标轴（纵向）的方向来叠加一摞数组。举例来说，可以在一个图像数据的二维数组上叠加另一幅图像的数据。

In [42]:

```
np.dstack((a,b))
```

Out[42]:

```
array([[[ 0,  0],
        [ 1,  2],
        [ 2,  4]],

       [[ 3,  6],
        [ 4,  8],
        [ 5, 10]],

       [[ 6, 12],
        [ 7, 14],
        [ 8, 16]]])
```

列式堆叠：columns_stack()函数以列方式对一维数组进行堆叠。

In [43]:

```
oned =np.arange(2)
oned
```

Out[43]:

```
array([0, 1])
```

In [44]:

```
twice_oned = 2*oned
twice_oned
```

Out[44]:

```
array([0, 2])
```

In [45]:

```
np.column_stack((oned,twice_oned))
```

Out[45]:

```
array([[0, 0],
       [1, 2]])
```

用这种方法堆叠二维数组时，过程类似于hstack（）函数。

In [46]:

```
np.column_stack((a,b))
```

Out[46]:

```
array([[ 0,  1,  2,  0,  2,  4],
       [ 3,  4,  5,  6,  8, 10],
       [ 6,  7,  8, 12, 14, 16]])
```

In [47]:

```
np.column_stack((a,b)) == np.hstack((a,b))
```

Out[47]:

```
array([[ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True]], dtype=bool)
```

行式堆叠：同时，NumPy自然也有以行方式对数组进行堆叠的函数，这个用于一维数组的函数名为row_stack(),它将数组作为行码放到二维数组中：

In [48]:

```
np.row_stack((oned,twice_oned))
```

Out[48]:

```
array([[0, 1],
       [0, 2]])
```

对于二维数组，row_stack()函数相当于vstack()函数：

In [49]:

```
np.row_stack((a,b))
```

Out[49]:

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16]])
```

In [50]:

```
np.row_stack((a,b)) == np.vstack((a,b))
```

Out[50]:

```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]], dtype=bool)
```

拆分NumPy数组¶

可以从纵向，横向和深度方向来拆分数组，相关函数有hsplit(),vsplit(),dsplit()和split()。我们即可以把数组分成相同的形状的数组，也可以从规定的位置开始切取数组。

横向拆分：对于一个3行3列数组，可以沿着横轴方向将其分解为3部分，并且各部分的大小和形状完全一致。

In [51]:

```
a
```

Out[51]:

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [52]:

```
np.hsplit(a,3)
```

Out[52]:

```
[array([[0],
        [3],
        [6]]), array([[1],
        [4],
        [7]]), array([[2],
        [5],
        [8]])]
```

这个相当于调用了参数axis = 1的split（）函数：

In [53]:

```
np.split(a,3,axis = 1)
```

Out[53]:

```
[array([[0],
        [3],
        [6]]), array([[1],
        [4],
        [7]]), array([[2],
        [5],
        [8]])]
```

纵向拆分：同理

In [54]:

```
np.vsplit(a,3)
```

Out[54]:

```
[array([[0, 1, 2]]), array([[3, 4, 5]]), array([[6, 7, 8]])]
```

In [55]:

```
np.split(a,3,axis = 0)
```

Out[55]:

```
[array([[0, 1, 2]]), array([[3, 4, 5]]), array([[6, 7, 8]])]
```

深向拆分：dsplit()函数会沿着深度方向分解数组。下面以秩为3的数组为例进行说明：

In [56]:

```
c = np.arange(27).reshape(3,3,3)
c
```

Out[56]:

```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]],

       [[18, 19, 20],
        [21, 22, 23],
        [24, 25, 26]]])
```

In [57]:

```
np.dsplit(c,3)
```

Out[57]:

```
[array([[[ 0],
         [ 3],
         [ 6]],

        [[ 9],
         [12],
         [15]],

        [[18],
         [21],
         [24]]]), array([[[ 1],
         [ 4],
         [ 7]],

        [[10],
         [13],
         [16]],

        [[19],
         [22],
         [25]]]), array([[[ 2],
         [ 5],
         [ 8]],

        [[11],
         [14],
         [17]],

        [[20],
         [23],
         [26]]])]
```

NumPy数组的属性：¶

下面举例说明NumPy数组各种属性的详细用法。

In [58]:

```
b = np.arange(24).reshape(2,12)
print('In:b')
print(b)
```

```
In:b
[[ 0  1  2  3  4  5  6  7  8  9 10 11]
 [12 13 14 15 16 17 18 19 20 21 22 23]]
```

In [59]:

```
print('In:b.ndim')
print(b.ndim)
```

```
In:b.ndim
2
```

In [60]:

```
print('In:b.size')
print(b.size)
```

```
In:b.size
24
```

In [61]:

```
print('In:b.itemsize')
print(b.itemsize)
```

```
In:b.itemsize
8
```

SciPy¶

SciPy是世界上著名的Python开源科学计算库，建立在Numpy之上。它增加的功能包括数值积分、最优化、统计和一些专用函数。

本篇文档包括以下内容：
1、文件的输入/输出；
2、统计；
3、信号处理；
4、最优化；
5、插值

1 scipy .io文件的输入和输出¶

In [1]:

```
from scipy import io as spio
import numpy as np
```

In [2]:

```
# 保存文件
a = np.ones((3, 3))
spio.savemat('file.mat', {'a': a})
#导入文件
data = spio.loadmat('file.mat', struct_as_record=True)
data
```

Out[2]:

```
{'__globals__': [],
 '__header__': b'MATLAB 5.0 MAT-file Platform: posix, Created on: Wed Mar 15 06:47:47 2017',
 '__version__': '1.0',
 'a': array([[ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.]])}
```

载入txt文件：numpy.loadtxt()/numpy.savetxt()
智能导入文本/csv文件：numpy.genfromtxt()/numpy.recfromcsv()

高速，有效率但numpy特有的二进制格式：numpy.save()/numpy.load()

2 统计¶

2.1 分析随机数¶

In [3]:

```
import scipy.stats as stats
```

In [9]:

```
# 生成正态分析的随机数
generated = stats.norm.rvs(size = 900)
generated
```

Out[9]:

```
array([  2.91995561e-01,   4.23394920e-01,   1.51373530e+00,
         4.20375684e-01,  -1.84395848e-01,   1.47141997e-02,
         1.30986792e+00,   3.02135017e-01,   5.65506403e-01,
        -2.50445465e-01,   7.14659560e-01,  -3.54103756e-01,
         8.18393064e-01,   1.01818146e+00,  -1.83431799e+00,
        -5.31036609e-01,   3.96669365e-01,  -1.17554640e+00,
         2.88267371e-01,  -8.37390352e-01,   3.67528321e-01,
         5.28392496e-01,   4.59818832e-01,  -4.75492118e-01,
        -1.03408508e-01,   5.99794845e-01,  -1.97344168e+00,
        -8.32686424e-01,   1.10844196e+00,  -8.58263017e-01,
         5.07691243e-01,  -3.39394230e-01,   6.35276755e-01,
         1.91207057e-01,  -3.75750184e-01,  -9.86426169e-01,
        -7.34533938e-01,   6.22772785e-01,  -2.39779856e-01,
        -1.13794163e+00,   1.15743826e+00,  -8.84113057e-01,
         5.34886518e-01,  -1.53075225e+00,  -1.38786559e-01,
        -8.42618235e-01,  -4.61052835e-02,  -9.29650255e-01,
        -2.02122984e-01,  -1.82490183e-01,   5.52516981e-01,
        -6.93633624e-01,   1.14161306e-01,  -1.41196594e+00,
        -1.26506042e+00,   4.33787880e-01,   1.21457423e+00,
         6.33349592e-01,  -1.51573414e+00,   4.48902595e-01,
         1.13461537e+00,  -1.05732690e+00,   2.79508112e-01,
        -9.62079366e-01,   1.56799705e-01,  -1.03382155e-01,
        -9.71018227e-01,  -6.42283670e-01,   1.59596080e+00,
         3.74831614e-01,  -1.20843915e+00,   5.31669253e-01,
        -1.04132208e+00,   1.11071247e+00,   1.70792703e+00,
         4.86504027e-01,   1.38700100e-02,   8.79488592e-01,
        -6.23097850e-01,   3.36863717e-02,   4.11966618e-01,
        -5.58770450e-01,   1.70043439e+00,   6.65604548e-01,
         1.61423011e-01,  -2.79699897e-01,   1.16370529e+00,
         1.84674828e+00,  -6.25982430e-01,  -1.20533925e+00,
        -1.52051751e+00,  -5.47661316e-01,  -1.04164033e-01,
         3.44763447e-01,   4.25307274e-01,  -1.47418724e+00,
        -2.71553332e-01,   6.07998277e-01,  -1.16046091e+00,
        -1.49786062e+00,   2.33303965e-02,  -7.99021712e-01,
         1.55521561e+00,   1.04839313e+00,   2.35028835e-01,
         4.06620559e-01,  -2.37676781e+00,  -5.32362343e-01,
         6.65978168e-01,  -1.88208011e+00,   9.92647528e-01,
        -2.61456590e-02,   2.18597106e+00,   7.27773229e-01,
        -2.18455484e-01,   2.61960522e-01,  -5.64735241e-01,
        -2.75337183e-01,  -5.68348651e-02,   1.28498172e-01,
         9.86048697e-01,   3.27754689e-02,   9.97539226e-01,
         6.75352797e-01,  -4.85782917e-01,   4.75604589e-01,
         5.07875601e-01,  -4.47473058e-01,  -1.10468912e-02,
         3.64075234e-01,  -3.50862594e-01,  -6.82866249e-01,
        -5.51598973e-01,   1.74597091e-01,  -4.98206444e-01,
         2.54783607e+00,   1.67124799e-01,   1.77811008e+00,
        -5.38285765e-01,   5.02347283e-01,   2.57562859e+00,
         3.59365915e-01,   3.22325088e-01,   5.29080096e-01,
         1.93015426e+00,   7.94177575e-01,   8.38649032e-01,
        -1.24143551e+00,  -2.11955883e+00,   7.07511775e-01,
         6.79250959e-02,   1.44672617e+00,  -3.65403805e-01,
         7.19341024e-01,  -1.35689882e+00,  -7.68147942e-01,
        -9.34196053e-01,   1.96570805e+00,  -5.22093312e-01,
        -6.14274711e-01,  -1.00432338e+00,  -2.59400416e-01,
         1.91671021e-01,  -8.45970027e-01,  -2.26076894e+00,
        -7.55856426e-01,   2.63451620e-01,  -1.01197676e+00,
        -9.29121394e-01,   5.46592677e-01,  -2.49677269e-01,
         5.44884347e-01,   9.14766410e-01,  -1.63423532e-01,
         1.15467336e+00,  -1.97342697e+00,   9.48454655e-01,
        -6.33190860e-01,  -3.47984841e-01,  -1.72533889e+00,
         6.32462647e-01,  -1.48487434e-01,  -3.95927041e-01,
         6.07335677e-01,  -1.90501382e+00,  -8.95157261e-01,
```

```
 8.37732184e-01,   1.99336148e-01,   2.26820011e-01,
-4.59827655e-01,  -4.41173001e-01,   4.67073858e-01,
-5.38511290e-01,   1.55537333e+00,   1.90178328e+00,
 6.37115599e-02,  -9.99639822e-01,  -1.45572305e+00,
-1.46465190e+00,  -1.44648823e+00,  -3.01083650e-01,
-1.65503385e+00,  -6.02364884e-01,  -1.01994997e+00,
-8.05571012e-01,  -7.68327328e-01,   7.66501635e-01,
-8.44282850e-01,  -1.91511269e+00,  -1.51553597e+00,
 4.88276206e-01,  -1.15016683e-01,   1.05071288e+00,
 4.38049898e-01,  -1.25013703e+00,   1.18294853e+00,
 9.23186349e-01,   7.90603159e-01,  -3.20429751e-01,
-1.16474516e+00,   3.37262790e-01,  -9.02638825e-01,
 3.87631478e-01,   8.90868788e-01,  -8.93111056e-02,
 7.55225987e-01,  -2.02890647e-01,  -1.27159439e+00,
 5.09285062e-01,   5.70718877e-02,   4.82013836e-01,
 9.45224113e-01,  -6.89211538e-01,   7.57038029e-01,
 1.69194907e+00,  -5.20059699e-01,   3.51822316e-01,
-1.43756064e+00,   1.18780271e+00,   3.19834196e-01,
-9.87963594e-03,  -9.18985833e-02,  -7.79765247e-01,
-3.57494229e-01,   1.40374771e+00,  -2.95799856e-01,
 1.37623442e+00,   8.36728810e-01,  -1.66195444e+00,
-1.02979350e+00,   3.88040035e-01,  -5.93790522e-01,
-1.86725144e+00,  -5.28617010e-01,  -1.06596511e+00,
-1.91025535e+00,   1.04931968e+00,   2.05354643e-01,
 2.94238255e+00,   9.93771874e-01,   1.94350919e+00,
 9.35079222e-01,   4.22194333e-01,  -4.95021643e-02,
-1.02041618e+00,  -6.98780167e-01,  -1.85659646e+00,
 7.32098100e-01,  -4.05194944e-01,  -4.25548340e-01,
 8.86905188e-02,  -1.94405388e-02,   1.09956415e+00,
-1.16668429e-03,  -2.77397016e+00,   4.99996550e-01,
-3.15507837e-01,   1.92255350e-01,  -4.85058545e-02,
 1.07745731e-01,   1.72922518e+00,  -1.05700687e+00,
 2.94405553e-02,  -1.98347462e+00,  -1.69068231e+00,
 7.15235168e-01,  -1.09478287e+00,  -1.52698714e+00,
 1.04016666e+00,  -8.38268376e-01,   8.81294677e-02,
-6.88028158e-01,   1.44079068e+00,  -2.31292527e-01,
 4.77711605e-01,   1.22950178e+00,  -3.00154113e-02,
 8.90433857e-01,  -2.95976354e-01,  -8.55171013e-01,
 1.12035137e+00,   2.54574196e+00,  -2.07124792e+00,
-3.17564604e-01,   5.60210270e-01,  -8.32660461e-01,
 5.31686732e-01,   6.94718480e-01,  -1.36117377e+00,
-9.12002352e-01,   5.78162834e-01,   8.85754309e-01,
-2.76389009e-01,  -9.57810539e-01,  -1.35285005e+00,
-6.59562241e-01,   3.51035160e-01,  -2.98034930e-01,
-1.89834429e-01,  -6.83326742e-01,   3.17170732e-01,
 1.88102707e+00,  -2.15807168e-01,  -1.25498509e+00,
-7.67792483e-01,   1.03381781e+00,   5.45088896e-01,
 7.85354217e-02,   1.68488155e+00,   1.19966821e+00,
-1.43313416e+00,   4.25692854e-01,  -4.67574231e-01,
-1.38909946e+00,  -3.16409933e-01,   2.53439280e+00,
-6.35589802e-01,  -2.24045479e+00,  -1.25297077e+00,
 8.71386267e-02,  -8.96052541e-01,   4.47937846e-01,
 5.49416290e-01,   5.51129268e-02,   1.44624264e+00,
 2.26652048e+00,  -8.60302003e-01,  -6.94749904e-01,
-7.61323474e-01,   5.36414866e-01,  -1.00750103e+00,
 2.26881091e-01,  -1.11916335e-01,   9.55137697e-02,
-1.48777556e-01,   6.81146837e-01,   1.60159205e+00,
 6.90501130e-01,  -1.11838518e-01,  -2.39000282e-01,
-1.51318841e+00,  -3.72614297e-01,   1.91947275e-03,
-6.29287299e-01,  -6.36882089e-02,  -4.61046003e-02,
 4.28669644e-01,  -9.73689177e-01,   1.60372580e+00,
-3.02382830e-01,  -1.74423270e+00,  -5.65691501e-01,
-6.46891697e-01,   1.19726155e+00,  -3.05857433e-01,
 3.33760152e-01,   1.30723043e-02,  -1.29311743e+00,
 1.77452158e+00,   1.14396517e+00,   3.02926073e-02,
 1.25867950e+00,  -6.39505410e-01,   1.85981511e+00,
 1.42072903e+00,  -2.01662580e+00,  -7.45011233e-01,
-1.44095905e+00,  -5.42851318e-01,  -5.16875155e-01,
-3.81903026e-01,  -3.99804013e-01,  -1.75457850e+00,
-1.52458945e+00,   4.78978119e-01,  -4.25635868e-01,
 8.11233019e-02,   8.10523048e-01,   2.82638674e-01,
 2.94027975e-01,  -1.19723507e+00,   6.17343133e-01,
-2.92977095e-01,  -1.02550583e-01,   2.51344594e-01,
 4.02551535e-01,   8.41419984e-01,   3.08284248e-01,
-1.28898076e+00,  -8.30079785e-03,  -1.94602583e+00,
 1.43469096e+00,   1.93354815e+00,   7.71414017e-01,
 5.41289795e-01,  -1.00032704e+00,   6.00377503e-01,
 3.94377979e-01,   3.65937223e-01,  -7.92549889e-01,
-8.01496059e-01,   4.85722377e-01,   2.31439428e-01,
 6.30215223e-01,  -3.07866853e-01,  -1.86052044e+00,
 1.20320890e+00,   3.79491993e-01,   1.18208781e+00,
 1.07032289e-01,   1.78503688e+00,   1.38855255e+00,
```

```
-4.07074179e-01,   1.69488610e-01,  -7.08733745e-01,
 2.03322749e+00,   1.54258366e+00,  -6.68710364e-01,
 1.09401195e+00,  -3.33774082e-01,   3.42718990e-01,
 2.58697046e-01,  -2.51644126e+00,   7.93648773e-01,
 2.91861881e-01,   5.66450937e-01,  -1.18350194e+00,
 4.07757155e-01,   1.88978404e-01,  -5.30166707e-02,
 3.59709433e-01,  -7.54336929e-01,  -9.38809839e-04,
 9.46623248e-01,  -6.59694800e-01,  -7.15569399e-01,
-1.54970151e+00,  -2.95394518e-01,   1.88563324e+00,
 1.75979347e+00,   3.66580654e-02,   1.39068720e+00,
-3.14246938e-01,  -4.45801605e-01,   6.73664592e-01,
 8.71975917e-01,   4.44875247e-01,  -1.19499471e+00,
 3.43948456e-01,   7.61230269e-01,  -2.98279714e-02,
 8.39801194e-02,   2.31128517e-01,  -4.80195898e-01,
 9.95942233e-02,  -1.28336408e+00,  -7.22953546e-02,
 2.29933763e+00,  -7.71905857e-01,   3.31509086e-02,
 1.63312408e-01,   8.39395315e-01,  -9.58912346e-02,
 4.58373033e-01,  -5.30596688e-01,   3.72705501e-01,
-2.50615745e-01,   1.96232099e+00,   6.91774600e-01,
-1.44025462e+00,   2.70777672e-01,  -2.77114581e+00,
-1.84034167e+00,  -3.89198223e-01,   7.90729391e-01,
 1.97643525e+00,  -1.78388542e-01,  -2.45915837e+00,
-1.15586746e+00,   1.16123586e+00,  -1.30091895e-01,
-1.67656675e+00,  -4.82424896e-01,   1.14265538e+00,
-6.15804935e-01,   7.59385933e-01,   5.11107641e-01,
-5.21127893e-01,   1.29840819e+00,   1.56946418e+00,
-8.40527743e-01,   6.51606205e-02,   2.24484597e-01,
 1.25383540e+00,   8.71226682e-01,  -2.22602395e+00,
 1.66709032e+00,   8.65449950e-01,  -4.88354706e-01,
 1.44633830e+00,  -7.03362113e-01,  -1.77401341e+00,
-1.21469380e+00,   6.32348052e-01,   2.68036734e-01,
 2.97525508e-01,   4.22124198e-01,  -1.47335851e+00,
-2.40170976e-01,  -7.56273078e-01,   6.95642532e-02,
-1.47857252e+00,  -3.63157118e-01,  -2.84013136e-01,
 1.68955173e+00,   4.54060430e-01,   9.32620228e-01,
 7.01246564e-01,  -1.61682529e+00,   9.00878464e-02,
-1.79274948e-01,  -2.44066931e-01,  -1.70424947e-01,
-9.48292405e-01,   6.33594300e-01,  -7.07821112e-02,
-1.08062797e+00,   2.15684963e+00,   7.33924846e-01,
-2.90533035e-01,   1.26397597e-01,   1.49869712e+00,
-1.96448195e+00,   5.73685541e-01,  -1.42726158e-01,
 3.85773636e-01,  -1.58563144e+00,   6.04552996e-01,
 8.69352876e-01,   7.67391012e-02,   1.44068945e-01,
-6.50147870e-01,  -8.08179839e-01,   1.19762932e+00,
 3.78034184e-01,   1.98608217e-01,   4.89567438e-01,
-1.16967972e+00,  -6.93606523e-01,  -5.39348944e-01,
 5.63370371e-02,  -1.21150246e+00,  -2.21799185e-01,
-8.71107357e-01,  -6.19362699e-01,  -7.58579337e-01,
-1.23508663e+00,  -8.06427971e-01,   5.12018260e-01,
 3.79676231e-01,  -8.36079356e-01,   1.12765265e+00,
 1.47735971e+00,   6.69660072e-01,  -2.04057299e-01,
-6.61346312e-01,   1.01975742e+00,  -4.81195044e-01,
 1.33587735e-01,  -3.93705388e-01,  -3.17981117e-01,
-1.08655243e+00,  -5.22495781e-01,  -3.05390975e-01,
 8.57244849e-01,  -1.10511575e+00,   8.35658217e-01,
-6.99244846e-01,  -5.53952318e-01,  -7.38175031e-01,
 1.31530684e+00,   1.44175622e+00,   9.39697337e-01,
 3.46863060e-01,   1.41321504e+00,   5.10789147e-01,
 1.31812048e+00,  -8.21916393e-01,   1.42010915e+00,
 3.60781510e-01,  -1.08623949e+00,  -1.69844853e+00,
-4.70276035e-01,  -7.10063605e-02,  -6.19846867e-01,
 7.80308032e-01,   8.56190018e-02,  -1.76932440e-01,
-8.61069590e-01,   4.56431421e-01,  -1.11009180e+00,
-9.64914205e-01,   4.80991563e-01,   6.30947710e-02,
 8.73772590e-01,  -3.07968156e-01,  -6.91809356e-01,
-2.21885382e+00,   5.12569244e-02,   6.40808807e-01,
-3.94709302e-01,   1.55626509e-01,  -6.23203167e-01,
 6.22951969e-01,   9.46187373e-01,  -1.31414943e+00,
 1.12578169e+00,   2.96379205e-01,  -8.34327048e-01,
-4.76344059e-01,  -1.11882889e-01,  -1.31734956e+00,
 9.01391723e-02,  -2.60386094e-01,   1.12519035e+00,
-8.56750769e-01,  -8.07724831e-01,  -9.19691927e-01,
-1.54537887e-01,   8.99159621e-01,  -1.32420988e+00,
-1.93096798e-01,  -3.39091205e-01,  -6.47227983e-01,
-8.30776346e-01,  -2.24398370e+00,   1.47689039e+00,
 1.07728122e-01,  -8.84660266e-01,   1.71002846e-01,
-1.02263287e+00,  -9.45933175e-01,   1.03253478e-01,
 1.17498962e+00,  -9.30160020e-01,  -3.34858197e-01,
-2.24496864e+00,  -8.02485465e-01,   5.66894262e-01,
-1.64575055e-02,  -3.62372533e-01,   1.01481327e+00,
 1.18543681e-01,  -1.02292163e+00,   1.14868753e+00,
 1.24703968e-01,   1.32776670e+00,  -1.64445184e+00,
```

```
 6.43965362e-01,   1.77276713e+00,  -1.59837766e+00,
-1.20352682e+00,  -1.19830510e+00,  -1.06109252e+00,
 1.24609755e+00,  -1.41212661e+00,   7.38484823e-01,
 9.65754904e-01,  -7.57725287e-01,  -4.72981374e-01,
-5.38287189e-01,  -1.03781402e+00,   2.07377483e+00,
-7.40714794e-01,   1.63400553e+00,   1.12577127e+00,
-4.84333930e-01,  -6.56376330e-01,   9.11918983e-01,
-3.62675952e-01,   3.33312799e-01,  -1.26179812e+00,
 8.57636087e-01,   2.71919312e-01,  -1.17561954e+00,
 2.73632257e-01,  -2.36852513e-01,   1.41591336e+00,
-7.45501396e-01,  -1.48882181e+00,  -1.82731499e-01,
 2.67183576e+00,  -4.11266525e-01,   5.60018984e-01,
 3.01901677e-01,   3.55424487e-01,   6.57439185e-01,
 5.28834282e-01,  -1.71861292e+00,   5.83918660e-01,
-4.50316925e-01,  -1.50458318e+00,  -1.16338798e+00,
 7.29694514e-01,   5.32289993e-01,   1.43043276e+00,
 6.04860836e-01,  -8.70443575e-01,   1.86401749e-01,
-2.84195622e-01,  -1.09458040e+00,  -5.20474046e-01,
-3.15505939e-01,   3.63550794e-01,   1.19004107e+00,
 1.03042812e-01,   2.27404628e-01,  -1.17627321e-02,
 1.24803793e+00,  -1.79191799e-01,  -1.21301000e+00,
-1.06838097e-01,  -1.94812704e-01,   1.04147692e+00,
-1.52034640e+00,   2.00930069e+00,   1.64931997e+00,
 6.24001180e-01,   3.05858238e-01,   2.53841059e-01,
 9.09088800e-01,  -7.20850805e-01,  -1.04387930e+00,
-2.12975889e-01,  -6.27280058e-01,  -7.60173258e-01,
-1.22059747e-01,  -5.92602368e-01,   7.00765104e-01,
-1.02328372e+00,   2.27484699e-01,  -2.31281555e-01,
-1.88977680e+00,   6.48940156e-01,  -6.68271147e-01,
 3.49167134e-01,   2.16110036e-02,   2.92004848e-02,
-1.72862274e+00,  -2.61853541e-01,   1.16760916e+00,
-3.74831617e-01,  -5.99547985e-01,   7.38232889e-01,
 1.04677554e+00,  -1.95062931e+00,  -2.04278288e+00,
 5.84497477e-02,  -8.07539793e-01,  -2.19364752e+00,
-5.71550725e-01,  -7.26571509e-01,   1.43371143e+00,
-5.18859730e-02,   1.87108144e+00,  -1.24676105e-01,
-4.11427629e-01,  -5.37066062e-01,  -4.42155250e-01,
-9.30856991e-01,   7.37486195e-01,   8.98246411e-01,
 1.83775374e+00,  -3.05048060e-01,   7.94801709e-02,
-2.92540689e-02,   3.32197244e-02,   6.73491566e-01,
-7.05561898e-01,   2.68577937e-01,  -6.66434478e-01,
 1.41443303e+00,   5.12422072e-01,  -4.48721313e-01,
 7.36011003e-01,  -2.01417965e+00,  -5.23674297e-01,
-1.31894517e+00,  -1.29212194e+00,   7.37425637e-01,
 1.46026959e+00,  -5.02240599e-01,   6.51977147e-01,
 1.13412383e+00,   9.65254204e-02,  -5.97981098e-01,
-1.45283081e+00,  -1.44559140e+00,   5.02674699e-01,
-8.22388619e-01,  -1.36865798e-01,   1.41644920e+00,
-2.15360897e+00,   2.84041508e-02,   3.18504981e-01,
-1.42761824e+00,  -4.66631461e-02,  -1.97584722e-01,
-8.04854159e-01,   9.02099400e-02,  -2.00101306e+00,
-1.75629102e-01,   1.07468515e+00,  -5.15195137e-01,
 2.17512336e-01,  -1.02887724e+00,   8.36688610e-01,
 4.81733615e-01,   1.86985715e-01,  -1.37262591e+00,
 8.99349430e-01,   1.51647819e+00,  -2.83399657e-01,
-2.33745038e+00,   5.98819177e-01,  -1.00274840e+00,
 1.67953835e-02,  -1.69004184e-01,  -1.22409774e-01,
-8.81518518e-01,  -2.92639517e-02,  -1.29616708e+00,
-5.98594974e-01,  -7.59329140e-01,   7.59458188e-01,
-4.01086745e-01,  -8.70314527e-01,  -7.22597598e-01,
-1.00043696e+00,   4.51217930e-01,   2.33979130e-01,
 1.31096406e-02,  -1.00684773e+00,   5.00992021e-01,
-1.20464815e+00,   4.57077142e-01,  -1.48003329e+00,
-7.01909103e-01,   3.46227260e-01,   2.59535531e-01,
 8.56045074e-01,  -4.97825001e-01,   7.06507406e-01,
-2.34246449e+00,   6.32175595e-02,   5.14811919e-01,
 1.17783385e+00,   1.12729270e+00,   4.33554234e-01,
-8.69614810e-01,  -5.33971972e-01,   9.88704187e-01,
-3.05081559e-01,   3.13764782e-01,   3.06294694e-01,
-1.19265904e+00,   7.09196715e-01,   4.79351769e-01,
 5.95264270e-01,  -9.40091124e-01,  -1.31200123e+00,
-5.75416929e-01,   1.80849918e+00,  -2.25493484e+00,
 8.81114392e-01,  -1.55665780e+00,  -7.01596238e-01,
 5.43933800e-01,   5.95387756e-01,   9.32241570e-01])
```

In [10]:

```
# 用正态分布去拟合生成的数据，得到均值和标准差
Mean,std = stats.norm.fit(generated)
print('Mean = ',Mean,',std = ',std)
```

```
Mean =  -0.0530670389458 ,std =  0.977924988987
```

偏度（skewnes）描述的是概率分布的偏斜程度，我们需要做一个偏度检验。该检验有两个返回值，其中第二个返回值是p-value，即观察到的数据服从正态分布的概率，取值为0-1

In [11]:

```
stats.skewtest(generated)
```

Out[11]:

```
SkewtestResult(statistic=0.045696219506681769, pvalue=0.96355237706019969)
```

我们有96%把握认为其服从正态分布

峰度（kurtosis）描述的是概率分布的陡峭程度。该检验和偏度检验类似。

In [12]:

```
stats.kurtosistest(generated)
```

Out[12]:

```
KurtosistestResult(statistic=-1.2269819622972704, pvalue=0.21982937052710516)
```

正态性检验（normality test）可以检验数据服从正太分布的程度。

In [13]:

```
stats.normaltest(generated)
```

Out[13]:

```
NormaltestResult(statistic=1.507572880280063, pvalue=0.47058134698516707)
```

使用Scipy我们很方便的得到数据所在区域中某一百分比处的数值

In [14]:

```
# 例如得到95%处的数值
stats.scoreatpercentile(generated,95)
```

Out[14]:

```
1.5560778745880202
```

In [15]:

```
# 同样，可以反过来，得到数值所在的百分比
stats.percentileofscore(generated,1)
```

Out[15]:

```
86.222222222222229
```

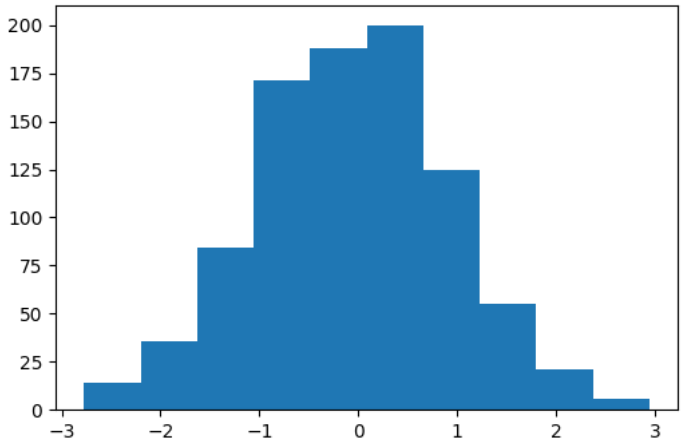
使用matplot绘制生成数据的分布直方图

In [16]:

```
import matplotlib.pyplot as plt
```

In [17]:

```
plt.hist(generated)
plt.show()
```



2.2 样本比对（比较股票对数收益率）¶

In [18]:

```
import numpy as np
price = get_price(['000001.XSHE','601398.XSHG'],start_date = '2016-01-01',end_date = '2017-01-01',fields='close')
price_001 = np.diff(np.log(np.array(price['000001.XSHE'])))
price_398 = np.diff(np.log(np.array(price['601398.XSHG'])))
```

均值检验可以检验两组不同的样本是否有相同的均值，返回值有两个，其中第二个为p-value,取值范围问为0~1

In [19]:

```
stats.ttest_ind(price_001,price_398)
```

Out[19]:

```
Ttest_indResult(statistic=-0.28934838774733274, pvalue=0.77243868777092461)
```

Kolmogorov-Smirnov检验可以判断两组样本同分布的可能性

In [20]:

```
stats.ks_2samp(price_001,price_398)
```

Out[20]:

```
Ks_2sampResult(statistic=0.094650205761316886, pvalue=0.21508957385193769)
```

在两支股票对数收益率的差值上运用Jarque-Bera正态性检验

In [21]:

```
stats.jarque_bera(price_001 -price_398)[-1]
```

Out[21]:

```
0.0
```

3 信号处理¶

3.1 检验股价的线性趋势¶

In [22]:

```
from datetime import date,datetime,time
from scipy import signal
import pandas as pd
from matplotlib.dates import DateFormatter
from matplotlib.dates import DayLocator
from matplotlib.dates import MonthLocator
price = get_price('000001.XSHE',start_date = '2016-01-01',end_date = '2017-01-01',fields='close')
```

In [23]:

```
y = signal.detrend(price)
trend = pd.Series(np.array(price) - y,index = price.index)
trend
```

Out[23]:

```
2016-01-04    8.358772
2016-01-05    8.363131
2016-01-06    8.367490
2016-01-07    8.371849
2016-01-08    8.376208
2016-01-11    8.380567
2016-01-12    8.384925
2016-01-13    8.389284
2016-01-14    8.393643
2016-01-15    8.398002
2016-01-18    8.402361
2016-01-19    8.406720
2016-01-20    8.411079
2016-01-21    8.415438
2016-01-22    8.419796
2016-01-25    8.424155
2016-01-26    8.428514
2016-01-27    8.432873
2016-01-28    8.437232
2016-01-29    8.441591
2016-02-01    8.445950
2016-02-02    8.450309
2016-02-03    8.454667
2016-02-04    8.459026
2016-02-05    8.463385
2016-02-15    8.467744
2016-02-16    8.472103
2016-02-17    8.476462
2016-02-18    8.480821
2016-02-19    8.485180
                 ...
2016-11-21    9.291571
2016-11-22    9.295930
2016-11-23    9.300289
2016-11-24    9.304648
2016-11-25    9.309007
2016-11-28    9.313366
2016-11-29    9.317725
2016-11-30    9.322083
2016-12-01    9.326442
2016-12-02    9.330801
2016-12-05    9.335160
2016-12-06    9.339519
2016-12-07    9.343878
2016-12-08    9.348237
2016-12-09    9.352596
2016-12-12    9.356954
2016-12-13    9.361313
2016-12-14    9.365672
2016-12-15    9.370031
2016-12-16    9.374390
2016-12-19    9.378749
2016-12-20    9.383108
2016-12-21    9.387467
2016-12-22    9.391825
2016-12-23    9.396184
2016-12-26    9.400543
2016-12-27    9.404902
2016-12-28    9.409261
2016-12-29    9.413620
2016-12-30    9.417979
dtype: float64
```

In [24]:

```
# 可以看下面去除趋势的作用
help(signal.detrend)
```

```
Help on function detrend in module scipy.signal.signaltools:

detrend(data, axis=-1, type='linear', bp=0)
    Remove linear trend along axis from data.

    Parameters
    ----------
    data : array_like
        The input data.
    axis : int, optional
        The axis along which to detrend the data. By default this is the
```

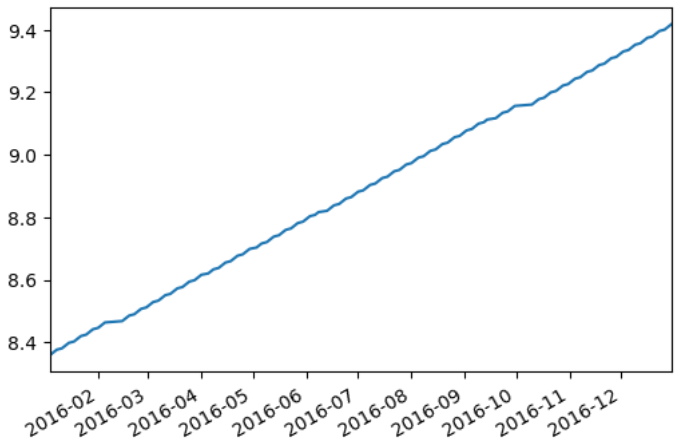
```
        last axis (-1).
    type : {'linear', 'constant'}, optional
        The type of detrending. If ``type == 'linear'`` (default),
        the result of a linear least-squares fit to `data` is subtracted
        from `data`.
        If ``type == 'constant'``, only the mean of `data` is subtracted.
    bp : array_like of ints, optional
        A sequence of break points. If given, an individual linear fit is
        performed for each part of `data` between two break points.
        Break points are specified as indices into `data`.

    Returns
    -------
    ret : ndarray
        The detrended input data.

    Examples
    --------
    >>> from scipy import signal
    >>> randgen = np.random.RandomState(9)
    >>> npoints = 1000
    >>> noise = randgen.randn(npoints)
    >>> x = 3 + 2*np.linspace(0, 1, npoints) + noise
    >>> (signal.detrend(x) - noise).max() < 0.01
    True
```

In [25]:

```
trend.plot()
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47f0480518>
```



3.2 傅里叶分析¶

对去除趋势后的信号进行滤波¶

In [26]:

```
from scipy import fftpack
```

In [27]:

```
# 运用傅里叶变换，得到信号的频谱
amps = np.abs(fftpack.fftshift(fftpack.rfft(y)))
```

In [28]:

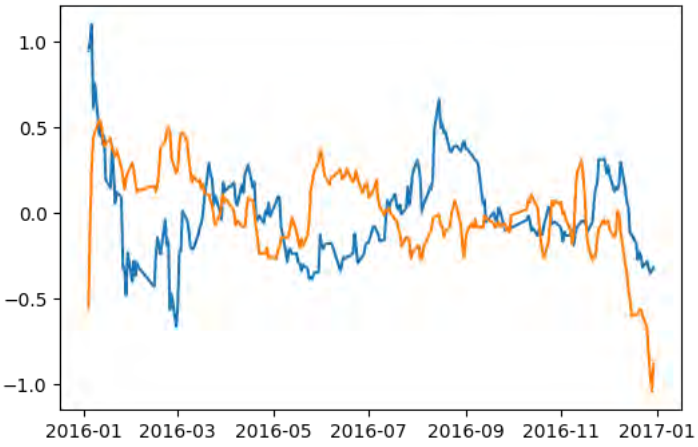
```
# 过滤噪音，如果某一频率分量的大小低于最强分贝的10%，过滤
amps[amps < 0.1*amps.max()] = 0
```

In [49]:

```
# 将过滤后的信号返回时域，并和去除趋势后的信号一起绘制出来
plt.plot(price.index,y,label = 'datrended')
plt.plot(price.index,-fftpack.irfft(fftpack.ifftshift(amps)),label = 'filtrend')
```

Out[49]:

```
[<matplotlib.lines.Line2D at 0x7fae0347c828>]
```



4 数学优化¶

优化算法尝试寻求某一问题的最优解，例如找到函数的最大值或最小值，函数可以是线性的也可以是非线性的。解可能也有一定的约束，例如大于1，。在scipy.optimize模块中提供了一些优化算法，包括最小二乘法。

4.1拟合正弦波¶

在上一章节，我们为去除趋势后的数据创建了一个简单的滤波器。我们可以回忆一下，一个正弦波优四个参数决定 Asin(ωx+φ)+k。
A——振幅，当物体作轨迹符合正弦曲线的直线往复运动时，其值为行程的1/2。
(ωx+φ)——相位，反映变量y所处的状态。
φ——初相，x=0时的相位；反映在坐标系上则为图像的左右移动。
k——偏距，反映在坐标系上则为图像的上移或下移。
ω——角速度， 控制正弦周期(单位角度内震动的次数)。

In [29]:

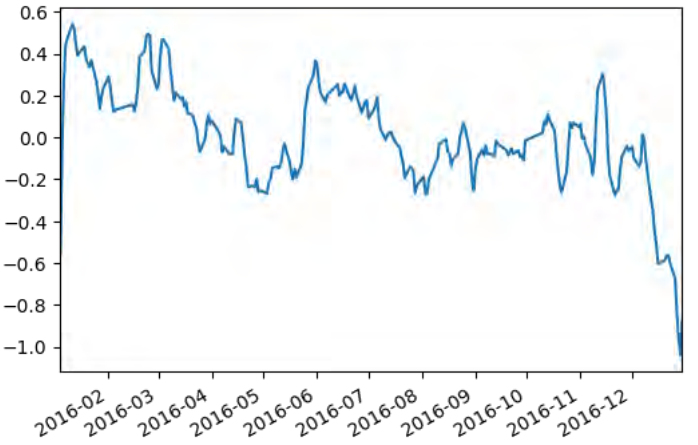
```
# 定义正弦波模型
def residuals(p,y,x):
    A,k,theta,b = p
    err = y-A*np.sin(2*np.pi*k*x+theta)+b
    return err
```

In [30]:

```
# 将滤波后的信号变换为时域
filtered = -fftpack.irfft(fftpack.ifftshift(amps))
pd.Series(filtered,index = price.index).plot()
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47eff59470>
```



In [31]:

```
N = len(filtered)
f = np.linspace(-N/2,N/2,N)
```

```
p0 = [filtered.max(),f[amps.argmax()]/(2*N),0,0]
p0
```

Out[31]:

```
[0.54062009323472304, 0.013374485596707843, 0, 0]
```

In [32]:

```
# 调用leastsp函数
from scipy import optimize
plsq = optimize.leastsq(residuals,p0,args = (filtered,f))
p = plsq[0]
```

In [33]:

```
p
```

Out[33]:

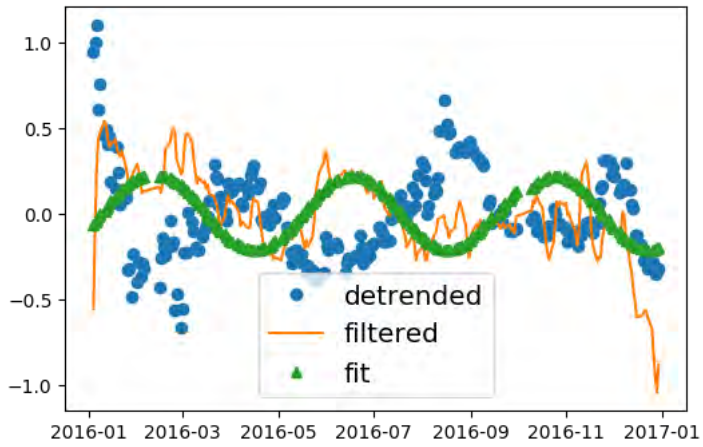
```
array([-0.21649664,  0.01171864, -0.76821357,  0.0065715 ])
```

In [34]:

```
fig = plt.figure()
plt.plot(price.index,y,'o',label = 'detrended')
plt.plot(price.index,filtered,label = 'filtered')
plt.plot(price.index,p[0]*np.sin(2*np.pi*f*p[1]+p[2])+p[3],'^',label = 'fit')
plt.legend(prop = {'size':'x-large'})
```

Out[34]:

```
<matplotlib.legend.Legend at 0x7f47efebd550>
```



4.2积分（以高斯积分为例）¶

高斯积分在概率论和连续傅里叶变换等的统一化等计算中有广泛的应用。在误差函数的定义中它也出现。虽然误差函数没有初等函数，但是高斯积分可以通过微积分学的手段解析求解。

In [35]:

```
from scipy import integrate
integrate.quad(lambda x: np.exp(-x**2),-np.inf,np.inf)
```

Out[35]:

```
(1.7724538509055159, 1.4202636780944923e-08)
```

In [36]:

```
np.sqrt(np.pi)
```

Out[36]:

```
1.7724538509055159
```

最常用的数值积分方法是quad,其它可用的积分方案有fixed_quad,quadrature,romberg.

插值¶

In [37]:

```
# 创建数据并添加噪音
x = np.linspace(-18,18,36)
noise = 0.1*np.random.random(len(x))
signal = np.sin(x) +noise
```

In [38]:

```
from scipy import interpolate
# 创造一个线性插值函数
interpreted = interpolate.interp1d(x,signal)
x2 = np.linspace(-18,18,180)
y = interpreted(x2)
```

In [39]:

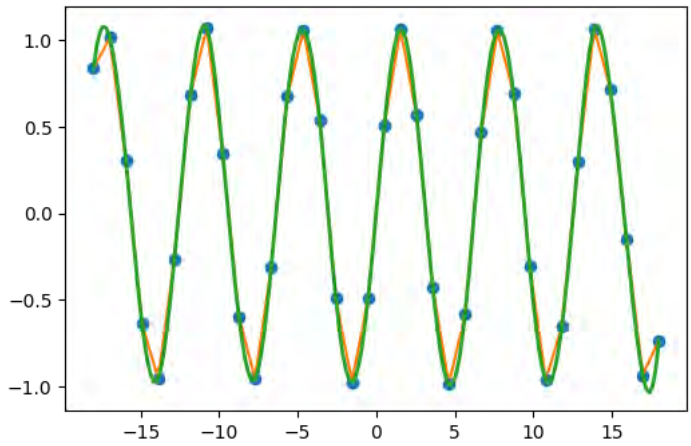
```
# 执行前一步相同的操作，这里使三次插值
cubic = interpolate.interp1d(x,signal,kind = 'cubic')
y2 = cubic(x2)
```

In [88]:

```
plt.plot(x,signal,'o',label = 'data')
plt.plot(x2,y,'-',label = 'lenear')
plt.plot(x2,y2,'-',lw = 2,label = 'cubic')
```

Out[88]:

```
[<matplotlib.lines.Line2D at 0x7fae031877b8>]
```



In [ ]:

```

```

Pandas¶

pandas 是基于 Numpy 构建的，让以 Numpy 为中心的应用变得更加简单。

pandas主要包括三类数据结构，分别是：

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。

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。

Pandas官网，更多功能请参考http://pandas-docs.github.io/pandas-docs-travis/index.html

In [1]:

```
#首先导入库
import pandas as pd
import numpy as np
```

Series¶

由一组数据（各种Numpy数据类型），以及一组与之相关的标签数据（即索引）组成。仅由一组数据即可产生最简单的Series，可以通过传递一个list对象来创建一个Seriess

In [2]:

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

Out[2]:

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

获取 Series 的索引:

In [3]:

```
s.index
```

Out[3]:

```
RangeIndex(start=0, stop=6, step=1)
```

DataFrame¶

DataFrame是一个表格型的数据结构，它含有一组有序的列，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（数值、字符、布尔值等）。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。DataFrame既有行索引也有列索引，可以被看做由Series组成的字典（共用同一个索引）。

创建一个DataFrame，包括一个numpy array，时间索引和列名字：¶

In [4]:

```
dates = pd.date_range('20130101', periods=6)
dates
```

Out[4]:

```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

In [5]:

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

Out[5]:

	A	B	C	D
2013-01-01	-0.023872	-0.662149	0.035905	-0.673801
2013-01-02	-1.294255	0.338153	0.073737	-0.158299
2013-01-03	0.298498	1.069045	1.249769	0.734961
2013-01-04	-0.883381	-0.115720	-0.878370	0.265243
2013-01-05	-0.090449	-0.084664	-1.489024	-0.838460
2013-01-06	1.255759	1.121589	1.469664	-1.234349

通过传递一个能够被转换成类似序列结构的字典对象来创建一个DataFrame：¶

In [6]:

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

```
                    })
df2
```

Out[6]:

	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

查看不同列的数据类型：¶

In [7]:

```
df2.dtypes
```

Out[7]:

```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

使用Tab自动补全功能会自动识别所有的属性以及自定义的列¶

查看数据¶

我们以平台获取的数据为例进行讲解：

In [55]:

```
# 获取平安银行近几个工作日的开盘价、最高价、最低价、收盘价。
df = get_price('000001.XSHE','2017-06-01','2017-06-14','1d',['open','high','low','close'])
df
```

Out[55]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03
2017-06-06	9.01	9.06	8.99	9.04
2017-06-07	9.02	9.15	9.01	9.13
2017-06-08	9.11	9.15	9.08	9.13
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11
2017-06-13	9.11	9.14	9.05	9.12
2017-06-14	9.12	9.13	9.04	9.08

查看前几条数据：

In [10]:

```
df.head()
```

Out[10]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17

	open	high	low	close
2017-06-05	9.13	9.17	8.99	9.03
2017-06-06	9.01	9.06	8.99	9.04
2017-06-07	9.02	9.15	9.01	9.13

查看后几条数据：

In [11]:

```
df.tail()
```

Out[11]:

	open	high	low	close
2017-06-08	9.11	9.15	9.08	9.13
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11
2017-06-13	9.11	9.14	9.05	9.12
2017-06-14	9.12	9.13	9.04	9.08

显示索引、列和底层的numpy数据：

In [12]:

```
df.index
```

Out[12]:

```
DatetimeIndex(['2017-06-01', '2017-06-02', '2017-06-05', '2017-06-06',
               '2017-06-07', '2017-06-08', '2017-06-09', '2017-06-12',
               '2017-06-13', '2017-06-14'],
              dtype='datetime64[ns]', freq=None)
```

查看 DataFrame 的列名

In [13]:

```
df.columns
```

Out[13]:

```
Index(['open', 'high', 'low', 'close'], dtype='object')
```

查看 DataFrame 的值

In [15]:

```
df.values
```

Out[15]:

```
array([[ 9.2 ,  9.23,  9.12,  9.19],
       [ 9.18,  9.29,  9.14,  9.17],
       [ 9.13,  9.17,  8.99,  9.03],
       [ 9.01,  9.06,  8.99,  9.04],
       [ 9.02,  9.15,  9.01,  9.13],
       [ 9.11,  9.15,  9.08,  9.13],
       [ 9.15,  9.22,  9.12,  9.15],
       [ 9.15,  9.19,  9.1 ,  9.11],
       [ 9.11,  9.14,  9.05,  9.12],
       [ 9.12,  9.13,  9.04,  9.08]])
```

describe()函数对于数据的快速统计汇总：

In [16]:

```
df.describe()
```

Out[16]:

	open	high	low	close
count	10.000000	10.00000	10.000000	10.000000
mean	9.118000	9.17300	9.064000	9.115000
std	0.061608	0.06343	0.056016	0.052122
min	9.010000	9.06000	8.990000	9.030000
25%	9.110000	9.14250	9.017500	9.087500
50%	9.125000	9.16000	9.065000	9.125000
75%	9.150000	9.21250	9.115000	9.145000
max	9.200000	9.29000	9.140000	9.190000

对数据的转置(tranverse):

In [17]:

```
df.T
```

Out[17]:

	2017-06-01 00:00:00	2017-06-02 00:00:00	2017-06-05 00:00:00	2017-06-06 00:00:00	2017-06-07 00:00:00	2017-06-08 00:00:00	2017-06-09 00:00:00	2017-06-12 00:00:00	2017-06-13 00:00:00	2017-06-14 00:00:00
open	9.20	9.18	9.13	9.01	9.02	9.11	9.15	9.15	9.11	9.12
high	9.23	9.29	9.17	9.06	9.15	9.15	9.22	9.19	9.14	9.13
low	9.12	9.14	8.99	8.99	9.01	9.08	9.12	9.10	9.05	9.04
close	9.19	9.17	9.03	9.04	9.13	9.13	9.15	9.11	9.12	9.08

按轴进行排序

In [18]:

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

Out[18]:

	open	low	high	close
2017-06-01	9.20	9.12	9.23	9.19
2017-06-02	9.18	9.14	9.29	9.17
2017-06-05	9.13	8.99	9.17	9.03
2017-06-06	9.01	8.99	9.06	9.04
2017-06-07	9.02	9.01	9.15	9.13
2017-06-08	9.11	9.08	9.15	9.13
2017-06-09	9.15	9.12	9.22	9.15
2017-06-12	9.15	9.10	9.19	9.11
2017-06-13	9.11	9.05	9.14	9.12
2017-06-14	9.12	9.04	9.13	9.08

6.按值进行排序

In [19]:

```
df.sort_values(by= 'open',ascending = False)
```

Out[19]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-09	9.15	9.22	9.12	9.15

	open	high	low	close
2017-06-12	9.15	9.19	9.10	9.11
2017-06-05	9.13	9.17	8.99	9.03
2017-06-14	9.12	9.13	9.04	9.08
2017-06-08	9.11	9.15	9.08	9.13
2017-06-13	9.11	9.14	9.05	9.12
2017-06-07	9.02	9.15	9.01	9.13
2017-06-06	9.01	9.06	8.99	9.04

选择数据¶

通过下标选取数据:¶

df['open']，df.open 以上两个语句是等效的，都是返回 df 名称为 open 列的数据，返回的为一个 Series。 df[0:3], df['2017-06-01':'2017-06-05'] 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0-2三条记录。 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,如：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。具体看下方示例： 以上两种方式返回的都是DataFrame。

选择一列数据：

In [20]:

```
df['open']
```

Out[20]:

```
2017-06-01    9.20
2017-06-02    9.18
2017-06-05    9.13
2017-06-06    9.01
2017-06-07    9.02
2017-06-08    9.11
2017-06-09    9.15
2017-06-12    9.15
2017-06-13    9.11
2017-06-14    9.12
Name: open, dtype: float64
```

选择多列数据：

In [21]:

```
df[['open','close']]
```

Out[21]:

	open	close
2017-06-01	9.20	9.19
2017-06-02	9.18	9.17
2017-06-05	9.13	9.03
2017-06-06	9.01	9.04
2017-06-07	9.02	9.13
2017-06-08	9.11	9.13
2017-06-09	9.15	9.15
2017-06-12	9.15	9.11
2017-06-13	9.11	9.12
2017-06-14	9.12	9.08

选择多行：

In [22]:

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

Out[22]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03

In [23]:

```
df['2017-06-01':'2017-06-05']
```

Out[23]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03

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[:,'open'] #选取 open 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。

PS：loc为location的缩写。

In [27]:

```
df.loc['2017-06-01','open']
```

Out[27]:

```
9.1999999999999993
```

In [30]:

```
df.loc['2017-06-01':'2017-06-06']
```

Out[30]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03
2017-06-06	9.01	9.06	8.99	9.04

In [31]:

```
df.loc[:, 'open']
```

Out[31]:

```
2017-06-01    9.20
2017-06-02    9.18
2017-06-05    9.13
2017-06-06    9.01
2017-06-07    9.02
2017-06-08    9.11
2017-06-09    9.15
2017-06-12    9.15
2017-06-13    9.11
2017-06-14    9.12
Name: open, dtype: float64
```

In [32]:

```
df.loc['2017-06-01':'2017-06-06','open']
```

Out[32]:

```
2017-06-01    9.20
2017-06-02    9.18
2017-06-05    9.13
2017-06-06    9.01
Name: open, dtype: float64
```

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] #选取所有记录的第二列的值，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：iloc 则为 integer & location 的缩写

In [33]:

```
df.iloc[1,1] # 选取第二行，第二列的值，返回的为单个值
```

Out[33]:

```
9.2899999999999991
```

In [56]:

```
df.iloc[[0,2],:] # 选取第一行及第三行的数据
```

Out[56]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-05	9.13	9.17	8.99	9.03

In [57]:

```
df.iloc[0:2,:] # 选取第一行到第三行（不包含）的数据
```

Out[57]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17

In [58]:

```
df.iloc[:,1] # 选取所有记录的第一列的值，返回的为一个Series
```

Out[58]:

```
2017-06-01    9.23
2017-06-02    9.29
2017-06-05    9.17
2017-06-06    9.06
2017-06-07    9.15
2017-06-08    9.15
2017-06-09    9.22
2017-06-12    9.19
2017-06-13    9.14
2017-06-14    9.13
Name: high, dtype: float64
```

In [60]:

```
df.iloc[1,:] # 选取第一行数据，返回的为一个Series
```

Out[60]:

```
open     9.18
high     9.29
low      9.14
close    9.17
Name: 2017-06-02 00:00:00, dtype: float64
```

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

In [62]:

```
df.ix[1,1]
```

Out[62]:

```
9.2899999999999991
```

In [63]:

```
df.ix['2017-06-01':'2017-06-05']
```

Out[63]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03

In [65]:

```
df.ix['2017-06-05','open']
```

Out[65]:

```
9.1300000000000008
```

In [66]:

```
df.ix[1,'open']
```

Out[66]:

```
9.1799999999999997
```

In [68]:

```
df.ix['2017-06-01',0]
```

Out[68]:

```
9.1999999999999993
```

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

In [69]:

```
# 筛选出 open 大于 8.8 的数据
df[df.open > 9.12]
```

Out[69]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19

	open	high	low	close
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11

In [71]:

```
# 筛选出 open 大于 9.12 的数据，并且 close 小于 9.17 的数据
df[(df.open >9.12) & (df.close < 9.17)]
```

Out[71]:

	open	high	low	close
2017-06-05	9.13	9.17	8.99	9.03
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11

使用 条件过来更改数据。

In [72]:

```
df[df>10]
```

Out[72]:

	open	high	low	close
2017-06-01	NaN	NaN	NaN	NaN
2017-06-02	NaN	NaN	NaN	NaN
2017-06-05	NaN	NaN	NaN	NaN
2017-06-06	NaN	NaN	NaN	NaN
2017-06-07	NaN	NaN	NaN	NaN
2017-06-08	NaN	NaN	NaN	NaN
2017-06-09	NaN	NaN	NaN	NaN
2017-06-12	NaN	NaN	NaN	NaN
2017-06-13	NaN	NaN	NaN	NaN
2017-06-14	NaN	NaN	NaN	NaN

观察可以发现，df 中小于 9 的数都变为 NaN。

下面我们就把大于 9 的数赋值为0.

In [73]:

```
df[df > 9.12] = 0
```

In [74]:

```
df
```

Out[74]:

	open	high	low	close
2017-06-01	0.00	0.00	9.12	0.00
2017-06-02	0.00	0.00	0.00	0.00
2017-06-05	0.00	0.00	8.99	9.03
2017-06-06	9.01	9.06	8.99	9.04
2017-06-07	9.02	0.00	9.01	0.00
2017-06-08	9.11	0.00	9.08	0.00

	open	high	low	close
2017-06-09	0.00	0.00	9.12	0.00
2017-06-12	0.00	0.00	9.10	9.11
2017-06-13	9.11	0.00	9.05	9.12
2017-06-14	9.12	0.00	9.04	9.08

发现大于 9.12 的数都被替换为了0.

使用isin()方法来过滤在指定列中的数据：

In [75]:

```
# 选取 high 列中数为 0 和 9 的数。
df[df['high'].isin([0.00,9.00])]
```

Out[75]:

	open	high	low	close
2017-06-01	0.00	0.0	9.12	0.00
2017-06-02	0.00	0.0	0.00	0.00
2017-06-05	0.00	0.0	8.99	9.03
2017-06-07	9.02	0.0	9.01	0.00
2017-06-08	9.11	0.0	9.08	0.00
2017-06-09	0.00	0.0	9.12	0.00
2017-06-12	0.00	0.0	9.10	9.11
2017-06-13	9.11	0.0	9.05	9.12
2017-06-14	9.12	0.0	9.04	9.08

Panel¶

平台get_price，如果是多支股票, 则返回pandas.Panel对象。

可通过 panel[列标,行标，股票代码] 获取数据.

In [77]:

```
panel = get_price(['000001.XSHE','000002.XSHE'],'2017-06-01','2017-06-14', '1d', fields=['open','high','low','close'])
panel
```

Out[77]:

```
<class 'rqcommons.pandas_patch.HybridDataPanel'>
Dimensions: 4 (items) x 10 (major_axis) x 2 (minor_axis)
Items axis: open to close
Major_axis axis: 2017-06-01 00:00:00 to 2017-06-14 00:00:00
Minor_axis axis: 000001.XSHE to 000002.XSHE
```

由打印的结果可以看出：

列标(Items axis: close to open)

行标(Major_axis axis: 2017-06-01 00:00:00 to 2017-06-14 00:00:00)

股票代码(Minor_axis axis: 000001.XSHE to 000002.XSHE)

In [78]:

```
# 取出 'open'数据
panel['open',:,:]
```

Out[78]:

	000001.XSHE	000002.XSHE
2017-06-01	9.20	21.10
2017-06-02	9.18	21.08
2017-06-05	9.13	20.75

	000001.XSHE	000002.XSHE
2017-06-06	9.01	21.05
2017-06-07	9.02	20.87
2017-06-08	9.11	20.87
2017-06-09	9.15	20.87
2017-06-12	9.15	21.48
2017-06-13	9.11	21.91
2017-06-14	9.12	21.90

In [79]:

```
# 取出 '2016-07-15'数据
panel[:,'2017-06-01',:]
```

Out[79]:

	open	high	low	close
000001.XSHE	9.2	9.23	9.12	9.19
000002.XSHE	21.1	21.37	20.93	21.09

In [80]:

```
# 取出 000001 的 DataFrame 数据
panel[:,:,'000001.XSHE']
```

Out[80]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03
2017-06-06	9.01	9.06	8.99	9.04
2017-06-07	9.02	9.15	9.01	9.13
2017-06-08	9.11	9.15	9.08	9.13
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11
2017-06-13	9.11	9.14	9.05	9.12
2017-06-14	9.12	9.13	9.04	9.08

Panel 操作与 DataFrame 基本相同，下节会为大家讲解 DataFrame 的数据处理与规整。

缺失数据处理¶

去掉包含缺失值的行：¶

In [85]:

```
# 获取平安银行近几个工作日的开盘价、最高价、最低价、收盘价。
df = get_price('000001.XSHE','2017-06-01','2017-06-14', '1d', fields=['open','high','low','close'])
df = df[df > 9.0]
df
```

Out[85]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	NaN	9.03
2017-06-06	9.01	9.06	NaN	9.04

	open	high	low	close
2017-06-07	9.02	9.15	9.01	9.13
2017-06-08	9.11	9.15	9.08	9.13
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11
2017-06-13	9.11	9.14	9.05	9.12
2017-06-14	9.12	9.13	9.04	9.08

In [86]:

```
df.dropna()
```

Out[86]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-07	9.02	9.15	9.01	9.13
2017-06-08	9.11	9.15	9.08	9.13
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11
2017-06-13	9.11	9.14	9.05	9.12
2017-06-14	9.12	9.13	9.04	9.08

对缺失值进行填充：¶

In [87]:

```
df.fillna(value=0)
```

Out[87]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	0.00	9.03
2017-06-06	9.01	9.06	0.00	9.04
2017-06-07	9.02	9.15	9.01	9.13
2017-06-08	9.11	9.15	9.08	9.13
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11
2017-06-13	9.11	9.14	9.05	9.12
2017-06-14	9.12	9.13	9.04	9.08

判断数据是否为nan，并进行布尔填充：¶

In [88]:

```
pd.isnull(df)
```

Out[88]:

	open	high	low	close
2017-06-01	False	False	False	False
2017-06-02	False	False	False	False
2017-06-05	False	False	True	False

	open	high	low	close
2017-06-06	False	False	True	False
2017-06-07	False	False	False	False
2017-06-08	False	False	False	False
2017-06-09	False	False	False	False
2017-06-12	False	False	False	False
2017-06-13	False	False	False	False
2017-06-14	False	False	False	False

函数的应用和映射¶

In [89]:

```
df.mean()#列计算平均值
```

Out[89]:

```
open     9.1180
high     9.1730
low      9.0825
close    9.1150
dtype: float64
```

In [90]:

```
df.mean(1)#行计算平均值
```

Out[90]:

```
2017-06-01    9.185000
2017-06-02    9.195000
2017-06-05    9.110000
2017-06-06    9.036667
2017-06-07    9.077500
2017-06-08    9.117500
2017-06-09    9.160000
2017-06-12    9.137500
2017-06-13    9.105000
2017-06-14    9.092500
dtype: float64
```

In [91]:

```
df.mean(axis = 1,skipna = False) # skipna参数默认是 True 表示排除缺失值
```

Out[91]:

```
2017-06-01    9.1850
2017-06-02    9.1950
2017-06-05       NaN
2017-06-06       NaN
2017-06-07    9.0775
2017-06-08    9.1175
2017-06-09    9.1600
2017-06-12    9.1375
2017-06-13    9.1050
2017-06-14    9.0925
dtype: float64
```

常用的方法如上所介绍们，还要其他许多，可自行学习，下面罗列了一些，可供参考：

- count 非na值的数量
- describe 针对Series或个DataFrame列计算汇总统计
- min、max 计算最小值和最大值
- argmin、argmax 计算能够获取到最大值和最小值得索引位置（整数）
- idxmin、idxmax 计算能够获取到最大值和最小值得索引值
- quantile 计算样本的分位数（0到1）

- sum 值的总和
- mean 值得平均数
- median 值得算术中位数（50%分位数）
- mad 根据平均值计算平均绝对离差
- var 样本值的方差
- std 样本值的标准差
- skew 样本值得偏度（三阶矩）
- kurt 样本值得峰度（四阶矩）
- cumsum 样本值得累计和
- cummin，cummax 样本值得累计最大值和累计最小值
- cumprod 样本值得累计积
- diff 计算一阶差分（对时间序列很有用）
- pct_change 计算百分数变化

数据规整¶

Pandas提供了大量的方法能够轻松的对Series，DataFrame和Panel对象进行各种符合各种逻辑关系的合并操作

- concat 可以沿一条轴将多个对象堆叠到一起。
- append 将一行连接到一个DataFrame上
- duplicated 移除重复数据

concat ¶

In [92]:

```
df1 = get_price('000001.XSHE','2017-06-01', '2017-06-14','1d',['open','high','low','close'])
df1
```

Out[92]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03
2017-06-06	9.01	9.06	8.99	9.04
2017-06-07	9.02	9.15	9.01	9.13
2017-06-08	9.11	9.15	9.08	9.13
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11
2017-06-13	9.11	9.14	9.05	9.12
2017-06-14	9.12	9.13	9.04	9.08

In [94]:

```
df2 = get_price('000001.XSHE','2017-05-15','2017-05-31','1d',['open','high','low','close'])
df2
```

Out[94]:

	open	high	low	close
2017-05-15	8.89	8.95	8.81	8.86
2017-05-16	8.84	8.85	8.73	8.84
2017-05-17	8.81	8.81	8.75	8.77
2017-05-18	8.72	8.77	8.71	8.73

	open	high	low	close
2017-05-19	8.74	8.76	8.68	8.69
2017-05-22	8.68	8.73	8.61	8.68
2017-05-23	8.67	8.84	8.64	8.79
2017-05-24	8.78	8.83	8.68	8.81
2017-05-25	8.79	9.14	8.78	9.10
2017-05-26	9.08	9.13	9.04	9.10
2017-05-31	9.10	9.23	9.06	9.20

In [95]:

```
pd.concat([df1,df2],axis=0)
```

Out[95]:

```
            open  high   low  close
2017-06-01  9.20  9.23  9.12   9.19
2017-06-02  9.18  9.29  9.14   9.17
2017-06-05  9.13  9.17  8.99   9.03
2017-06-06  9.01  9.06  8.99   9.04
2017-06-07  9.02  9.15  9.01   9.13
2017-06-08  9.11  9.15  9.08   9.13
2017-06-09  9.15  9.22  9.12   9.15
2017-06-12  9.15  9.19  9.10   9.11
2017-06-13  9.11  9.14  9.05   9.12
2017-06-14  9.12  9.13  9.04   9.08
2017-05-15  8.89  8.95  8.81   8.86
2017-05-16  8.84  8.85  8.73   8.84
2017-05-17  8.81  8.81  8.75   8.77
2017-05-18  8.72  8.77  8.71   8.73
2017-05-19  8.74  8.76  8.68   8.69
2017-05-22  8.68  8.73  8.61   8.68
2017-05-23  8.67  8.84  8.64   8.79
2017-05-24  8.78  8.83  8.68   8.81
2017-05-25  8.79  9.14  8.78   9.10
2017-05-26  9.08  9.13  9.04   9.10
2017-05-31  9.10  9.23  9.06   9.20
```

横向拼接，index对不上的会用 NaN 填充:

In [97]:

```
pd.concat([df1,df2],axis=1)
```

Out[97]:

```
            open  high   low  close  open  high   low  close
2017-05-15   NaN   NaN   NaN    NaN  8.89  8.95  8.81   8.86
2017-05-16   NaN   NaN   NaN    NaN  8.84  8.85  8.73   8.84
2017-05-17   NaN   NaN   NaN    NaN  8.81  8.81  8.75   8.77
2017-05-18   NaN   NaN   NaN    NaN  8.72  8.77  8.71   8.73
2017-05-19   NaN   NaN   NaN    NaN  8.74  8.76  8.68   8.69
2017-05-22   NaN   NaN   NaN    NaN  8.68  8.73  8.61   8.68
2017-05-23   NaN   NaN   NaN    NaN  8.67  8.84  8.64   8.79
2017-05-24   NaN   NaN   NaN    NaN  8.78  8.83  8.68   8.81
2017-05-25   NaN   NaN   NaN    NaN  8.79  9.14  8.78   9.10
2017-05-26   NaN   NaN   NaN    NaN  9.08  9.13  9.04   9.10
2017-05-31   NaN   NaN   NaN    NaN  9.10  9.23  9.06   9.20
2017-06-01  9.20  9.23  9.12   9.19   NaN   NaN   NaN    NaN
2017-06-02  9.18  9.29  9.14   9.17   NaN   NaN   NaN    NaN
2017-06-05  9.13  9.17  8.99   9.03   NaN   NaN   NaN    NaN
2017-06-06  9.01  9.06  8.99   9.04   NaN   NaN   NaN    NaN
2017-06-07  9.02  9.15  9.01   9.13   NaN   NaN   NaN    NaN
2017-06-08  9.11  9.15  9.08   9.13   NaN   NaN   NaN    NaN
2017-06-09  9.15  9.22  9.12   9.15   NaN   NaN   NaN    NaN
2017-06-12  9.15  9.19  9.10   9.11   NaN   NaN   NaN    NaN
2017-06-13  9.11  9.14  9.05   9.12   NaN   NaN   NaN    NaN
2017-06-14  9.12  9.13  9.04   9.08   NaN   NaN   NaN    NaN
```

append ¶

In [98]:

```
df1
```

Out[98]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03
2017-06-06	9.01	9.06	8.99	9.04
2017-06-07	9.02	9.15	9.01	9.13
2017-06-08	9.11	9.15	9.08	9.13
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11
2017-06-13	9.11	9.14	9.05	9.12
2017-06-14	9.12	9.13	9.04	9.08

In [99]:

```
s = df1.iloc[0]
s
```

Out[99]:

```
open     9.20
high     9.23
low      9.12
close    9.19
Name: 2017-06-01 00:00:00, dtype: float64
```

In [100]:

```
df1.append(s, ignore_index=False) # ignore_index=False 表示索引不变
```

Out[100]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03
2017-06-06	9.01	9.06	8.99	9.04
2017-06-07	9.02	9.15	9.01	9.13
2017-06-08	9.11	9.15	9.08	9.13
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11
2017-06-13	9.11	9.14	9.05	9.12
2017-06-14	9.12	9.13	9.04	9.08
2017-06-01	9.20	9.23	9.12	9.19

In [101]:

```
df1.append(s, ignore_index=True) # ignore_index=True 表示索引重置
```

Out[101]:

	open	high	low	close
0	9.20	9.23	9.12	9.19
1	9.18	9.29	9.14	9.17
2	9.13	9.17	8.99	9.03
3	9.01	9.06	8.99	9.04

	open	high	low	close
4	9.02	9.15	9.01	9.13
5	9.11	9.15	9.08	9.13
6	9.15	9.22	9.12	9.15
7	9.15	9.19	9.10	9.11
8	9.11	9.14	9.05	9.12
9	9.12	9.13	9.04	9.08
10	9.20	9.23	9.12	9.19

移除重复数据duplicated ¶

In [102]:

```
z = df1.append(s, ignore_index=False)
z
```

Out[102]:

	open	high	low	close
2017-06-01	9.20	9.23	9.12	9.19
2017-06-02	9.18	9.29	9.14	9.17
2017-06-05	9.13	9.17	8.99	9.03
2017-06-06	9.01	9.06	8.99	9.04
2017-06-07	9.02	9.15	9.01	9.13
2017-06-08	9.11	9.15	9.08	9.13
2017-06-09	9.15	9.22	9.12	9.15
2017-06-12	9.15	9.19	9.10	9.11
2017-06-13	9.11	9.14	9.05	9.12
2017-06-14	9.12	9.13	9.04	9.08
2017-06-01	9.20	9.23	9.12	9.19

查看重复数据：

In [104]:

```
z.duplicated()
```

Out[104]:

```
2017-06-01    False
2017-06-02    False
2017-06-05    False
2017-06-06    False
2017-06-07    False
2017-06-08    False
2017-06-09    False
2017-06-12    False
2017-06-13    False
2017-06-14    False
2017-06-01     True
dtype: bool
```

Matplotlib ¶

数据的处理、分析和可视化已经成为 Python 近年来最重要的应用之一。这种现象又进一步引出“大数据”分析等类似的话题，而大数据分析在人们所能预见的诸多领域内都有广泛应用，这其中就包含笔者个人感兴趣的机器学习。

Python 在处理数据、分析数据以及数据可视化方面拥有很多功能强大的工具，这也是 Python 在科学领域中能够迅速发展的一个主要原因。

在接下来的一系列文章中，我们将介绍 Python 科学计算中涉及的主要的库，并且学习如何使用它们处理数据以满足我们的需求。但是我们并非只是停留在快速写出模板代码来使用这些库的层面上，我们还会了解这些库背后的数学知识，以帮助我们更好地理解库的运行原理。

首先，我们将从一个功能非常强大的库 Matplotlib 开始介绍，在后面的文章中也会一直用到这个库。

什么是 Matplotlib? ¶

简单来说，Matplotlib 是 Python 的一个绘图库。它包含了大量的工具，你可以使用这些工具创建各种图形，包括简单的散点图，正弦曲线，甚至是三维图形。Python 科学计算社区经常使用它完成数据可视化的工作。

你可以在他们的网站上了解到更多 Matplotlib 背后的设计思想，但是我强烈建议你先浏览一下他们的图库，体会一下这个库的各种神奇功能。

画一个简单的图形¶

首先我们要画一条在 [0, 2pi] 上的正弦曲线。读者应该会注意到我们在这里使用了 Numpy 库，但是即便你没有使用过这个库也不用担心，在后面的文章中我们也会介绍到 Numpy 库。

In [1]:

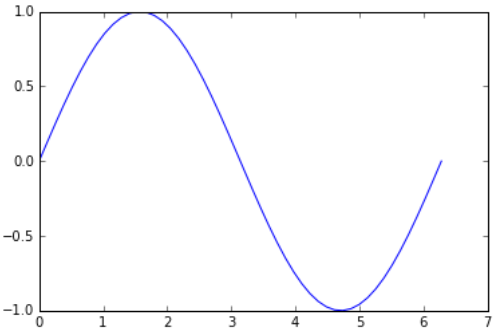
```
import matplotlib.pyplot as plt
import numpy as np
```

以上这些就是我们将要用到的导入模块。在我的上一篇文章(以及另一篇文章)中都提到过 from x import * 是一种糟糕的导入方式。我们不想在程序里重复书写 matplotlib.pyplot 和 numpy，这种书写方式过于冗长，因此我们采用了上面的折中写法。

简单的绘图¶

In [2]:

```
x = np.linspace(0, 2 * np.pi, 50)
plt.plot(x, np.sin(x)) # 如果没有第一个参数 x，图形的 x 坐标默认为数组的索引
plt.show() # 显示图形
```



上面的代码将画出一个简单的正弦曲线。np.linspace(0, 2 * np.pi, 50) 这段代码将会生成一个包含 50 个元素的数组，这 50 个元素均匀的分布在 [0, 2pi] 的区间上。

plot 命令以一种简洁优雅的方式创建了图形。提醒一下，如果没有第一个参数 x，图形的 x 轴坐标将不再是 0 到 2pi，而应该是数组的索引范围。

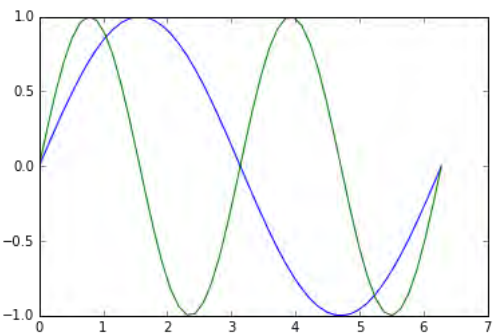
最后一行代码 `plt.show()`将图形显示出来，如果没有这行代码图像就不会显示。

在一张图上绘制两个数据集¶

大多数时候读者可能更想在一张图上绘制多个数据集。用 Matplotlib 也可以轻松实现这一点。

In [3]:

```
x = np.linspace(0, 2 * np.pi, 50)
plt.plot(x, np.sin(x),
         x, np.sin(2 * x))
plt.show()
```



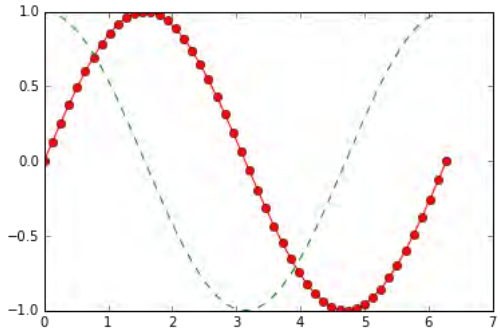
上面的代码同时绘制了表示函数 sin(x) 和 sin(2x) 的图形。这段代码和前面绘制一个数据集的代码几乎完全相同，只有一点例外，这段代码在调用 plt.plot() 的时候多传入了一个数据集，并用逗号与第一个数据集分隔开。

自定义图形的外观¶

当在同一个图形上展示多个数据集时，通过改变线条的外观来区分不同的数据集变得非常必要。

In [4]:

```
# 自定义曲线的外观
x = np.linspace(0, 2 * np.pi, 50)
plt.plot(x, np.sin(x), 'r-o',
        x, np.cos(x), 'g--')
plt.show()
```



上述代码展示了两种不同的曲线样式：'r-o' 和 'g--'。字母 'r' 和 'g' 代表线条的颜色，后面的符号代表线和点标记的类型。例如 '-o' 代表包含实心点标记的实线，'--' 代表虚线。其他的参数需要读者自己去尝试，这也是学习 Matplotlib 最好的方式。

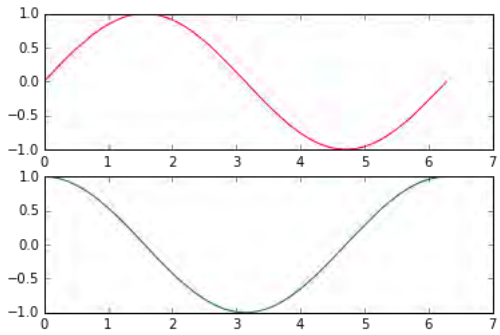
颜色： 蓝色 - 'b' 绿色 - 'g' 红色 - 'r' 青色 - 'c' 品红 - 'm' 黄色 - 'y' 黑色 - 'k'（'b'代表蓝色，所以这里用黑色的最后一个字母） 白色 - 'w' 线： 直线 - '-' 虚线 - '--' 点线 - ':' 点划线 - '-.' 常用点标记 点 - '.' 像素 - ',' 圆 - 'o' 方形 - 's' 三角形 - '^' 更多点标记样式点击:http://matplotlib.org/api/markers_api.html

使用子图¶

使用子图可以在一个窗口绘制多张图。

In [5]:

```
# 使用子图
x = np.linspace(0, 2 * np.pi, 50)
plt.subplot(2, 1, 1) # （行，列，活跃区）
plt.plot(x, np.sin(x), 'r')
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x), 'g')
plt.show()
```



使用子图只需要一个额外的步骤，就可以像前面的例子一样绘制数据集。即在调用 plot() 函数之前需要先调用 subplot() 函数。该函数的第一个参数代表子图的总行数，第二个参数代表子图的总列数，第三个参数代表活跃区域。

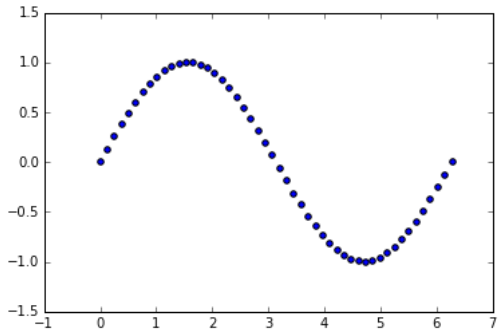
活跃区域代表当前子图所在绘图区域，绘图区域是按从左至右，从上至下的顺序编号。例如在 4×4 的方格上，活跃区域 6 在方格上的坐标为 (2, 2)。

简单的散点图¶

散点图是一堆离散点的集合。用 Matplotlib 画散点图也同样非常简单。

In [6]:

```
# 简单的散点图
x = np.linspace(0, 2 * np.pi, 50)
y = np.sin(x)
plt.scatter(x,y)
plt.show()
```



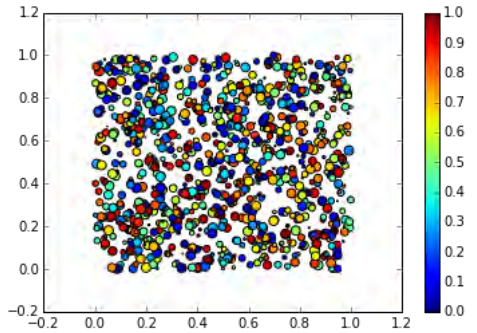
正如上面代码所示，你只需要调用 scatter() 函数并传入两个分别代表 x 坐标和 y 坐标的数组。注意，我们通过 plot 命令并将线的样式设置为 'bo' 也可以实现同样的效果。

彩色映射散点图¶

另一种你可能用到的图形是彩色映射散点图。这里我们会根据数据的大小给每个点赋予不同的颜色和大小，并在图中添加一个颜色栏。

In [7]:

```
# 彩色映射散点图
x = np.random.rand(1000)
y = np.random.rand(1000)
size = np.random.rand(1000) * 50
colour = np.random.rand(1000)
plt.scatter(x, y, size, colour)
plt.colorbar()
plt.show()
```



上面的代码大量的用到了 np.random.rand(1000)，原因是我们绘图的数据都是随机产生的。

同前面一样我们用到了 scatter() 函数，但是这次我们传入了另外的两个参数，分别为所绘点的大小和颜色。通过这种方式使得图上点的大小和颜色根据数据的大小产生变化。

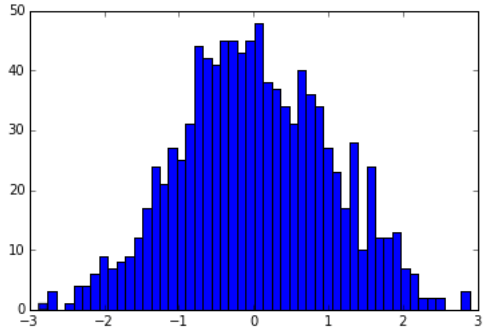
然后我们用 colorbar() 函数添加了一个颜色栏。

直方图¶

直方图是另一种常见的图形，也可以通过几行代码创建出来。

In [8]:

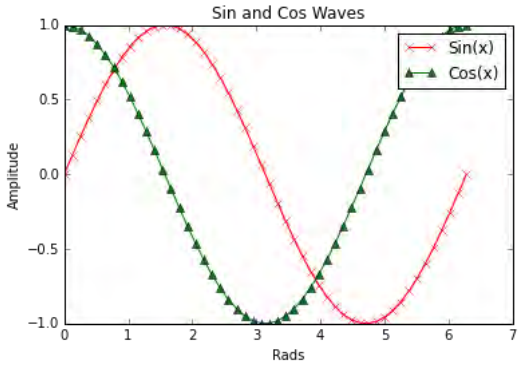
```
# 直方图
x = np.random.randn(1000)
plt.hist(x, 50)
plt.show()
```



直方图是 Matplotlib 中最简单的图形之一。你只需要给 hist() 函数传入一个包含数据的数组。第二个参数代表数据容器的个数。数据容器代表不同的值的间隔，并用来包含我们的数据。数据容器越多，图形上的数据条就越多。

In [9]:

```
# 添加标题，坐标轴标记和图例
x = np.linspace(0, 2 * np.pi, 50)
plt.plot(x, np.sin(x), 'r-x', label='Sin(x)')
plt.plot(x, np.cos(x), 'g-^', label='Cos(x)')
plt.legend() # 展示图例
plt.xlabel('Rads') # 给 x 轴添加标签
plt.ylabel('Amplitude') # 给 y 轴添加标签
plt.title('Sin and Cos Waves') # 添加图形标题
plt.show()
```



为了给图形添加图例，我们需要在 plot() 函数中添加命名参数 'label' 并赋予该参数相应的标签。然后调用 legend() 函数就会在我们的图形中添加图例。

接下来我们只需要调用函数 title()，xlabel() 和 ylabel() 就可以为图形添加标题和标签。

以上内容应该足够帮助读者开始使用 Matplotlib 和 Python 实现数据可视化，但是这些内容并不全面。我强烈建议读者亲自尝试使用这个工具，笔者也是通过这种方式掌握了这个工具。画一些图形，改变样式并使用子图功能，然后你就会很快掌握 Matplotlib 的使用方式。

这是一篇是关于如何使用 Matplotlib 和 Python 完成数据可视化的文章，也是 Python 科学计算系列文章中的第一篇。我希望读者能从中有所收获，并且对 Matplotlib 库更加熟悉。

In [ ]:

```
```

Python 策略范例

一步步开始写Alpha ¶

回测框架简介¶

我们提供了Beta版本的在线Python编辑器给您并且将会提供更多的功能改善编辑体验如自动代码、股票代码补全来让您更加享受编写您的算法，使得您可以很轻松地在Ricequant平台上编写您的第一个策略。

我们平台会对您的算法策略做编译和运行时的验证，在您运行完整的回测之前可以使用我们的在线编辑器的编译功能做快速验证以节省您的宝贵时间。

我们目前支持股票的日线和分钟线的回测，以及股票分钟线的实时模拟交易。

我们平台还兼容各类Python的数学、统计和机器学习等扩展模块，如Pandas, Numpy, Scipy, statsmodels, Scikit Learn等等，您可以实现技术分析、多因子、数据挖掘、统计套利、事件驱动等各种流派和风格的证券投资策略。

我们平台使用的是中国A股市场从2005年到最新交易日的每分钟以及每日行情数据，包括股票数据、ETF/LOF/分级基金数据、指数数据、期货数据、财务数据、国债收益率曲线。

我们将不同市场、不同金融产品的数据和交易规则尽量统一进我们的Ricequant API，因此您不必去担心不同市场、不同金融产品之间的不同数据格式和不同的交易规则，可以更专注于算法策略和交易盈亏本身。

我们还处于Beta测试阶段，正在努力开发更多的功能如您可以在社区上和其他人分享想法共同进步，如您有对我们的各种建议，不要犹豫，请联系我们。

核心函数介绍¶

init 函数（必须实现）¶

初始化方法 - 在回测和实时模拟交易只会在启动的时候触发一次。你的算法会使用这个方法来设置你需要的各种初始化配置。 context 对象将会在你的算法的所有其他的方法之间进行传递以方便你可以拿取到。

In [17]:

```
def init(context):
    # cash_limit的属性是根据用户需求自己定义的，你可以定义无限多种自己随后需要的属性，ricequant的系统默认只是会占用context.portfolio的关键字来
    context.cash_limit = 5000
```

before_trading 函数（选择实现）¶

可选择实现的函数。每天在策略开始交易前会被调用。不能在这个函数中发送订单。需要注意，该函数的触发时间取决于用户当前所订阅合约的交易时间。

举例来说，如果用户订阅的合约中存在有夜盘交易的期货合约，则该函数可能会在前一日的20:00触发，而不是早晨08:00.

In [18]:

```
def before_trading(context, bar_dict):
    # 拿取财务数据的逻辑，自己构建SQLAlchemy query
    fundamental_df = get_fundamentals(your_own_query)

    # 把查询到的财务数据保存到conext对象中
    context.fundamental_df = fundamental_df

    # 手动更新股票池
    update_universe(context.fundamental_df.columns.values)
```

handle_bar 函数（必须实现）¶

bar数据的更新会自动触发该方法的调用。策略具体逻辑可在该方法内实现，包括交易信号的产生、订单的创建等。在实时模拟交易中，该函数在交易时间内会每分钟被触发一次。

In [19]:

```
def handle_bar(context, bar_dict):
    # put all your algorithm main logic here.
    # ...
    order_shares('000001.XSHE', 500)
    # ...
```

after_trading 函数（选择实现）¶

可选择实现的函数。每天在收盘后被调用。不能在这个函数中发送订单。您可以在该函数中进行当日收盘后的一些计算。

在实时模拟交易中，该函数会在每天15:30触发。

证券的命名¶

股票、债券的命规则¶

股票代码是沪深两地证券交易所给上市股票分配的数字代码。这类代码涵盖所有在交易所挂牌交易的证券。熟悉这类代码有助于增加我们对交易品种的理解。

A股代码：沪市的为600×××，深市的为0×××；两市的后3位数字均是表示上市的先后顺序；

B股代码：沪市的为900×××，深市的为2×××；两市的后3位数字也是表示上市的先后顺序；

基金代码：沪市的为500×××，深市的为4×××；

配股代码：沪市的为700×××，深市的为8×××；后3位数为该股票代码的后3位数；

转配股代码：沪市的为710×××，深市的为3×××；后3位数为该股票代码的后3位数；

新股申购代码：沪市的为730×××，深市的即为该股票代码；

国债代码：沪市的为00××××，深市的为19××；

企业债券代码：沪市的为12××××，深市为10××；

回购债券：沪市的为2×××××，深市的为1×××。

期货的命名规则¶

期货合约的命名一般为：品种代码+交割年份（后两位）+交割月份（两位），如郑州硬麦：p0901

主力连续合约命名：品种代码+88，例如'IF88'

指数连续合约命名：品种代码+99

下单介绍¶

股票下单¶

平台提供了许多股票落单方式函数：

order_shares （指定股数交易）¶

落指定股数的买/卖单，最常见的落单方式之一。如有需要落单类型当做一个参量传入，如果忽略掉落单类型，那么默认是市价单（market order），也可以选择限价单（limit order）。

例如：购买Buy 2000 股的平安银行股票，并以市价单发送：

In [ ]:

```
order_shares('000001.XSHE', 2000)
```

order_lots （指定手数交易）¶

指定手数发送买/卖单。如有需要落单类型当做一个参量传入，如果忽略掉落单类型，那么默认是市价单（market order），也可以选择限价单（limit order）。

例如：买入20手的平安银行股票，并且发送市价单

In [ ]:

```
order_lots('000001.XSHE', 20)
```

order_value （指定价值交易）¶

使用想要花费的金钱买入/卖出股票，而不是买入/卖出想要的股数，正数代表买入，负数代表卖出。股票的股数总是会被调整成对应的100的倍数（在A中国A股市场1手是100股）。当您提交一个卖单时，该方法代表的意义是您希望通过卖出该股票套现的金额。如果金额超出了您所持有股票的价值，那么您将卖出所有股票。需要注意，如果资金不足，该API将不会创建发送订单。

例如：买入价值￥10000的平安银行股票，并以市价单发送。如果现在平安银行股票的价格是￥7.5，那么下面的代码会买入1300股的平安银行，因为少于100股的数目将会被自动删除掉

In [ ]:

```
order_value('000001.XSHE', 10000)
```

order_percent （一定比例下单）¶

发送一个等于目前投资组合价值（市场价值和目前现金的总和）一定百分比的买/卖单，正数代表买，负数代表卖。股票的股数总是会被调整成对应的一手的股票数的倍数（1手是100股）。百分比是一个小数，并且小于或等于1（<=100%），0.5表示的是50%.需要注意，如果资金不足，该API将不会创建发送订单。

例如：买入等于现有投资组合50%价值的平安银行股票。如果现在平安银行的股价是￥10/股并且现在的投资组合总价值是￥4000，用来买入的资金为￥2000，那么将会买入200股的平安银行股票。（不包含交易成本和滑点的损失）

In [ ]:

```
order_percent('000001.XSHG', 0.5)
```

order_target_value （目标价值下单）¶

买入/卖出并且自动调整该证券的仓位到一个目标价值。如果还没有任何该证券的仓位，那么会买入全部目标价值的证券。如果已经有了该证券的仓位，则会买入/卖出调整该证券的现在仓位和目标仓位的价值差值的数目的证券。需要注意，如果资金不足，该API将不会创建发送订单。

例如：如果现在的投资组合中持有价值￥3000的平安银行股票的仓位并且设置其目标价值为￥10000，以下代码范例会发送价值￥7000的平安银行的买单到市场。（向下调整到最接近每手股数即100的倍数的股数）：

In [ ]:

```
order_target_value('000001.XSHE', 10000)
```

order_target_percent （目标比例下单）¶

买入/卖出证券以自动调整该证券的仓位到占有一个指定的投资组合的目标百分比。

如果投资组合中没有任何该证券的仓位，那么会买入等于现在投资组合总价值的目标百分比的数目的证券。 如果投资组合中已经拥有该证券的仓位，那么会买入/卖出目标百分比和现有百分比的差额数目的证券，最终调整该证券的仓位占据投资组合的比例至目标百分比。 其实我们需要计算一个$position_to_adjust$ (即应该调整的仓位)

$$position\_to\_adjust = target\_position - current\_position$$

投资组合价值等于所有已有仓位的价值和剩余现金的总和。买/卖单会被下舍入一手股数（A股是100的倍数）的倍数。目标百分比应该是一个小数，并且最大值应该<=1，比如0.5表示50%。

如果$position_to_adjust$ 计算之后是正的，那么会买入该证券，否则会卖出该证券。 需要注意，如果资金不足，该API将不会创建发送订单。

例如：如果投资组合中已经有了平安银行股票的仓位，并且占据目前投资组合的10%的价值，那么以下代码会买入平安银行股票最终使其占据投资组合价值的15%

In [ ]:

```
order_target_percent('000001.XSHE', 0.15)
```

期货下单¶

平台提供了许多期货落单的方式函数：

buy_open （买开）¶

买入开仓

例如：以价格为3500的限价单开仓买入2张上期所AG1607合约

In [ ]:

```
buy_open('AG1607', amount=2, style=LimitOrder(3500))
```

sell_close （平多仓）¶

平多仓

$sell\_close(id\_or\_ins, amount, style=OrderType)$

sell_open （卖开）¶

卖开

$sell\_open(id\_or\_ins, amount, style=OrderType)$

buy_close （买开）¶

买开

$buy\_close(id\_or\_ins, amount, style=OrderType)$

cancel_order （撤单）¶

撤单

$cancel\_order(order)$

get_open_orders （拿到未成交订单信息）¶

获取一个由order_id到order对象映射的dict，凡在此dict中的order都未被完全成交或取消

$get\_open\_orders()$

数据格式¶

为了使用户能对更多的市场进行算法回测研究，我们自定义了一套统一的金融数据格式，有pandas Panel/DataFrame/Series以及ndarray

以获取股票历史行情函数get_price返回的数据格式为例介绍：

传入一个order_book_id ，多个fields， 函数会返回一个pandas DataFrame ¶

In [30]:

```
get_price('000001.XSHE', start_date='2013-01-04', end_date='2014-01-04', frequency='1d', fields=['close','open'], adjust_type='pre',
```

Out[30]:

	close	open
2013-01-04	5.5226	5.6366
2013-01-07	5.6297	5.5192
2013-01-08	5.5261	5.6297
2013-01-09	5.4777	5.5123
2013-01-10	5.4812	5.4777

传入一个order_book_id ，一个field， 函数会返回pandas Series ¶

In [31]:

```
get_price('000001.XSHE', start_date='2013-01-04', end_date='2014-01-04', frequency='1d', fields='close', adjust_type='pre', skip_susp
```

Out[31]:

```
2013-01-04    5.5226
2013-01-07    5.6297
2013-01-08    5.5261
2013-01-09    5.4777
2013-01-10    5.4812
Name: close, dtype: float64
```

传入多个order_book_id ，一个field，函数会返回一个pandas DataFrame ¶

In [32]:

```
get_price(['000001.XSHE','000002.XSHE'], start_date='2013-01-04', end_date='2014-01-04', frequency='1d', fields='close', adjust_type=
```

Out[32]:

	000001.XSHE	000002.XSHE
2013-01-04	5.5226	8.7657
2013-01-07	5.6297	8.7657
2013-01-08	5.5261	8.7657
2013-01-09	5.4777	8.7657
2013-01-10	5.4812	8.7657

传入多个order_book_id ，函数会返回一个pandas Panel ¶

In [38]:

```
get_price(['000001.XSHE','000002.XSHE'], start_date='2013-01-04', end_date='2014-01-04', frequency='1d', adjust_type='pre', skip_susp
```

Out[38]:

```
<class 'rqcommons.pandas_patch.HybridDataPanel'>
Dimensions: 8 (items) x 240 (major_axis) x 2 (minor_axis)
Items axis: open to limit_down
Major_axis axis: 2013-01-04 00:00:00 to 2014-01-03 00:00:00
Minor_axis axis: 000001.XSHE to 000002.XSHE
```

history_bars 函数¶

In [ ]:

```
history_bars(order_book_id, bar_count, frequency, fields)
```

获取指定合约的历史行情，同时支持日以及分钟历史数据。不能在init中调用。 注意，该API会自动跳过停牌数据。

可支持：日回测获取日历史数据、分钟回测获取日历史数据、分钟回测获取分钟历史数据

传入参数：¶

order_book_id ¶

$str\ 合约代码，必填项$

bar_count ¶

$int\ 获取的历史数据数量，必填项$

frequency ¶

$str\ 获取数据什么样的频率进行。'1d'或'1m'分别表示每日和每分钟，必填项$

fields ¶

$str\ 返回数据字段。必填项。见下方列表$

返回¶

ndarray ，方便直接与talib等计算库对接，效率较history返回的DataFrame更高

例如：获取最近5天的日线收盘价序列（策略当前日期为20160706）

In [ ]:

```
history_bars('000002.XSHE', 5, '1d', 'close')
```

上文已经提及了数据格式pandas Panel/DataFrame/Series ，那么在获得了这些数据格式之后怎么进行下一步的操作呢，下面将要介绍到pandas：¶

Pandas 模块¶

官网：http://pandas.pydata.org/pandas-docs/stable/index.html ¶

Pandas是一个开源的，为Python编程语言提供高性能，易于使用的数据结构和数据分析工具。它提供快速，灵活和富有解释性的数据结构以方便直观的处理"相关"或"被标记的"数据。

Pandas 适合处理多种类型的数据：¶

- 具有不同数据类型列的表格数据，如SQL表或Excel电子表格
- 有序或无序（不固定频率）的时间序列数据。
- 带有行和列标签的任意矩阵数据
- 任何其他形式的观测/统计数据集。

Pandas主要包含三种数据结构，分别是Series(一维)，DataFrame(二维)，Panel(三维)。其中Series和DataFrame可以用于处理绝大多数金融，统计，社会科学和许多工程领域的典型问题。对于R用户而言，DataFrame在支持所有R的data.frame的功能的基础上还能有更丰富的应用。Pandas库建立在NumPy库之上，旨在与科学计算环境和许多其他第三方库完美集成。

Pandas 的优势：¶

- 可以轻易的处理浮点及非浮点数据类型的缺失值(NaN)
- 大小可变：DataFrame和Panel都可以删除或插入列
- 数据自动对齐
- 灵活强大的分组功能，可对数据集进行拆分组合操作
- 方便的将其他Python和NumPy数据结构中不同类索引的数据转换为DataFrame对象
- 基于智能标签的切片，花式索引，轻易从大数据集中取出子集
- 直观的合并，连接数据集
- 轻易的重新定义数据集形状和转置
- 轴(axes)的分层标签（使每个元组有多个标签成为可能）

Pandas库是统计科学家在分析数据时的理想工具，非常适合应用于数据清洗，分析/建模，然后将分析结果组织成适合于绘图或表格显示的形式的全部过程。statsmodel库依赖Pandas库，使其成为Python统计计算系统的重要组成部分。

Pandas库已经广泛应用于金融数据。

*以上内容总结翻译自官网。

如果你已经安装了anaconda,Pandas库涵盖在anaconda提供的基本库内，因此我们无须安装即可使用。

下面是关于Pandas 基本操作的简单展示：¶

In [42]:

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

Series¶

Series是Pandas最重要的数据结构之一，是一种类似于一维数组的对象，它由一组数据（各种Numpy数据类型）以及一组与之相关的数据标签（即索引）组成。仅由一组数据即可产生简单的Series。 下面是最简单的一种创建Series对象的方式：

In [ ]:

```
s = pd.Series(data,index=index)#data的数据类型可以是任何NumPy数据类型，而index只需与其等长。
```

In [45]:

```
# 下面我们随机生成一组ndarray格式数据，并创建Series
s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
# 若不输入索引则默认索引为从零开始的整数列
j = pd.Series(np.random.randn(5))

# 打印s，j
print('show s :','\n',s)
print('\n','show j :','\n',j)

# 打印s的索引
print('\n','s.index :',s.index)

# Series中索引可以看作dict中的key而元组中的值可以看作dict中的value
# 像对dict进行操作一样，我们取出key为b的元素s['b']，并打印出来
print('\n','s[\'b\'] :',s['b'])
```

```
# 打印dict的keys，可以发现索引就是key
print('\n','s.keys() :',s.keys())

# 通过to_dict函数将Series转为dict
mapping = s.to_dict()
print('\n','mapping :','\n',mapping)

# 正如Series可以通过dict操作进行管理，通过词典方式也可以构建Series
print('\n','Series(mapping) :','\n',Series(mapping))

#一条为了美观的分割线
print('--------------------------------------------------------------------------------')
```

```
show s :
 a   -1.263266
b    1.193284
c   -1.145649
d   -0.116367
e   -0.354037
dtype: float64

 show j :
 0   -0.767192
1    0.791023
2    0.357123
3   -0.091236
4    1.230029
dtype: float64

 s.index : Index(['a', 'b', 'c', 'd', 'e'], dtype='object')

 s['b'] : 1.19328368136

 s.keys() : Index(['a', 'b', 'c', 'd', 'e'], dtype='object')

 mapping :
 {'d': -0.11636668522224589, 'e': -0.3540371134798011, 'a': -1.2632657357021486, 'b': 1.1932836813579908, 'c': -1.1456485965931713}

 Series(mapping) :
 a   -1.263266
b    1.193284
c   -1.145649
d   -0.116367
e   -0.354037
dtype: float64
--------------------------------------------------------------------------------
```

从ricequant数据库中提取出的数据默认保存为pandas的三种数据结构，下面我们调用get_price函数取出收盘价数据。

In [46]:

```
# 默认取出数据的长度为一年
# 取出收盘价数据的方式比较灵活，但在正常情况下，ts_2的运行速度会更快一些
ts_1 = get_price('600208.XSHG')['ClosingPx'][-5:]
ts_2 = get_price('600208.XSHG',fields='close')[-5:]

# 打印取出的近五天收盘价
print('ts_1 :','\n',ts_1,'\n\n','ts_2 :','\n',ts_2)
# 打印二者数据类型
print('\n','type(ts_1) :',type(ts_1),'\n','type(ts_2) :',type(ts_2))


print('--------------------------------------------------------------------------------')
```

```
ts_1 :
 2013-12-27    3.0431
2013-12-30    3.0528
2013-12-31    3.0816
2014-01-02    3.0720
2014-01-03    3.0142
Name: close, dtype: float64

 ts_2 :
 2013-12-27    3.0431
2013-12-30    3.0528
2013-12-31    3.0816
2014-01-02    3.0720
2014-01-03    3.0142
Name: close, dtype: float64
```

```
 type(ts_1) : <class 'pandas.core.series.Series'>
 type(ts_2) : <class 'pandas.core.series.Series'>
--------------------------------------------------------------------------------
```

get_price函数将一维数据保存在pandas.Series中，index为日期。

了解更多关于get_price函数的信息：https://www.ricequant.com/api/python/chn#data-methods-get_price

pandas中提供大量函数能够方便的帮助我们管理数据，下面我们以ts_1中取出的数据为例为各位展示其基本操作。

In [47]:

```
# 取前3项，并打印出来
print('ts_1的前三项：\n',ts_1[:3])

# 打印第三个索引
print('\n第三个索引名：',ts_1.index[2])

# 按索引2013-12-30输出当日收盘价，并打印
print('\n2013-12-30的收盘价：',ts_1['2013-12-30'])

# 按行输出第三行的收盘价，并打印
print('\nts_1[2] :',ts_1[2])

print('--------------------------------------------------------------------------------')
```

```
ts_1的前三项：
 2013-12-27    3.0431
2013-12-30    3.0528
2013-12-31    3.0816
Name: close, dtype: float64

第三个索引名： 2013-12-31 00:00:00

2013-12-30的收盘价： 3.0528

ts_1[2] : 3.0816
--------------------------------------------------------------------------------
```

Pandas可以实现R中非常令人喜爱的which函数功能（即Matlab中的find函数），在以下标为索引查找数据时比which函数更简便更直观。

In [48]:

```
# 打印ts_1前三项中最大值的索引
print('\nts_1.index[ts_1[:3] == ts_1[:3].max()] :\n',ts_1[:3].index[ts_1 == ts_1[:3].max()])

# 在筛选符合条件的项时操作比R和Matlab更简便
# 打印ts_1前三项中大于3.05的值
print('\nts_1[:3][ts_1>3.05]:\n',ts_1[:3][ts_1>3.05])

# 打印ts_1中等于3.0431的值
print('\nts_1[ts_1 == 3.0431]:\n',ts_1[ts_1 == 3.0431])

print('--------------------------------------------------------------------------------')
```

```
ts_1.index[ts_1[:3] == ts_1[:3].max()] :
 DatetimeIndex(['2013-12-31'], dtype='datetime64[ns]', freq=None)

ts_1[:3][ts_1>3.05]:
 2013-12-30    3.0528
2013-12-31    3.0816
Name: close, dtype: float64

ts_1[ts_1 == 3.0431]:
 2013-12-27    3.0431
Name: close, dtype: float64
--------------------------------------------------------------------------------
```

Pandas还能轻松的调用numpy库中的基本函数，也能做四则运算：

In [49]:

```
# 一阶差分
print('np.diff(ts_1):',np.diff(ts_1))

# 以e为底的指数
print('\nnp.exp(ts_1[:3]) :\n',np.exp(ts_1[:3]))

# 四则运算
print('\n加法：\nts_1[:3]+ts_1[:3]=\n',ts_1[:3]+ts_1[:3])
```

```
print('\n减法：\nts_1[:3]-ts_1[:3]=\n',ts_1[:3]-ts_1[:3])
print('\n乘法：\nts_1[:3] * 2 =\n',ts_1[:3] * 2)
print('\n除法：\nts_1[:3] / 2 =\n',ts_1[:3] / 2)

print('--------------------------------------------------------------------------------------------')
```

```
np.diff(ts_1): [ 0.0097  0.0288 -0.0096 -0.0578]

np.exp(ts_1[:3]) :
 2013-12-27    20.970150
2013-12-30    21.174550
2013-12-31    21.793244
Name: close, dtype: float64

加法：
ts_1[:3]+ts_1[:3]=
 2013-12-27    6.0862
2013-12-30    6.1056
2013-12-31    6.1632
Name: close, dtype: float64

减法：
ts_1[:3]-ts_1[:3]=
 2013-12-27    0.0
2013-12-30    0.0
2013-12-31    0.0
Name: close, dtype: float64

乘法：
ts_1[:3] * 2 =
 2013-12-27    6.0862
2013-12-30    6.1056
2013-12-31    6.1632
Name: close, dtype: float64

除法：
ts_1[:3] / 2 =
 2013-12-27    1.52155
2013-12-30    1.52640
2013-12-31    1.54080
Name: close, dtype: float64
--------------------------------------------------------------------------------------------
```

DataFrame¶

DataFrame是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等）。你可以把它看作是一个Excel表或者SQL表，或者value是一个Series的dict。

get_price函数同时支持取出一组股票的收盘价，结果将保存在DataFrame中。下面我们依然以get_price函数为例，简单介绍DataFrame的基本操作：

In [61]:

```
# 打印一组股票的收盘价，并打印其数据类型
df = get_price(['000024.XSHE', '000001.XSHE', '000002.XSHE'],start_date='20161010',end_date='20161020',fields='close')
print('df[:3] =\n',df[:3],'\n\ntype(df) :',type(df)) # 为了版面美观我们只打印前三项

# 000024.XSHE在这段时间没有收盘价，也许是因为已经退市或者未上市，我们删除这一列
# 在这里有三种不同的删除列的方法可供选择：
df_new = df.dropna(axis=1)#axis=0是按行删失，axis=1是按列删失
print('\ndf_new[:3] =\n',df_new[:3])

# 也可以队列中的pop函数
pop = df.pop('000024.XSHE')
print('\ndf[:3] :\n',df[:3])

df['000024.XSHE'] = pop


print('--------------------------------------------------------------------------------------------')
```

```
df[:3] =
             000024.XSHE  000001.XSHE  000002.XSHE
2016-10-10          NaN         9.12        25.97
2016-10-11          NaN         9.15        26.23
2016-10-12          NaN         9.13        26.60

type(df) : <class 'pandas.core.frame.DataFrame'>

df_new[:3] =
             000001.XSHE  000002.XSHE
2016-10-10         9.12        25.97
```

```
2016-10-11         9.15        26.23
2016-10-12         9.13        26.60

df[:3] :
             000001.XSHE  000002.XSHE
2016-10-10         9.12        25.97
2016-10-11         9.15        26.23
2016-10-12         9.13        26.60
--------------------------------------------------------------------------------
```

In [51]:

```
# 添加列
df['Ratio']=df['000001.XSHE']/df['000002.XSHE']
print('df[:3] :\n',df[:3])

# 打印行索引
print('\ndf[:3].index :\n',df[:3].index)
# 打印列索引
print('\ndf[:3].columns :\n',df[:3].columns)

# 转置df[:3]并打印
print('\ndf[:3].T:\n',df[:3].T)

print('--------------------------------------------------------------------------------')
```

```
df[:3] :
             000001.XSHE  000002.XSHE     Ratio
2016-10-10         9.12        25.97  0.351174
2016-10-11         9.15        26.23  0.348837
2016-10-12         9.13        26.60  0.343233

df[:3].index :
 DatetimeIndex(['2016-10-10', '2016-10-11', '2016-10-12'], dtype='datetime64[ns]', freq=None)

df[:3].columns :
 Index(['000001.XSHE', '000002.XSHE', 'Ratio'], dtype='object')

df[:3].T:
              2016-10-10  2016-10-11  2016-10-12
000001.XSHE    9.120000    9.150000    9.130000
000002.XSHE   25.970000   26.230000   26.600000
Ratio          0.351174    0.348837    0.343233
--------------------------------------------------------------------------------
```

在DataFrame中选择行列的方式与Series有些不同，在Series中data['index']可以根据行索引取出元素，而在DataFrame中输入索引默认的是查找列索引data['column']，这显然不太方便。那么当我们需要按行查找数据的时候怎么办呢？

Pandas库中有两个函数可以实现按行下标搜索和按行索引搜索的功能：（当然如果不嫌麻烦的话也可以转置后按列取的）

In [52]:

```
# 按行索引搜索
print('\ndf.loc[\'2016-10-11\'] :\n',df.loc['2016-10-11'])
# 按行下标搜索
print('\ndf.iloc[2] :\n',df.iloc[2])

print('--------------------------------------------------------------------------------')
```

```
df.loc['2016-10-11'] :
 000001.XSHE     9.150000
000002.XSHE    26.230000
Ratio           0.348837
Name: 2016-10-11 00:00:00, dtype: float64

df.iloc[2] :
 000001.XSHE     9.130000
000002.XSHE    26.600000
Ratio           0.343233
Name: 2016-10-12 00:00:00, dtype: float64
--------------------------------------------------------------------------------
```

还有一个解决方案：df.ix 可以混合使用索引和下标进行访问，但是行列内部需保持一致

In [54]:

```
# 按行下标
print('df.ix[1]:\n',df.ix[1])
# 按行索引
print('\ndf.ix[\'2016-10-11\'] :\n',df.ix['2016-10-11'])
```

```
print('--------------------------------------------------------------------------------------')
```

```
df.ix[1]:
 000001.XSHE     9.150000
000002.XSHE    26.230000
Ratio           0.348837
Name: 2016-10-11 00:00:00, dtype: float64

df.ix['2016-10-11'] :
 000001.XSHE     9.150000
000002.XSHE    26.230000
Ratio           0.348837
Name: 2016-10-11 00:00:00, dtype: float64
--------------------------------------------------------------------------------
```

如果只需要访问某位置的特定元素的话，df.iat 是速度最快的方式:

In [56]:

```
# 按下标
print('df.iat[1,0] =',df.iat[1,0])

print('--------------------------------------------------------------------------------------')
```

```
df.iat[1,0] = 9.15
--------------------------------------------------------------------------------
```

在DataFrame中，因为每一列都保存为Series结构，因此继承了Series的各类优点，可以进行数据筛选和各类按列运算，不仅如此，在DataFrame数据结构下，对其中每个元素的各种运算也非常便捷:

In [57]:

```
# 数乘
print('df[:3] * 5 +2 =\n',df[:3] * 5 +2)

# 平方
print('\ndf[:3] ** 2 =\n',df[:3] ** 2)

# 取倒数
print('\n1/df[:3] =\n',1/df[:3])

print('--------------------------------------------------------------------------------------')
```

```
df[:3] * 5 +2 =
             000001.XSHE  000002.XSHE     Ratio
2016-10-10        47.60       131.85  3.755872
2016-10-11        47.75       133.15  3.744186
2016-10-12        47.65       135.00  3.716165

df[:3] ** 2 =
             000001.XSHE  000002.XSHE     Ratio
2016-10-10      83.1744     674.4409  0.123323
2016-10-11      83.7225     688.0129  0.121687
2016-10-12      83.3569     707.5600  0.117809

1/df[:3] =
             000001.XSHE  000002.XSHE     Ratio
2016-10-10     0.109649     0.038506  2.847588
2016-10-11     0.109290     0.038124  2.866667
2016-10-12     0.109529     0.037594  2.913472
--------------------------------------------------------------------------------
```

In [58]:

```
# 按照步长为2取出数据
df2 = df.ix[::2]
print('df2:\n',df2)
print('\ndf + df2 =\n',df + df2)

print('--------------------------------------------------------------------------------------')
```

```
df2:
             000001.XSHE  000002.XSHE     Ratio
2016-10-10         9.12        25.97  0.351174
2016-10-12         9.13        26.60  0.343233
2016-10-14         9.09        26.84  0.338674
2016-10-18         9.09        26.00  0.349615
```

```
2016-10-20         9.07        25.94  0.349653

df + df2 =
             000001.XSHE  000002.XSHE     Ratio
2016-10-10         18.24        51.94  0.702349
2016-10-11           NaN          NaN       NaN
2016-10-12         18.26        53.20  0.686466
2016-10-13           NaN          NaN       NaN
2016-10-14         18.18        53.68  0.677347
2016-10-17           NaN          NaN       NaN
2016-10-18         18.18        52.00  0.699231
2016-10-19           NaN          NaN       NaN
2016-10-20         18.14        51.88  0.699306
--------------------------------------------------------------------------------
```

可以看出，对于那些df2中不存在索引的项，dataframe默认为NaN，四则运算的结果依然为NaN。

DataFrame中有两个常用的函数，分别是join和concat。均负责连接两数据集。

In [60]:

```
# join函数内连接取交集：
s1 = df[:2]
s2 = df[4:6]
innerjoin = s1.align(s2,join='inner')
print('innerjoin =\n',innerjoin)

# join函数外连接取并集
outerjoin = s1.align(s2,join='outer')
print('\nouterjoin =\n',outerjoin)

# concat连接两dataframe
df_new = get_price(['000006.XSHE','000056.XSHE'],start_date='20161010',end_date='20161020',fields='close')
df_concat = pd.concat([df,df_new],axis=1)
print('\ndf_concat =\n',df_concat[:3])

print('--------------------------------------------------------------------------------')
```

```
innerjoin =
 (Empty DataFrame
Columns: [000001.XSHE, 000002.XSHE, Ratio]
Index: [], Empty DataFrame
Columns: [000001.XSHE, 000002.XSHE, Ratio]
Index: [])

outerjoin =
 (            000001.XSHE  000002.XSHE     Ratio
2016-10-10         9.12        25.97  0.351174
2016-10-11         9.15        26.23  0.348837
2016-10-14           NaN          NaN       NaN
2016-10-17           NaN          NaN       NaN,             000001.XSHE  000002.XSHE     Ratio
2016-10-10           NaN          NaN       NaN
2016-10-11           NaN          NaN       NaN
2016-10-14         9.09        26.84  0.338674
2016-10-17         9.05        26.00  0.348077)

df_concat =
             000001.XSHE  000002.XSHE     Ratio  000006.XSHE  000056.XSHE
2016-10-10         9.12        25.97  0.351174         9.92        12.74
2016-10-11         9.15        26.23  0.348837        10.43        12.91
2016-10-12         9.13        26.60  0.343233        10.70        12.80
--------------------------------------------------------------------------------
```

官网上还有更多范例和讲解等待各位临幸：

- http://pandas.pydata.org/pandas-docs/stable/cookbook.html#cookbook
- http://pandas.pydata.org/pandas-docs/stable/dsintro.html

一步一步写一个简单的技术指标策略¶

技术分析是股票投资中的主要方法之一，ricequant支持的talib库可以很方便的为开发者计算技术指标。

talib目前支持的指标列表：http://ta-lib.org/function.html

相信大家已经学会使用回测平台运行简单策略了。接下来我们一起挑战自我写一个技术指标策略吧。

在写策略之前，我们需要找到一个idea，不需要多么复杂高端，自圆其说即可。比如反转策略，逻辑虽然十分简单，有时也能有很好的回测结果。

我们这就随意的选择了一个看起来比较萌的技术指标：TR(波幅)

$$TR_t = max(|\ High_t - Low_t\ |, |\ High_t - Close_{t-1}\ |, |\ Close_{t-1} - Low_t\ |)$$

难以直观理解数学公式在股市中的意义也没关系，根据百度的教导，我们知道TR描述的是股价的震荡幅度，在通常情况下，波幅会保持在一定常态下，但当有主力资金进出时，波幅变化会加剧，进入异常区间。

我们推断，当价格下跌十分剧烈时，受到羊群效应的影响，股价可能会进一步下跌一段时间，因此应该卖出持有的股票。这个逻辑是十分通顺的。那么根据这个设想，我们就确定了卖出条件。

那么买入什么股票呢？权重是多少呢？

在这里我们简单的选取每日选取中证500中市值最小的10只股票，按市值由小到大依次由大到小赋权，每个月第十个交易日更新中证500股票池，并在持仓股票昨日收盘价-前日收盘价 < -0.95倍的昨日TRANGE时卖出所持有的全部股票。

整个过程逻辑自洽，只等待我们将代码编写出来，用回测验证结果了。

按照已有思路，我们轻松的写出如下代码，并运行回测：

In [ ]:

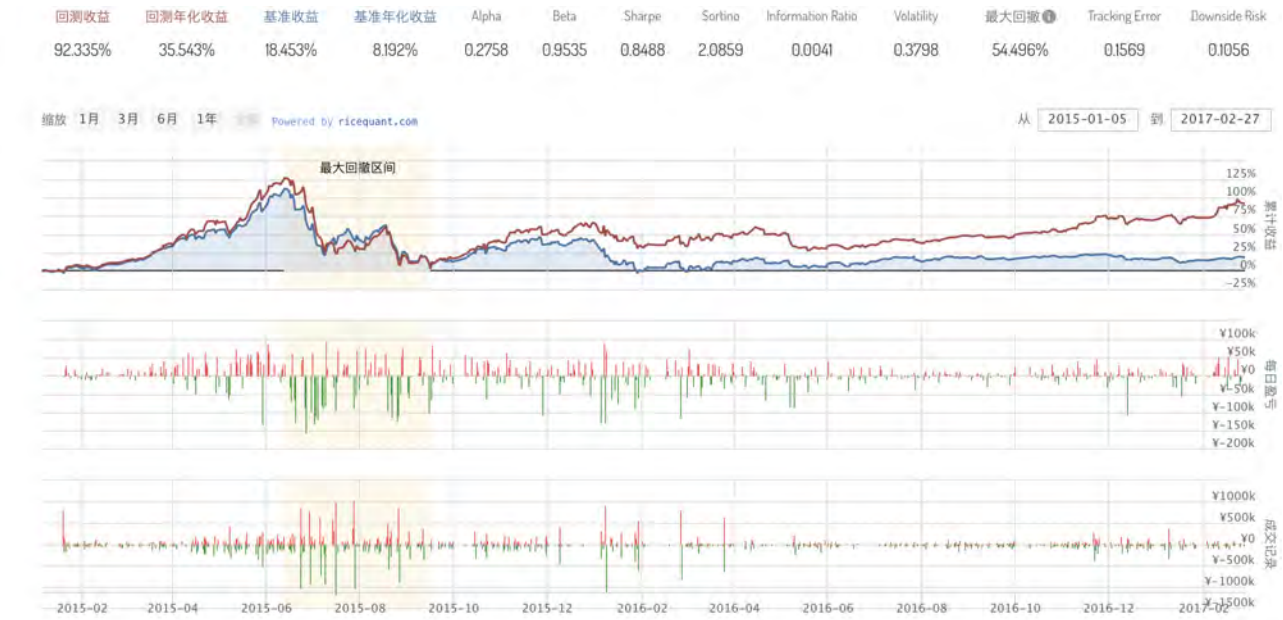
```
import pandas as pd
import numpy as np
import time
import talib

# 获取过去30个交易日的fields = data_index的数据
def get_data(stock,data_index):
    # 获取过去30个交易日的fields = data_index的数据，并保存在res中
    res = history_bars(stock,30,'1d',data_index)
    # 将type(res) 转为 numpy.array格式
    res = np.array(res)
    # 返回res的值
    return res

# 生成保存下单权重的列表
def weight_decide(res):
    # 申明alist是一个列表
    alist = []
    for i in range(len(res)):
    # 向alist中添加一个dict，key是股票名称，value是购买的比例，根据市值由小到大rank并由大到小赋权。
        alist.append({res[i] : (2*(len(res)-i))/((1+len(res))*len(res))})
    # 返回alist
    return alist

# 生成将要购买的股票池
def choose_stocks(context):
    # 选出context.stock_list中市值最小的10支股票
    market_cap = get_fundamentals(query(fundamentals.eod_derivative_indicator.market_cap).filter(fundamentals.stockcode.in_(context.s
    # 取出这些股票代码，并赋给context.chosen
    context.chosen = list(market_cap.columns)
    pass

# 获得中证500成分股股票列表
def get_stock_list(context,bar_dict):
    context.stock_list = index_components('中证500(深)')
    pass

def init(context):
    # 申明自定义全局变量 context.chosen
    context.chosen = []
    # 申明自定义全局变量 context.price_list
    context.price_list = pd.DataFrame() # 持仓股票的若干天收盘价
    # 申明自定义全局变量 context.stock_list
    context.stock_list = [] # 中证500 成分股股票池
    # 每月第10个交易日调用get_stock_list函数，更新中证500股票池。
    scheduler.run_monthly(get_stock_list,tradingday=10)
    pass

def before_trading(context):
    # 将中证500成分股赋给context.stock_list
    context.stock_list = index_components('中证500(深)')
    # 生成将要购买的股票池
    choose_stocks(context)
    # 时间戳转换，start_date(st)和end_date(ed)
    ts = time.strptime(str(context.now),"%Y-%m-%d %H:%M:%S")
    st = int(time.mktime(ts)) - 24*3600*5
    st = time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(st))
    ed = int(time.mktime(ts)) - 24*3600*1
    ed = time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(ed))
    # 如果目前有持仓：
    if list(context.portfolio.positions.keys()):
    # 取出持仓股票在st至ed时间之间的收盘价，结果传入context.price_list
        context.price_list = get_price(list(context.portfolio.positions.keys()),start_date=st,end_date=ed,fields='close')
    pass
```

```
def handle_bar(context,bar_dict):
    # 遍历持仓股票
    for stock in list(context.portfolio.positions.keys()):
        # 获得近30个交易日的TRANGE技术指标，并转格式为pd.Series()
        res = pd.Series(talib.TRANGE(get_data(stock,'high'), get_data(stock,'low'), get_data(stock,'close')))
        try:
            # 若stock昨日收盘价-前日收盘价 < -0.95倍的昨日TRANGE
            if context.price_list[stock].iloc[-1] - context.price_list[stock].iloc[-2] < -0.95 * res.iloc[-2]:
            # 卖出所有持仓股票（目标仓位为0）
                order_target_percent(stock,0)
        except:
            # 若stock不在context的列名中
            if stock not in context.price_list.columns:
            # 卖出所有持仓股票（目标仓位为0）
                order_target_percent(stock,0)
    # 调用weight_decide函数，生成保存下单权重的列表，并将结果保存在alist中。
    alist = weight_decide(context.chosen)
    for stock in alist:
        # 按权重下单（目标仓位保存在weight_decide的value中）
        order_target_percent(list(stock.keys())[0],list(stock.values())[0])
    pass

def after_trading(context):
    pass
```

我们选择2015-01-01 至 2017-02-27 的日级别数据作为回测事件，并选择中证500作为基准，回测结果如下：

In [3]:

```
from IPython.display import display, Image
display(Image(filename='test_result.jpeg'))
```



介绍日回测和分钟级回测¶

ricequant 除提供日频率的回测外，为满足用户更多可能的需求，还提供更精确的分钟级别回测。

用户进入回测平台后，右侧可以调节回测时间段，账户初始金额，回测频率可选每日 / 每分钟，在更多中还可以更换基准策略和撮合机制，设置滑点等。

In [1]:

```
from IPython.display import display, Image
display(Image(filename='minute_or_day.jpeg'))
```

介绍scheduler ，引入更低频的策略¶

有时我们虽然使用日频回测，但是并不希望每天都运行代码的全部，正如在分钟级别回测时，我们很可能希望每天只更新（计算）一次股票池，而不是每一分钟都运行一次更新股票池的程序，这会大大增加回测的时间成本。这时，scheduler函数为我们提供了一份解决方案。

注： scheduler函数只能在init中运行。

In [ ]:

```
# 定义一个更新股票池的函数
def update_universe(context,bar_dict): # 该函数必须且仅能包含context和bar_dict两个输入参数
    # 选出当日沪深300成分股，并传给context
    context.stock_list = index_components('沪深300')
    pass

# 每天运行
scheduler.run_daily(update_universe)

# 每周运行
# 定时在每周x运行
scheduler.run_weekly(update_universe,weekday=x)
# 定时在每周第t个交易日运行
scheduler.run_weekly(update_universe,tradingday=t)

# 每月运行
# 定时在每月第t个交易日运行
scheduler.run_monthly(update_universe,tradingday=t)
```

在分钟级别回测中，我们还可以使用time_ruler 更精确的使用定时器scheduler

In [ ]:

```
# 每日的开市后20分钟运行：
scheduler.run_daily(function, time_rule=market_open(minute=20)) #注：如果不设置time_rule 默认开市后一分钟运行

# 每周的第t个交易日闭市前1小时运行：
scheduler.run_weekly(function, tradingday=t, time_rule=market_close(hour=1))

# 每天开始交易前运行：
scheduler.run_daily(function, time_rule='before_trading')
```

介绍context 里面的成员，以及相关的设置¶

context 是一组全局变量，用于在init，before_trading，handle_bar，after_trading间传递参数。 我们提供了6个context函数分别是:

- context.now —— handle_bar中当前bar的时间
- context.portfolio —— 投资组合信息
- context.stock_portfolio —— 股票投资组合信息
- context.future_portfolio —— 期货投资组合信息
- context.run_info —— 策略运行信息
- context.universe —— 策略合约池

In [ ]:

```
# 当前bar时间
context.now

# 投资组合信息 — 以positions为例
context.portfolio.positions # 返回一个dict

# 股票投资组合信息 — 以投资组合至今累积收益率为例
context.stock_portfolio.total_returns

# 期货投资组合信息 — 以当日盈亏为例
context.future_portfolio.daily_pnl

# 策略运行信息 — 以benchmark为例
context.run_info.benchmark

# 策略合约池
context.universe
# 更新策略合约池
update_universe
```

同时，context也可用于用户自定义的参数传递。 自定义的context变量需要先在init函数中申明。

In [ ]:

```
# 自定义选股股票池为当日沪深300成分股
context.stock_list = index_components('沪深300')
```

下单管理¶

ricequant 为用户提供多样的下单方式。

股票¶

- order_shares —— 指定股数交易
- order_lots —— 指定手数交易
- order_value —— 指定价值交易
- order_percent —— 指定比例下单
- order_target_value —— 目标价值下单
- order_target_percent —— 目标比例下单

期货¶

- buy_open —— 买开
- sell_close —— 平多仓
- sell_open —— 卖开
- buy_close —— 买开
- cancel_order —— 撤单
- get_open_orders —— 拿到未成交订单信息

In [ ]:

```

```

通过期货与股票策略对比，带你了解米筐撮合机制¶

instrument ¶

可以通过all_instruments函数来获取我们可以市场上所有合约的信息，目前只支持中国市场。

In [ ]:

```
all_instruments(type=None)
```

当然，有事我们只需要查询某一类合约的信息，那么只需要传入如下种类的的type参数：

CS¶

```
Common Stock, 即股票
```

ETF¶

```
Exchange Traded Fund, 即交易所交易基金
```

LOF¶

```
Listed Open-Ended Fund，即上市型开放式基金
```

FenjiMu ¶

```
Fenji Mu Fund，即分级母基金
```

FenjiA ¶

```
Fenji A Fund，即分级A类基金
```

FenjiB ¶

```
Fenji B Funds，即分级B类基金
```

INDX¶

```
Index，即指数
```

Future ¶

```
Futures，即期货，包含股指、国债和商品期货
```

获取获取中国市场所有股票信息：

In [1]:

```
all_instruments('CS').head()#取前5行
```

Out[1]:

	abbrev_symbol	board_type	concept_names	de_listed_date	exchange	industry_code	industry_name	lis
0	HGGF	MainBoard	融资融券|业绩预降	0000-00-00	XSHE	C31	黑色金属冶炼及压延加工业	19
1	ZHZB	GEM	业绩预升|陕甘宁	0000-00-00	XSHE	C38	电气机械及器材制造业	20
2	SYD	UNKNOWN	4G概念|海峡西岸|摘帽概念|宽带提速	0000-00-00	XSHE	C39	计算机、通信和其他电子设备制造业	20
3	XJH	UNKNOWN	O2O模式|内贸规划	0000-00-00	XSHE	F52	零售业	20
4	RSJT	UNKNOWN	海峡西岸	0000-00-00	XSHE	C36	汽车制造业	20

获取获取中国市场所有期货信息：

In [2]:

```
all_instruments('Future').head()#取前5行
```

Out[2]:

	abbrev_symbol	contract_multiplier	de_listed_date	exchange	listed_date	margin_rate	maturity_date	ord
0	ZLY1309	10.0	2013-09-13	DCE	2012-09-17	0.05	2013-09-13	P13
1	DE1505	10.0	2015-05-15	DCE	2014-05-19	0.05	2015-05-15	B15
2	LWG1305	10.0	2013-05-15	SHFE	2012-05-16	0.07	2013-05-15	RB
3	ZLY1305	10.0	2013-05-15	DCE	2012-05-16	0.05	2013-05-15	P13

	abbrev_symbol	contract_multiplier	de_listed_date	exchange	listed_date	margin_rate	maturity_date	ord
4	XWB1709	500.0	2017-09-14	DCE	2016-09-19	0.20	2017-09-14	FB

context ¶

context是储存策略自定义参数、设置、仓位、投资组合信息的全局变量，属性通过点标记（"."）来取到

例如将查询到的财务数据添加进去context里面保存，并且之后可以在handle_bar里使用，即你的算法策略之后可以使用这些保存在dataframe中的数据：

In [ ]:

```
context.fundamental_df = fundamental_df
```

在这里，我们还将介绍一下context的一些有用的属性

now （当前时间）¶

使用以上的方式就可以在handle_bar中拿到当前的bar的时间，比如day bar的话就是那天的时间，minute bar的话就是这一分钟的时间点。 返回数据类型为datetime.datetime

context.now

portfolio （投资组合信息）¶

该投资组合在单一股票或期货策略中分别为股票投资组合和期货投资组合。在股票+期货的混合策略中代表汇总之后的总投资组合。

context.portfolio

portfolio还有以下属性

starting_cash ¶

```
float    初始资金，为子组合初始资金的加总
```

cash ¶

```
float    可用资金，为子组合可用资金的加总
```

total_returns ¶

```
float    投资组合至今的累积收益率。计算方法是现在的投资组合价值/投资组合的初始资金
```

daily_returns ¶

```
float    投资组合每日收益率
```

daily_pnl ¶

```
float    当日盈亏，子组合当日盈亏的加总
```

market_value ¶

```
float    投资组合当前的市场价值，为子组合市场价值的加总
```

portfolio_value ¶

```
float    总权益，为子组合总权益加总
```

pnl ¶

```
float    当前投资组合的累计盈亏
```

start_date ¶

datetime.datetime	策略投资组合的回测/实时模拟交易的开始日期

annualized_returns ¶

float	投资组合的年化收益率

positions ¶

dict	一个包含所有仓位的字典，以order_book_id作为键，position对象作为值

stock_portfolio （股票投资组合信息）¶

获取股票投资组合信息。

在单独股票类型策略中，内容与portfolio一致，都代表当前投资组合；在期货+股票混合策略中代表股票子组合；在单独期货策略中，不能被访问。

In [ ]:

```
context.stock_portfolio
```

stock_portfolio还有以下属性

starting_cash ¶

float	回测或实盘交易给算法策略设置的初始资金

cash ¶

float	可用资金

total_returns ¶

float	投资组合至今的累积收益率。计算方法是现在的投资组合价值/投资组合的初始资金

daily_returns ¶

float	当前最新一天的每日收益

daily_pnl ¶

float	当日盈亏，当日投资组合总权益-昨日投资组合总权益

market_value ¶

float	投资组合当前所有证券仓位的市值的加总

portfolio_value ¶

float	总权益，包含市场价值和剩余现金

pnl ¶

float	当前投资组合的累计盈亏

start_date ¶

datetime.datetime	策略投资组合的回测/实时模拟交易的开始日期

annualized_returns ¶

float	投资组合的年化收益率

positions ¶

dict	一个包含所有证券仓位的字典，以order_book_id作为键，position对象作为值

dividend_receivable ¶

float	投资组合在分红现金收到账面之前的应收分红部分

future_portfolio （期货投资组合信息）¶

获取期货投资组合信息。 在单独期货类型策略中，内容与portfolio一致，都代表当前投资组合；在期货+股票混合策略中代表期货子组合；在单独股票策略中，不能被访问。

In [ ]:

```
context.future_portfolio
```

starting_cash ¶

float	初始资金

cash ¶

float	可用资金

frozen_cash ¶

float	冻结资金

total_returns ¶

float	投资组合至今的累积收益率，当前总权益/初始资金

daily_returns ¶

float	当日收益率 = 当日收益 / 昨日总权益

market_value ¶

float	投资组合当前所有期货仓位的名义市值的加总

pnl ¶

float	累计盈亏，当前投资组合总权益-初始资金

daily_pnl ¶

float	当日盈亏，当日浮动盈亏 + 当日平仓盈亏 - 当日费用

daily_holding_pnl ¶

float	当日浮动盈亏

daily_realized_pnl ¶

float	当日平仓盈亏

portfolio_value ¶

float	总权益，昨日总权益+当日盈亏

transaction_cost ¶

float	总费用

start_date ¶

Date	回测开始日期

annualized_returns ¶

float	投资组合的年化收益率

positions ¶

dict	一个包含期货仓位的字典，以order_book_id作为键，position对象作为值

margin ¶

```
float    已占用保证金
```

run_info （策略运行信息）¶

In [ ]:

```
context.run_info
```

run_id ¶

```
str    标识策略每次运行的唯一id
```

run_type ¶

```
RUN_TYPE    RUN_TYPE.BACKTEST表示当前策略在进行回测，RUN_TYPE.PAPER_TRADING表示当前策略在进行实盘模拟
```

start_date ¶

```
datetime.date    策略的开始日期
```

end_date ¶

```
datetime.date    策略的结束日期
```

frequency ¶

```
str    策略频率，'1d'或'1m'
```

stock_starting_cash ¶

```
float    股票账户初始资金
```

future_starting_cash ¶

```
float    期货账户初始资金
```

slippage ¶

```
float    滑点水平
```

margin_multiplier ¶

```
float    保证金倍率
```

commission_multiplier ¶

```
float    佣金倍率
```

benchmark ¶

```
str    基准合约代码
```

matching_type ¶

```
MATCHING_TYPE    撮合方式，MATCHING_TYPE.NEXT_BAR_OPEN代表以下一bar开盘价撮合，MATCHING_TYPE.CURRENT_BAR_CLOSE代表以当前bar收盘价撮合
```

universe （策略合约池）¶

In [ ]:

```
context.universe
```

在运行update_universe，subscribe或者unsubscribe的时候，合约池会被更新。 需要注意，合约池内合约的交易时间（包含股票的策略默认会在股票交易时段触发）是handle_bar被触发的依据。

subscribe （订阅）¶

鉴于不同合约交易时间的不同（例如股票没有夜间交易，期货一些品种有夜盘交易），您在编写策略的时候需要注意策略的有效运行时间。比如在2015年12月之前，中金所股指期货的交易时间段是09:15~11:30, 13:00~15:15，比A股市场多出了30分钟。在这个时候进行混合回测的时候就需要通过订阅的方式让策

略引擎'知道'handle_bar是要在每天09:16产生第一个bar数据，而不是股票的09:31。

如果您创建的是单一的期货策略，则必须在策略初始化的时候订阅(subscribe)有效期货合约。由于期货有到期日，所以您需要保证在回测期间，始终都有正在交易的合约被订阅。

In [ ]:

```
subscribe(order_book_id)
```

订阅合约行情。该操作会导致合约池内合约的增加，从而影响handle_bar中处理bar数据的数量。 需要注意，用户在初次编写策略时候需要首先订阅合约行情，否则handle_bar不会被触发。

In [ ]:

```
unsubscribe(order_book_id)
```

取消订阅合约行情。取消订阅会导致合约池内合约的减少，如果当前合约池中没有任何合约，则策略直接退出。

bar¶

包含股票、期货数据

order_book_id ¶

```
str    股票代码
```

symbol ¶

```
str    合约简称
```

datetime ¶

```
datetime.datetime    时间戳
```

open ¶

```
float    开盘价
```

close ¶

```
float    收盘价
```

high ¶

```
float    最高价
```

low ¶

```
float    最低价
```

volume ¶

```
float    成交量
```

total_turnover ¶

```
float    成交额
```

prev_close ¶

```
float    昨日收盘价
```

limit_up ¶

```
float    涨停价
```

limit_down ¶

```
float    跌停价
```

isnan ¶

bool	当前bar数据是否有行情。例如，获取已经到期的合约数据，isnan此时为True

suspended ¶

bool	是否全天停牌

prev_settlement ¶

float	昨结算（期货日线数据专用）

settlement ¶

float	结算（期货日线数据专用）

通过bar_dict[order_book_id] 可以拿到某个证券的bar信息，key为order_book_id，value为bar数据。当前合约池内所有合约的bar数据信息都会更新在bar_dict里面

分钟回测有多精准？一起看看snapshot ¶

current.snapshot() 可以返回当前市场快照数据。

- 调用 logger.info(current_snapshot(context.f1))，打印context.f1(主力合约)在策略(分钟级)当前时间的数据：
    - 2014-01-02 09:29:00.00 INFO Snapshot(order_book_id: 'IF1401', datetime: datetime.datetime(2014, 1, 2, 9, 29), open: 2340.0, high: 2342.4, low: 2330.2, last: 2340.4, volume: 31768, total_turnover: 22278751740, prev_close: 2345.0, open_interest: 96452, prev_settlement: 2348.4)
- 我们也可以直接将current_snapshot(context.f1)打印出来：
    - 2014-01-02 09:29:00.00 INFO Snapshot(order_book_id: 'IF1401', datetime: datetime.datetime(2014, 1, 2, 9, 29), open: 2340.0, high: 2342.4, low: 2330.2, last: 2340.4, volume: 31768, total_turnover: 22278751740, prev_close: 2345.0, open_interest: 96452, prev_settlement: 2348.4)

效果相同。

订单到底该如何下，下了之后如何管理¶

我们已经了解了股票策略中的订单管理的方式，但是在期货策略中，管理订单的函数有所不同，分别是：

- buy_open(id_or_ins, amount, style=MarketOrder()) —— 买开仓
- buy_close(id_or_ins, amount, style=MarketOrder()) —— 平多仓
- sell_open(id_or_ins, amount, style=MarketOrder()) —— 卖开仓
- sell_close(id_or_ins, amount, style=MarketOrder()) —— 平空仓

平仓和反向开仓有什么区别呢？在实盘中，证券交易所处理订单的逻辑是优先平仓再处理开仓的，并且反向开仓相同手数，虽然实际上已经锁仓了但是依然占用保证金。因此二者并非等价。

天天都要下单，我们的资产组合里到底有什么？¶

- context.portfolio
- context.stock_portfolio
- context.future_portfolio

投资组合信息包含以下字段：

- starting_cash —— 初始资金
- cash —— 当前持有现金
- total_returns —— 投资组合累积收益率
- daily_returns —— 投资组合每日收益率
- daily_pnl —— 当日盈亏
- market_value —— 投资组合当前的市场价值
- portfolio_value —— 总权益，为子组合总权益加总
- pnl —— 当前投资组合的累计盈亏
- start_date —— 策略投资组合的回测/实时模拟交易的开始日期
- annualized_returns —— 投资组合的年化收益率
- positions —— 一个包含所有仓位的字典，以order_book_id作为键，position对象作为值

以上三个函数都可以获取投资组合中的信息。那么它们有什么区别呢？

- context.stock_portfolio 中储存股票投资组合信息
- context.future_portfolio 中储存期货投资组合信息
- context.portfolio 中保存股票和期货的投资组合信息，在单一股票策略或期货策略里分别与context.stock_portfolio或context.future_portfolio无异，但在混合策略中是二者的总和。

我们如何记录我们的仓位 — portfolio 对象¶

context.portfolio 中有存储仓位信息，这让仓位管理变的异常轻松。

股票和期货分别对应不同的position对象，详情查阅API接口：https://www.ricequant.com/api/python/chn

In [ ]:

```
# 读仓位信息，以期货投资组合的可平空仓量为例
context.future_portfolio.positions['stock_id'].closable_sell_quantity
```

In [ ]:

```

```

策略怎么样，米筐来分析¶

收益概览¶

展示策略的风险调整收益（Sharpe, InformationRatio等）、回测收益图、每日盈亏图以及成交记录图。您自己通过plot画图结果也会在该页面展示，具体展示的页面如下：

In [1]:

```
from IPython.display import display, Image
display(Image(filename='收益概览.png'))
```

其中：

回测收益率:¶

策略在期限内的收益率

$$回测收益 = \frac{期末投资组合总权益 - 期初投资组合总权益}{期初投资组合总权益}$$

In [2]:

```
from IPython.display import display, Image
display(Image(filename='收益.png'))
```

收益	1个月(策略&基准)		3个月(策略&基准)		6个月(策略&基准)		12个月(策略&基准)	
2015-01	6.935%	-5.756%	N/A	N/A	N/A	N/A	N/A	N/A
2015-02	5.53%	4.025%	N/A	N/A	N/A	N/A	N/A	N/A
2015-03	20.234%	13.369%	35.682%	11.145%	N/A	N/A	N/A	N/A
2015-04	22.544%	17.224%	55.487%	38.246%	N/A	N/A	N/A	N/A
2015-05	20.635%	1.912%	77.743%	35.438%	N/A	N/A	N/A	N/A
2015-06	-6.793%	-7.59%	37.788%	10.398%	86.955%	22.702%	N/A	N/A
2015-07	-30.314%	-14.656%	-21.645%	-19.625%	21.831%	11.115%	N/A	N/A
2015-08	-10.22%	-11.778%	-41.686%	-30.423%	3.649%	-5.766%	N/A	N/A
2015-09	0.194%	-4.852%	-37.315%	-28.361%	-13.627%	-20.911%	N/A	N/A
2015-10	19.968%	10.322%	7.916%	-7.395%	-15.442%	-25.569%	N/A	N/A
2015-11	6.716%	0.913%	28.272%	5.928%	-25.199%	-26.298%	N/A	N/A
2015-12	10.032%	4.608%	40.867%	16.46%	-11.697%	-16.569%	65.087%	2.371%
2016-01	-18.345%	-21.008%	-4.12%	-16.613%	3.47%	-22.78%	26.059%	-14.197%
2016-02	-0.829%	-2.325%	-10.899%	-19.289%	14.292%	-14.505%	18.463%	-19.434%

年化收益¶

采用了复利累积以及Actual/365 Fixed的年化方式 计算得到的年化收益。

$$年化收益率 = (1 + R)^{\frac{1}{t}} - 1$$

基准收益率¶

相同条件下，一个简单的买入并持有基准合约策略的收益率（默认基准合约为沪深300指数，这里假设指数可交易，最小交易单位为1）

$$基准收益率 = \frac{买入并持有至期末投资组合总权益 - 期初投资组合总权益}{期初投资组合总权益}$$

阿尔法($alpha/ \alpha$): ¶

CAPM模型表达式中的残余项。表示策略所持有投资组合的收益中和市场整体收益无关的部分，是策略选股能力的度量。当策略所选股票的总体表现优于市场基准组合成分股时，阿尔法取正值；反之取负值。

$$\alpha = E[r_p - [r_f + \beta \cdot (r_b - r_f)]]$$

其中r_p是策略所持有投资组合收益；r_f为无风险投资组合收益；β为CAPM模型中的贝塔系数；$E[\cdot]$表示随机变量的期望

In [3]:

```
from IPython.display import display, Image
display(Image(filename='alpha.png'))
```

ALPHA(阿尔法)	1个月	3个月	6个月	12个月
2015-04	0.0205	0.1226	N/A	N/A
2015-05	0.0975	0.1994	N/A	N/A
2015-06	-0.0022	0.144	0.3274	N/A
2015-07	-0.109	-0.0177	0.0626	N/A
2015-08	0.0224	-0.0831	0.0605	N/A
2015-09	0.0552	-0.0481	0.0614	N/A
2015-10	0.0412	0.125	0.0917	N/A
2015-11	0.0441	0.1555	0.0247	N/A
2015-12	0.0457	0.175	0.0565	0.4757
2016-01	0.0565	0.1245	0.2659	0.3514
2016-02	0.0258	0.1047	0.2976	0.3671
2016-03	-0.0146	0.0737	0.2794	0.3327
2016-04	0.0276	0.0201	0.1975	0.2791
2016-05	-0.1095	-0.1545	0.0354	0.0581
2016-06	0.0223	-0.0787	0.0173	0.0787

贝塔($beta/ \beta$): ¶

CAPM模型中市场基准组合项的系数，表示资产收益对市场整体收益波动的敏感程度

$$\beta = \frac{Cov(r_p, e, r_b, e)}{Var(r_b, e)}$$

其中r_p,e为策略超额收益(策略收益率-无风险组合收益率)；r_b,e为市场基准组合超额收益(市场基准组合收益率-无风险组合收益率)；$Cov(\cdot)$表示协方差；$Var(\cdot)$表示方差

Beta值	解释	举例
$\beta < 0$	投资组合和指数基准的走向通常反方向	反向指数ETF或空头头寸
$\beta = 0$	投资组合和指数基准的走向没有相关性	固定收益产品，他们的走向通常和股市不相关
$0 < \beta < 1$	投资组合和指数基准的走向相同，但是比指数基准的移动幅度更小	稳定的股票，比如制作肥皂的公司的股票，通常和市场的走势相同，但是受到每日
$\beta = 1$	投资组合和指数基准的走向相同，并且和指数基准的移动幅度贴近	蓝筹股，指数中占比重大的股票
$\beta > 1$	投资组合和指数基准的走向相同，但是比指数基准的移动幅度更大	受每日市场消息或是受经济情况影响很大的股票

In [4]:

```
from IPython.display import display, Image
display(Image(filename='beta.png'))
```

BETA(贝塔)	1个月	3个月	6个月	12个月
2015-01	0.212	N/A	N/A	N/A
2015-02	0.3573	N/A	N/A	N/A
2015-03	0.4722	0.3075	N/A	N/A
2015-04	1.0962	0.6872	N/A	N/A
2015-05	1.0566	0.9276	N/A	N/A
2015-06	0.8923	0.965	0.7669	N/A
2015-07	0.9568	0.9707	0.9401	N/A
2015-08	1.1425	1.0042	1.0099	N/A
2015-09	1.376	1.0923	1.0542	N/A
2015-10	1.4305	1.2231	1.0695	N/A
2015-11	1.1063	1.289	1.0698	N/A
2015-12	0.813	1.1044	1.1054	0.9733
2016-01	1.1619	1.091	1.1649	1.0469
2016-02	1.2808	1.1252	1.19	1.0697

风险调整后收益指标($Shaepe\ Ratio$) ¶

衡量策略相对于无风险组合的表现，是策略所获得风险溢价的度量——即如果策略额外承担一单位的风险，可以获得多少单位的收益作为补偿

$$Daily\ Sharpe\ Ratio = \frac{\overline{r}_e}{\sigma_e}$$

$$\overline{r}_e = \frac{1}{n}\sum_{i=1}^{n}[r_p(i) - r_f(i)]$$

$$\sigma_e = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}[r_p(i) - r_f(i) - \overline{r}_e]^2}$$

$$Sharpe\ Ratio = \sqrt{244} \cdot Daily\ Sharpe\ Ratio$$

其中$\overline r_e$为回测期内策略日超额收益均值；n为回测期内交易日数；$r_p(i)$,$r_f(i)$分别为第i个交易日策略所持有投资组合的日收益率以及无风险组合日收益率；σ_e为策略超额收益率的波动率

In [5]:

```
from IPython.display import display, Image
display(Image(filename='sharpe.png'))
```

SHARPE(夏普比率)	1个月	3个月	6个月	12个月
2015-01	0.0321	N/A	N/A	N/A
2015-02	0.025	N/A	N/A	N/A
2015-03	0.3436	0.8039	N/A	N/A
2015-04	0.2215	0.9019	N/A	N/A
2015-05	0.2101	0.9797	N/A	N/A
2015-06	-0.1161	0.3734	1.1478	N/A
2015-07	-0.3063	-0.2738	0.2008	N/A
2015-08	-0.1019	-0.4671	-0.0134	N/A
2015-09	-0.038	-0.4194	-0.2107	N/A
2015-10	0.3253	0.0553	-0.233	N/A
2015-11	0.1024	0.4034	-0.373	N/A
2015-12	0.22	0.8939	-0.2062	0.9952
2016-01	-0.2954	-0.1356	0.0144	0.3813
2016-02	-0.0671	-0.2785	0.2382	0.2763
2016-03	0.3079	-0.24	0.6309	0.1451

索提诺比率($Sortino\ Ratio$): ¶

衡量策略相对于目标收益的表现。其使用下行波动率作为风险度量，因此区别于夏普率。在目前的计算中，我们使用基准组合收益作为目标收益，以此作为区分向上波动和向下波动的标准

$$Daily\ Sortino\ Ratio = \frac{\sqrt{244} \cdot \overline{r}_e}{\sigma_d}$$

$$\overline{r}_e = \frac{1}{n}\sum_{i=1}^{n}[r_p(i) - r_f(i)]$$

$$Sortino\ Ratio = \sqrt{244} \cdot Daily\ Sharpe\ Ratio$$

其中$\overline r_e$为回测期内策略日超额收益率均值；n为回测期内交易日数目；$r_p(1)$,$r_f(i)$分别为第i个交易日策略所持有投资组合的日收益率以及无风险组合日收益率；σ_d为策略年化下行波动率。

In [8]:

```
from IPython.display import display, Image
display(Image(filename='sortino.png'))
```

SORTINO(索提诺比率)	1个月	3个月	6个月	12个月
2015-01	0.135	N/A	N/A	N/A
2015-02	0.1062	N/A	N/A	N/A
2015-03	2.1256	2.3804	N/A	N/A
2015-04	1.6965	3.6154	N/A	N/A
2015-05	2.8811	5.2764	N/A	N/A
2015-06	-1.3444	2.3621	4.3912	N/A
2015-07	-1.7242	-1.1901	0.6296	N/A
2015-08	-1.0893	-2.077	-0.045	N/A
2015-09	-0.2584	-1.7303	-0.687	N/A
2015-10	5.2859	0.2964	-0.7895	N/A
2015-11	1.6903	2.0029	-1.2405	N/A
2015-12	1.3357	5.5194	-0.6499	2.3501
2016-01	-5.7471	-1.025	0.0593	0.9655
2016-02	-1.1993	-2.1773	1.0051	0.7169
2016-03	2.1308	-1.8414	3.1748	0.3737

信息比率($Information\ Ratio$) ¶

衡量策略相对于市场基准组合的表现。一般用于评估纯多头的主动交易策略（包括阿尔法策略和基准择时策略）。需要注意的是，信息率不适用于多-空结合的对冲策略的表现评估

$$Daily\ Information\ Ratio = \frac{\sqrt{244} \cdot \overline{r}_{pa}}{\sigma_t}$$

$$\overline{r}_{pa} = \frac{1}{n}\sum_{i=1}^{n}[r_p(i) - r_b(i)]$$

$$Information\ Ratio = \sqrt{244} \cdot Daily\ Information\ Ratio$$

其中$\overline r_{pa}$为回测期主动日收益率均值；n为回测期内交易日数目；$r_p(i)$,$r_b(i)$分别为第i个交易日策略所持有投资组合的日收益率以及基准组合日收益率；\sigma_t为策略跟踪误差。

在我们提供的三个风险调整后收益指标中，信息率用于评估投资组合相对于市场基准组合的表现，一般适用于纯多头的主动交易策略（包括阿尔法策略和基准择时策略）；夏普率用于评估投资组合相对于无风险组合的表现，一般适用于多-空结合的交易策略（例如市场中性策略或配对交易策略），或没有公认市场基准组合的投资品种的交易策略（例如期货CTA策略）。

索提诺比率使用下行波动率作为风险度量，因而有别于信息率和夏普率。下行波动率区分了收益向上波动和向下波动两种情况，并认为收益向下波动才代表风险。因此，索提诺比率的优点，在于其使用的风险度量更为切合我们实际投资中面对的风险；而其缺点则是不如信息率和夏普率常用，认知度较低，且其目标收益（区分收益波动是向上还是向下的标准）的设定是任意的，并不依赖于任何基准组合（不同于信息率和夏普率）。因此，在横向对比不同策略或基金业绩时，我们需要使用统一的目标收益来区分向上波动和向下波动。在实际计算中，我们以无风险组合收益作为索提诺比率的目标收益。

In [7]:

```
from IPython.display import display, Image
display(Image(filename='information ratio.png'))
```

Information Ratio(信息比率)	1个月	3个月	6个月	12个月
2015-01	0.0667	N/A	N/A	N/A
2015-02	0.0204	N/A	N/A	N/A
2015-03	0.0451	0.029	N/A	N/A
2015-04	0.0302	0.0196	N/A	N/A
2015-05	0.1162	0.0365	N/A	N/A
2015-06	0.0065	0.0307	0.0211	N/A
2015-07	-0.0617	-0.0013	0.0045	N/A
2015-08	0.0131	-0.014	0.0044	N/A
2015-09	0.027	-0.0082	0.0041	N/A
2015-10	0.1017	0.0193	0.0058	N/A
2015-11	0.0715	0.0273	0.0017	N/A
2015-12	0.0378	0.0371	0.0033	0.0076
2016-01	0.0295	0.024	0.0144	0.0065
2016-02	0.0358	0.0196	0.0168	0.0066
2016-03	0.0048	0.0111	0.0167	0.0058

跟踪误差($Tracking\ Error\ ,\sigma_t$) ¶

纯多头主动交易策略（阿尔法策略和基准择时策略）收益和市场基准组合收益之间差异的度量。跟踪误差越大，意味着策略所持有投资组合偏离基准组合的程度越大。需要注意，跟踪误差不适用于多-空结合的对冲策略的风险评估

$$\sigma_t = \sqrt{\frac{244}{n-1}\sum_{i=1}^{n}[r_{pa}(i) - \overline{r}_{pa}]^2}$$

$$r_{pa}(i) = r_p(i) - r_b(i)$$

其中，n为回测期内交易日数量；$r_p(i)$,$r_b(i)$分别表示第i个交易日策略所持有投资组合的日收益率和基准组合的日收益率

年化下行波动率($Downside\ Risk,\sigma_d$) ¶

相比波动率，下行波动率对收益向下波动和向上波动两种情况做出了区分，并认为只有收益向下波动才意味着风险。在实际计算中，我们统一使用基准组合收益为目标收益，作为向上波动和向下波动的判断标准

$$\sigma_d = \sqrt{\frac{244}{n-1}\sum_{i=1}^{n}[r_p(i) - r_b(i)]^2 \cdot I(i)}$$

$$I(i) = \begin{cases} 1, & r_p(i) < r_b(i) \\ 0, & r_p(i) \geq r_b(i) \end{cases}$$

其中，n为回测期内交易日数量；$r_p(i)$,$r_b(i)$分别表示第个交易日策略所持有投资组合的日收益率、基准组合的日收益率；$I(i)$为指示函数(indicator function)，如果第i个交易日策略所持有投资组合收益低于基准组合收益，则标记为1（向下波动），否则标记为0（向上波动）

最大回撤($Max\ Drawdown$) ¶

计算回测期内策略的最低总权益，和此前最高总权益的差距，评估策略表现的稳定性

$$Max\ Drawdown = \frac{\text{回测期内最低总权益出现前的最高总权益} - \text{回测期内最低总权益}}{\text{回测期内的最低总权益}}$$

例如，假定策略在回测开始时，资金为 ¥ 1,000,000，一段时间增加至¥ 1,100,000，其后跌至 ¥ 940,000，然后又回升到 ¥ 1,050,000，再跌至 ¥ 890,000，到回测结束时，剩余资金为 ¥ 1,200,000。则其最大回撤为

$$\frac{1100000 - 890000}{890000} = 23.60\%$$

股息率策略【已更新】

各位小伙伴实在抱歉，这几天不小心失了个恋

难得文艺一点，奈何文笔太差只能引用一下我爱的那部电影了：

You can be as mad as a mad dog at the way things went.Y ou could swear ,curse the fates.But when it comes to the end,you have to let it go.

有些故事结束了，作作作作作作作作作完了以后只能let it go。但有些故事才刚刚开始，比如说，安静的做一个量化界网红

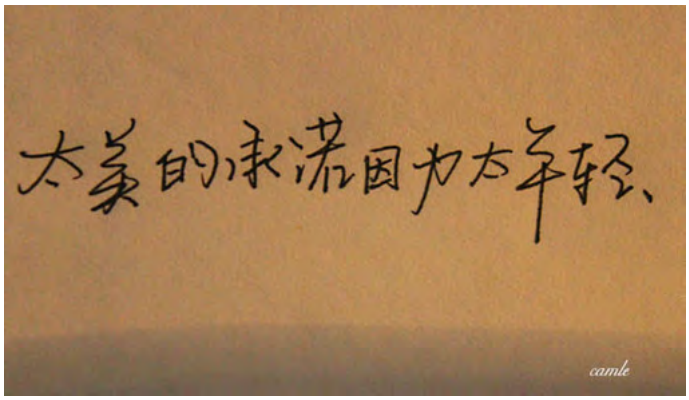
那么今天呢就给大家分享一个互联网流传甚广的“股息率”策略。

作为一个小老股民，曾经也是痴痴傻傻的相信爱情。哦不是相信上市公司讲的故事。

结果呢？

扇贝可以游走，存款可以消失，编不出理由？别怕，还有商誉减值和应收账款计提！

还真是应了那句



所以抛开那些带有颜色和味道的回忆，哦不是财务报表，我要看到你的心（钱）！

如果一定要选出一个最重要的指标来衡量一家公司的价值，
那么我一定会选择股息率！
因为，现金，现金，现金！才能给人安全感嘛！

ok，股息率一个指标就能代表公司的现金流情况，利润情况，估值情况。我们再加一个，
营业收入增长率>5% 这个值大家可以自行调整。来避开那些 价！值！陷！阱！
毕竟就算那个男人再好再有魅力，可他也是个老男人了！！！！（什么鬼）

一个好的爱情观，哦不是投资策略，最重要的应该包含两点：
1.选股
2.仓位

股息率指标天然的完成了这两点，这个策略长这样：

1.选出股息大于6%，营收增长率大于5%的股票加入股票池（未来可将6%改为分级A最高隐含收益率）
2.股票大于10支就等比例满仓，股票小于10支，就每支股票10%剩下的买固定收益（仓位控制）

最后呢，由于股息发放的时间不是固定的，我们用一个dataframe更新和维护股息信息，这样如果今年还没有发放股息，需要用去年的股息来覆盖

先用get_fundamentals 拿到今年至今为止的股息率情况，存起来。

```
fundamental_df = get_fundamentals(
       query(
           fundamentals.eod_derivative_indicator.dividend_yield,
           fundamentals.financial_indicator.inc_operating_revenue,
           fundamentals.eod_derivative_indicator.market_cap
       ).filter(
           fundamentals.financial_indicator.inc_operating_revenue >5
       ).filter(
           fundamentals.eod_derivative_indicator.dividend_yield > 4.5
       )

       .order_by(
           fundamentals.eod_derivative_indicator.dividend_yield .desc()
       ).limit(
           num_stocks
       )
   )
```

接下来创建或者维护 一个dataframe，计算每家家公司的股息，这样如果到了第二年还没有发放股息，去年的股息就被存下来了，如果发放了就被更新了。

```
li=list(fundamental_df.columns.values)
   if context.flag:
       dividend = []
       for stock in li:
         di = float(fundamental_df[stock]['dividend_yield'])/100
         di = di*float(fundamental_df[stock]['market_cap'])
         dividend.append([di])
       div=np.array(dividend)
       context.df = pd.DataFrame(div.T,index=['dividend'],columns=li)
       logger.info(context.df)
       context.flag = False
   else:
       li2 = list(context.df.columns.values)
       for stock in li:
           if stock not in li2:
               di = float(fundamental_df[stock]['dividend_yield'])/100
               di = di*float(fundamental_df[stock]['market_cap'])
               context.df.insert(0,stock,[di])
           else:
               di = float(fundamental_df[stock]['dividend_yield'])/100
               di = di*float(fundamental_df[stock]['market_cap'])
               context.df[stock]['dividend']=di
   #logger.info(context.df)

   stocks=context.df.columns.values
```

最后拿到dataframe 里的股票数据，重新拿到市值并按 股息/市值 计算股息率 并排序。

```
    fundamental_df = get_fundamentals(
        query(
            fundamentals.eod_derivative_indicator.market_cap,fundamentals.financial_indicator.inc_operating_revenue
        ).filter(
            fundamentals.financial_indicator.inc_operating_revenue >5
        ).filter(
            fundamentals.income_statement.stockcode.in_(stocks)
        )
    )
    stocks=fundamental_df.columns.values

    dividend_yield=[]
    for stock in stocks:
        rate=context.df[stock]['dividend']/float(fundamental_df[stock]['market_cap'])
        dividend_yield.append(rate)

    df = pd.DataFrame(dividend_yield,index=stocks,columns=['dividend_yield'])
    df=df[df['dividend_yield']>0.06]
    logger.info(df)

    context.fundamental_df = fundamental_df

    context.stocks = df.T.columns.values
```

这样我们就做到了用去年保存下来的股息来代替今年可能是空白的股息了，不会错过任何一个对的人哦不是对的股票!

https://www.ricequant.com/community/topic/1009/

海龟交易系统的Python 完全版

@zhaoyang-chen 海龟的Python版出炉。
为方便对比，这里把java、python两种语言代码同时贴出，回测时间及初始资金均使用页面默认的20140104-20150104，100000.0软妹币。

turtle_java

```
public class TurtleOriginalStrategy implements IHStrategy {
  Core talibCore;

//定义全局变量
  static int tradedayNum = 0;
  static double unit = 0;
  static double atr = 0;
  static String tradingSignal = "start";
  static String preTradingSignal = "";
  static int units_hold_max = 4;
  static int units_hold = 0;
  static double quantity = 0;
  static double max_add = 0;
  static double firstOpenPrice = 0;

  //计算最大最小值
  public double[] getExtrememem(double[] arrayHighPriceResult, double[] arrayLowPriceResult) {
      DescriptiveStatistics forMax = new DescriptiveStatistics();
      for (int i = 0; i < arrayHighPriceResult.length-1; i++) {
          forMax.addValue(arrayHighPriceResult[i]);
      }
      double maxResult = forMax.getMax();

      DescriptiveStatistics forMin = new DescriptiveStatistics();
      for (int i = 0; i < arrayLowPriceResult.length-1; i++) {
          forMin.addValue(arrayLowPriceResult[i]);
      }
      double minResult = forMin.getMin();

      double[] forExtremum = new double[2];
      forExtremum[0] = maxResult;
      forExtremum[1] = minResult;
      return forExtremum;
  }
  //计算Atr以及单位
  public double[] getAtrAndUnit(double[] atrArrayResult, MInteger atrLengthResult, double portfolioValueResult) {
      double atr = atrArrayResult[atrLengthResult.value-1];
      double unit = Math.floor(portfolioValueResult * .01 / atr);
      double[] atrAndUnit = new double[2];
      atrAndUnit[0] = atr;
      atrAndUnit[1] = unit;
      return atrAndUnit;
  }
```

```
//计算止损线价位
public double getStopPrice(double firstOpenPriceResult, int units_hold_result, double atrResult) {
    double stopPrice =  firstOpenPriceResult - 2*atrResult + (units_hold_result-1)*0.5*atrResult;
    return stopPrice;
}



@Override
public void init(IHInformer informer, IHInitializers initializers) {

  talibCore = new Core();


  int openObserveTime = 55;
  int closeObserveTime = 20;
  int atrTime = 20;
  MInteger atrBegin = new MInteger();
  MInteger atrLength = new MInteger();


  String stockId = "CSI300.INDX";
  initializers.instruments((universe) -> universe.add(stockId));


  initializers.events().statistics((stats, info, trans) -> {

      //获取组合总价值，包含市场价值与剩余资金
      double portfolioValue = info.portfolio().getPortfolioValue();


      double[] highPrice = stats.get(stockId).history(openObserveTime+1, HPeriod.Day).getHighPrice();
      double[] lowPriceForAtr = stats.get(stockId).history(openObserveTime+1, HPeriod.Day).getLowPrice();
      double[] lowPriceForExtrememem = stats.get(stockId).history(closeObserveTime+1, HPeriod.Day).getLowPrice();
      double[] closePrice = stats.get(stockId).history(openObserveTime+2, HPeriod.Day).getClosingPrice();

      double closePriceForAtr[] = new double[closePrice.length-1];
      for (int i = 0; i < closePrice.length-1; i++) {
          closePriceForAtr[i] = closePrice[i];
      }


      double[] atrArray = new double[openObserveTime];
      //Talib计算N即ATR
      RetCode retCode = talibCore.atr(0, openObserveTime-1, highPrice, lowPriceForAtr, closePriceForAtr, atrTime, atrBegin, atrLength, atrArra


      double max = getExtrememem(highPrice, lowPriceForExtrememem)[0];
      double min = getExtrememem(highPrice, lowPriceForExtrememem)[1];


      double atr = atrArray[atrLength.value-1];

      informer.info(lowPriceForExtrememem[lowPriceForExtrememem.length - 1]);
      informer.info("#######");
      informer.info(max);
      informer.info(min);
      informer.info(atr);
      informer.info("#######");

      if (tradingSignal != "start") {
          if (units_hold != 0) {
          max_add += 0.5 * getAtrAndUnit(atrArray, atrLength, portfolioValue)[0];
          }
      } else {
          max_add = stats.get(stockId).getLastPrice();
      }

      informer.info(units_hold);

      double curPosition = info.position(stockId).getNonClosedTradeQuantity();
      double availableCash = info.portfolio().getAvailableCash();
      double marketValue = info.portfolio().getMarketValue();


      if (curPosition > 0 & stats.get(stockId).getLastPrice() < getStopPrice(firstOpenPrice, units_hold, atr)) {
          tradingSignal = "stop";
      } else {
          if (curPosition > 0 & stats.get(stockId).getLastPrice() < min) {
              tradingSignal = "exit";
          } else {
              if (stats.get(stockId).getLastPrice() > max_add & units_hold != 0 & units_hold < units_hold_max & availableCash > stats.get(stoc
```

```
                    tradingSignal = "entry_add";
                } else {
                    if (stats.get(stockId).getLastPrice() > max & units_hold == 0) {
                        max_add = stats.get(stockId).getLastPrice();
                        tradingSignal = "entry";
                    }
                }
            }
        }

        //informer.info(tradingSignal);

        atr = getAtrAndUnit(atrArray, atrLength, portfolioValue)[0];
        if (tradedayNum % 5 == 0) {
            unit = getAtrAndUnit(atrArray, atrLength, portfolioValue)[1];
        }
        tradedayNum += 1;

        double quantity = unit;


        if (tradingSignal != preTradingSignal | (units_hold < units_hold_max & units_hold > 1) | tradingSignal == "stop") {


            if (tradingSignal == "entry") {
                quantity = unit;
                if (availableCash > stats.get(stockId).getLastPrice()*quantity) {
                    trans.buy(stockId).shares(quantity).commit();
                    firstOpenPrice = stats.get(stockId).getLastPrice();
                    units_hold = 1;
                    informer.info("entrybuy" + quantity);
                }
            }
            if (tradingSignal == "entry_add") {
                quantity = unit;
                trans.buy(stockId).shares(quantity).commit();
                units_hold += 1;
                informer.info("entry_addbuy" + quantity);
            }


            if (tradingSignal == "stop") {
                if (/*curPosition marketValue*/ units_hold > 0) {
                    trans.sell(stockId).shares(quantity).commit();
                    units_hold -= 1;
                    informer.info("stop" + quantity);
                }
            }
            if (tradingSignal == "exit") {
                if (curPosition > 0) {
                    trans.sell(stockId).shares(curPosition).commit();
                    units_hold = 0;
                    informer.info("exitsell" + curPosition);
                }
            }

        }

        preTradingSignal = tradingSignal;

    });

  }
}
```

结果：

turtle_python

```
import numpy as np
import talib
import math

def getExtrememem(arrayHighPriceResult, arrayLowPriceResult):
    np_arrayHighPriceResult = np.array(arrayHighPriceResult[:-1])
    np_arrayLowPriceResult = np.array(arrayLowPriceResult[:-1])
    maxResult = np_arrayHighPriceResult.max()
    minResult = np_arrayLowPriceResult.min()
    return [maxResult, minResult]

def getAtrAndUnit(atrArrayResult, atrLengthResult, portfolioValueResult):
    atr = atrArrayResult[atrLengthResult-1]
    unit = math.floor(portfolioValueResult * .01 / atr)
    return [atr, unit]

def getStopPrice(firstOpenPriceResult, units_hold_result, atrResult):
    stopPrice =  firstOpenPriceResult - 2*atrResult + (units_hold_result-1)*0.5*atrResult
    return stopPrice


def init(context):
    context.tradedayNum = 0
    context.unit = 0
    context.atr = 0
    context.tradingSignal = 'start'
    context.preTradingSignal = ''
    context.units_hold_max = 4
    context.units_hold = 0
    context.quantity = 0
    context.max_add = 0
    context.firstOpenPrice = 0
    context.s = 'CSI300.INDX'
    update_universe([context.s])
    context.openObserveTime = 55;
    context.closeObserveTime = 20;
    context.atrTime = 20;

def handle_bar(context, bar_dict):
    portfolioValue = context.portfolio.portfolio_value
    highPrice = history(context.openObserveTime+1, '1d', 'high')[context.s]
    lowPriceForAtr = history(context.openObserveTime+1, '1d', 'low')[context.s]
    lowPriceForExtrememem = history(context.closeObserveTime+1, '1d', 'low')[context.s]
    closePrice = history(context.openObserveTime+2, '1d', 'close')[context.s]
    closePriceForAtr = closePrice[:-1]

    atrArray = talib.ATR(highPrice.values, lowPriceForAtr.values, closePriceForAtr.values, timeperiod=context.atrTime)

    maxx = getExtrememem(highPrice.values, lowPriceForExtrememem.values)[0]
    minn = getExtrememem(highPrice.values, lowPriceForExtrememem.values)[1]
    atr = atrArray[-2]


    if (context.tradingSignal != 'start'):
```

```
    if (context.units_hold != 0):
        context.max_add += 0.5 * getAtrAndUnit(atrArray, atrArray.size, portfolioValue)[0]
else:
    context.max_add = bar_dict[context.s].last


curPosition = context.portfolio.positions[context.s].quantity
availableCash = context.portfolio.cash
marketValue = context.portfolio.market_value


if (curPosition > 0 and bar_dict[context.s].last < minn):
    context.tradingSignal = 'exit'
else:
    if (curPosition > 0 and bar_dict[context.s].last < getStopPrice(context.firstOpenPrice, context.units_hold, atr)):
        context.tradingSignal = 'stop'
    else:
        if (bar_dict[context.s].last > context.max_add and context.units_hold != 0 and context.units_hold < context.units_hold_max and avail
            context.tradingSignal = 'entry_add'
        else:
            if (bar_dict[context.s].last > maxx and context.units_hold == 0):
                context.max_add = bar_dict[context.s].last
                context.tradingSignal = 'entry'


atr = getAtrAndUnit(atrArray, atrArray.size, portfolioValue)[0]
if context.tradedayNum % 5 == 0:
    context.unit = getAtrAndUnit(atrArray, atrArray.size, portfolioValue)[1]
context.tradedayNum += 1
context.quantity = context.unit



if (context.tradingSignal != context.preTradingSignal or (context.units_hold < context.units_hold_max and context.units_hold > 1) or context

    if context.tradingSignal == 'entry':
        context.quantity = context.unit
        if availableCash > bar_dict[context.s].last*context.quantity:
            order_shares(context.s, context.quantity)
            context.firstOpenPrice = bar_dict[context.s].last
            context.units_hold = 1


    if context.tradingSignal == 'entry_add':
        context.quantity = context.unit
        order_shares(context.s, context.quantity)
        context.units_hold += 1


    if context.tradingSignal == 'stop':
        if (context.units_hold > 0):
            order_shares(context.s, -context.quantity)
            context.units_hold -= 1


    if context.tradingSignal == 'exit':
        if curPosition > 0:
            order_shares(context.s, -curPosition)
            context.units_hold = 0


context.preTradingSignal = context.tradingSignal
```

结果:

PLUS：

1.没用到python 的一些黑法术的情况下java代码190行，python 代码120行。
2.java编译小猫耳朵猛抖8下，python 编译小猫耳朵猛抖了13下。

https://www.ricequant.com/community/topic/180/

【策略】Dual Thrust 交易策略

一、引言

Dual Thrust系统，具体参见。详解程序化交易Dual Thrust策略-雪球

使用第n-1日（前天）以前N天的数据计算Range，第n-1日（昨天）的开盘价作为Open，第n-1日的收盘价或第n日（今天）的开盘价作为当前价与上界(BuyLine)进行比较。当股票突破上界，则认为该股票今天有较大行情，买入该股票。卖出与止损写的比较随意。

二、Dual Thrust 策略介绍

Dual Thrust是一个趋势跟踪系统，由Michael Chalek在20世纪80年代开发，曾被Future Thruth杂志评为最赚钱的策略之一。Dual Thrust系统具有简单易用、适用度广的特点，其思路简单、参数很少，配合不同的参数、止盈止损和仓位管理，可以为投资者带来长期稳定的收益，被投资者广泛应用于股票、货币、贵金属、债券、能源及股指期货市场等。

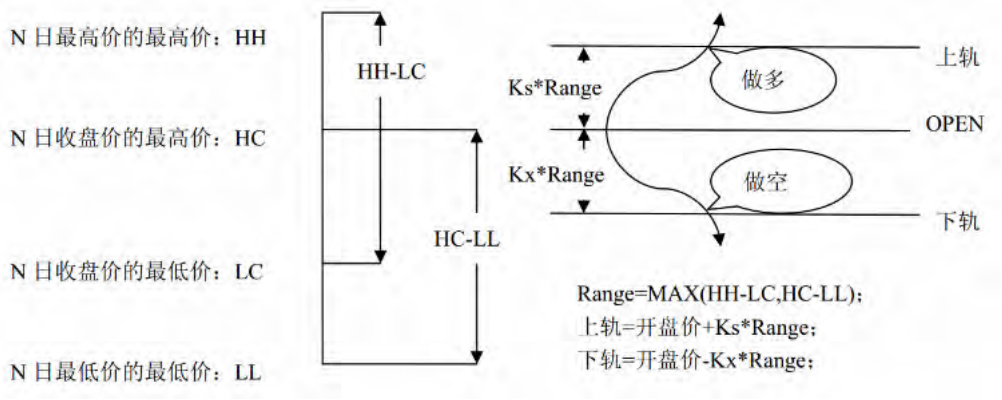
在Dual Thrust交易系统中，对于震荡区间的定义非常关键，这也是该交易系统的核心和精髓。Dual Thrust系统使用Range = Max(HH-LC,HC-LL)来描述震荡区间的大小。其中HH是N日High的最高价，LC是N日Close的最低价，HC是N日Close的最高价，LL是N日Low的最低价。
具体说：

1、首先计算:
(1)N日High的最高价HH, N日Close的最低价LC;
(2)N日Close的最高价HC，N日Low的最低价LL;
(3)Range = Max(HH-LC,HC-LL)

(4)BuyLine = Open + K1Range
(5)SellLine = Open + K2Range

2.构造系统
(1)当价格向上突破上轨时，如果当时持有空仓，则先平仓，再开多仓；如果没有仓位，则直接开多仓；
(2)当价格向下突破下轨时，如果当时持有多仓，则先平仓，再开空仓；如果没有仓位，则直接开空仓；

关键变量的意义如下图所示：



三、本策略思路

1.当股票突破上界（Buyline），则认为该股票今天有较大行情，买入该股票。
2.当股价两天内下跌6%，或者三天内下跌8%则卖出股票。
3.根据大盘止损，如果大盘下跌超过3%则空仓。

https://www.ricequant.com/community/topic/392/

止损策略目录（股票）

[鳄鱼原则]
该法则源于鳄鱼的吞噬方式：猎物越试图挣扎，鳄鱼的收获越多。假定一只鳄鱼咬住你的脚，如果你用手臂试图挣脱脚，则它的嘴巴便会同时咬你的脚与手臂。你越挣扎，便陷得越深。所以，万一鳄鱼咬住你的脚，务必记住：你惟一的生存机会便是牺牲一只脚。

所以当在市场交易时发现与市场背离的情况，或者当亏损到一定地步，已经不可能扭转局势的时候，就应该立刻清仓，避免造成更大的损失。

- 回撤止损

这种方式主要是在初始化时设定回撤的阈值，通过计算出现在的回撤，并且与阈值对比；

如果现在的回撤>规定的阈值，则卖出该股票。

这种方法在股灾和熔断两种情况下都适用。

具体介绍：https://www.ricequant.com/community/topic/3822/

- 阶梯止损（采用动态的止损价格）

这种止损方式的特点在于其止损价格是变动的

如果市价<止损价格，则卖出股票；

这种止损方式比较适用于熔断暴跌的时候。

具体介绍：https://www.ricequant.com/community/topic/3826/

- 时间&收益率止损

这种止损方式同时考虑了时间和收益率两个因素

如果某股票的持有天数已经超过时间阈值并且回报还小于收益率阈值，则卖出该股票。

这种方式比较适用于股灾暴跌时期

具体介绍：https://www.ricequant.com/community/topic/3828/

- 时间&动态比例止损

这种止损方式相比上面的在于其止损价格是动态的，并且其止损价格的计算方式是参照阶梯止损的价格计算方式（有一定的简化）

如果持有天数超过时间阈值&收益率小于动态的止损比例，则卖出该股票

如果目前的收益率跌了超过0.05，则卖出该股票

这种方式在熔断和股灾两种情况下都有一定抗险能力

具体介绍：https://www.ricequant.com/community/topic/3829/

- 利用ART指标止损

这种方式实际上也是一种动态的止损价格方式，通过计算ATR指标，规定止损价格=持有股票之后的最高价-3*ATR；

如果现在的价格小于止损价格，卖出该股票。

这种方式在熔断的时候抗险能力比股灾的时候好一些。

具体介绍：https://www.ricequant.com/community/topic/3832/

- 熔断止损（消极止损）

以沪深300指数为标杆，如果标杆的跌幅超过0.04，则清仓，并且一段时间不交易

这种方式在熔断和股灾两种情况下都有抗险能力

具体介绍：https://www.ricequant.com/community/topic/3833/

- 限价止损/止盈

规定了股价的波动范围，在初始化时预先设定止损乘数，止盈乘数；股价的波动范围是【平均成本*止损乘数，平均成本*止盈乘数】，如果股价超出了区间，实行卖出。

这种方式在熔断和股灾的时候都有一定效果。

具体介绍：https://www.ricequant.com/community/topic/3834/

我有一个“每日判断买入、卖出的策略思路”，如何一步一步转化成策略呢？

一个用户qq问我有一个策略想法：

> 我想咨询一下，我有一个策略是在开盘前算出一个价格，如果高于这个价格就卖出，低于3%以上就买入，比如周日我关注万科，我打算的数字是10，如果万科价格是9.7我就买入，涨到10我就卖出。周一晚上我算出来12，如果周二价格是12*0.97=11.64我就买入
> 这个策略用你们的平台可以回测吗？有没有类似的案例代码，java或者python的都可以。我第一次用你们的平台，还有点不会用

其实这也是一个大家经常遇到的问题，我可能会编程，也懂一些策略思路，但是还是不知道如何从ricequant上开始呢？我们来看看这个问题作为一个范例来深入了解吧~

1. 分析一个策略的用到什么样的数据。从这个策略来看只需要判断前一天的收盘价是否大于或小于某个价格就进行买入或卖出的操作，因此只使用到了市场交易数据，并且是希望拿到“前一天”的收盘价。那么所有的历史市场交易数据我们都可以从 history_bars拿到就可以了。

2. 调仓频率。从这个策略来看是希望在开盘前计算出来是否买入或卖出，因此是最多每日调仓一次即可。那么配合使用before_trading (每天盘前) 运行一次计算判断需要做买入或卖出的操作即可。

3. 下单交易。首先需要了解的是需要做什么样的组合落单，Ricequant提供了较为丰富的下单API，从这个例子中是单独操作一个股票满仓杀入或平仓，注意每日都是满仓操作，因此使用order_target_percent就最为方便了，只需要声明是99%就可以成功将资金下单了，留下1%预留给手续费使用。

那么首先我们先制定一个小的目标（赚取1000万！），在before_trading 中编写一个判断买入或卖出的逻辑，before_trading 是ricequant的策略API中必带的一个回调函数，这里面的逻辑会在每日的开盘前运行（真实时间大概是早上8：30左右），这段代码的comment已经在解释了：

```
# before_trading此函数会在每天交易开始前被调用，当天只会被调用一次
def before_trading(context, bar_dict):
    # 通过 history API 以及 只有1天来拿到前一天的历史收盘价
    hist_series = history('万科A',1, '1d', 'close')

    # 如果不清楚怎么使用的话，打印总是好的。这里返回的hist_series是一个series的类型，存储的是收盘价
    logger.info(str(hist_series))

    # 唯一的一个数据就是昨天的收盘价。
    yesterday_close = hist_series[0]

    # 利用context来储存一个全局变量 - 买入或卖出信号，这个信号由下面的逻辑来决定. 每天开盘前初始化变成‘’
    context.signal = ''

    # 将今日的操作重置为False
    context.fired = False

    # 假设计算出来的一个神奇数字是7.5，那么如果低于3%以上就买入，高于7.5这个价格就卖出。
    # 您可以任意修改这个数值或者通过一些复杂的公式来计算出来这个神奇数字
    magic_number = 7.5
    if yesterday_close <= magic_number * 0.97:
        context.signal = 'buy'
    elif yesterday_close > magic_number:
        context.signal = 'sell'

    # 保持良好的习惯继续打印吧：
    if context.signal != '':
        logger.info('获得调仓信号： ' + context.signal)
```

接着需要考虑下单执行了，这个逻辑只能在handle_bar 中触发，因为before_trading的时候市场还没开始呢！

```
# 你选择的证券的数据更新将会触发此段逻辑，例如日或分钟历史数据切片或者是实时数据切片更新
def handle_bar(context, bar_dict):
    # 开始编写你的主要的算法逻辑

    # 如果您的策略今天没有下过单 （便于和分钟回测进行兼容）。
    if not context.fired:
        if context.signal == 'buy':
            # 0.99 表示用现有资金 99% 全部买入，留一部分给手续费
            order_target_percent(context.stock, 0.99)

            # 今天已经调仓买入了，不再操作
            context.fired = True
        elif context.signal == 'sell':
            # 设置成0就是清仓全部卖出了
            order_target_percent(context.stock, 0)

            # 今天已经调仓卖出了，不再操作
            context.fired = True
```

那么这个策略就搞定了，您只需要后面再修改magic_number那部分的数值即可了。其实写一个策略并不难，可能难得只是写下第一行逻辑...

https://www.ricequant.com/community/topic/1763/

notebook 新功能介绍¶

现在，notebook 新增功能，支持运行回测（包括股票策略回测、期货策略回测及混合策略回测）及参数优化。

也就是说，notebook 中不仅可以方便地运行回测，还可以非常简便调整参数并获取自定义的回测报告。

首先，点击工具栏中最右侧的“生成回测代码模版”，选择“股票策略”，我们将看到如下的默认股票策略回测模版。

In [ ]:

```
%%rqalpha_plus -s 20160301 -e 20160901 --account stock 100000 -fq 1d -p -bm 000001.XSHG
# 上述命令参数可以通过运行 %%rqalpha_plus -h 查看到

def init(context):
    # 策略初始化运行
```

```
    logger.info('init')
    context.counter = 0

def before_trading(context):
    # 每日开盘前运行
    pass

def handle_bar(context, bar_dict):
    # 每个 bar 数据运行
    context.counter += 1
    if context.counter == 1:
        order_shares('000001.XSHE', 100)

def after_trading(context):
    # 每日收盘后运行
    pass
```

回测收益	回测年化收益	Alpha	Beta	Sharpe	Sortino	Information Ratio
0.200%	0.300%	-0.021	0.008	-11.44	-0.148	-1.348
基准收益	基准年化收益	Volatility	MaxDrawdown	Tracking Error	Downside Risk	最大回撤/最大回撤持续期
11.900%	24.800%	0.002	0.100%	0.166	0.128	MaxDD 2016-04-15 00:00:00~2016-05-24 00:00:00, 39 days MaxDDD 2016-04-15 00:00:00~2016-07-12 00:00:00, 88 days

回测模版中的命令参数¶

我们可以通过

```
%rqalpha_plus -h
```

来查看所有的命令参数。

其中比较常用的有：

```
-s 回测起始日期

-e 回测结束日期

--account stock 100000 股票账户初始资金 100000

-fq 回测评率

-p 打印回测结果

-bm 市场基准
```

In [ ]:

```
%rqalpha_plus -h
```

```
Usage: rq-research-kernel [OPTIONS]

  Start to run a strategy

Options:
  -h, --help                      Show this message and exit.
  -f, --strategy-file PATH
  -s, --start-date DATE
  -e, --end-date DATE
  -bm, --benchmark TEXT
  -mm, --margin-multiplier FLOAT
  -a, --account TEXT...           set account type with starting cash
  -fq, --frequency [1d|1m|tick]
  -rt, --run-type [b|p]
  --resume
  --source-code TEXT
  -l, --log-level [verbose|debug|info|error|none]
```

```
  --disable-user-system-log       disable user system log stdout
  --disable-user-log              disable user log stdout
  --locale [cn|en]
  --extra-vars TEXT               override context vars
  --enable-profiler               add line profiler to profile your strategy
  --dividend-reinvestment         enable dividend reinvestment
  --config TEXT                   config file path
  -mc, --mod-config TEXT...       mod extra config
  --stock-t1 / --no-stock-t1      [sys_accounts] enable/disable stock T+1
  --signal                        [sys_simulation] exclude match engine
  -sp, --slippage FLOAT           [sys_simulation] set slippage
  -cm, --commission-multiplier FLOAT
                                  [sys_simulation] set commission multiplier
  -me, --match-engine [current_bar|next_bar|last|best_own|best_counterparty]
                                  [Deprecated][sys_simulation] set matching
                                  type
  -mt, --matching-type [current_bar|next_bar|last|best_own|best_counterparty]
                                  [sys_simulation] set matching type
  --report PATH                   [sys_analyser] save report
  -o, --output-file PATH          [sys_analyser] output result pickle file
  -p, --plot / --no-plot          [sys_analyser] plot result
  --plot-save TEXT                [sys_analyser] save plot to file
  --no-short-stock / --short-stock
                                  [sys_risk] enable stock shorting
  --progress                      [sys_progress]show progress bar
  -rdu, --rqdatad-username TEXT   [ricequant_data] rqdatad username
  -rdpw, --rqdatad-password TEXT  [ricequant_data] rqdatad password
  -rda, --rqdatad-addr TEXT       [ricequant_data] rqdatad server address
  -rdpt, --rqdatad-port INTEGER   [ricequant_data] rqdatad server port
  --night-trading                 [ricequant_data] night trading
  --redis-url TEXT                [ricequant_data] bar & event redis url
  -d, --data-bundle-path PATH     [ricequant_data] data bundle path
```

获取回测报告¶

运行完回测后，报告会自动存储到 report 变量中。可以直接通过 report 变量获取当次回测的结果。

另外 notebook 的 mod 的输出会自动存储在 results 变量中。

In [ ]:

```
results.keys()
```

Out[ ]:

```
dict_keys(['sys_analyser'])
```

In [ ]:

```
report.keys()
```

Out[ ]:

```
dict_keys(['portfolio', 'stock_account', 'stock_positions', 'summary', 'benchmark_portfolio', 'trades'])
```

例如我们想看一下回测的 summary，这将会非常方便：

In [ ]:

```
report.summary
```

Out[ ]:

```
{'STOCK': 100000.0,
 'alpha': -0.021000000000000001,
 'annualized_returns': 0.0030000000000000001,
 'benchmark': '000001.XSHG',
 'benchmark_annualized_returns': 0.248,
 'benchmark_total_returns': 0.11899999999999999,
 'beta': 0.0080000000000000002,
 'cash': 99040.300000000003,
 'downside_risk': 0.128,
 'end_date': '2016-09-01',
 'information_ratio': -1.3480000000000001,
 'max_drawdown': 0.001,
 'run_type': 'BACKTEST',
 'sharpe': -11.435,
 'sortino': -0.14799999999999999,
 'start_date': '2016-03-01',
```

```
'strategy_file': 'strategy.py',
'strategy_name': 'strategy',
'total_returns': 0.002,
'total_value': 100174.3,
'tracking_error': 0.16600000000000001,
'unit_net_value': 1.002,
'units': 100000.0,
'volatility': 0.002}
```

回测功能¶

我们通过一个简单的“麦克•贝利 2-2-2 选股法则”来熟悉一下新的回测功能。

麦克•贝利简介¶

麦克•贝利(Michael Berry)，亚利桑那大学数量分析博士，财务研究专家，对美国股市具有深刻的研究经验，同时也是价值投资的拥护者。麦克•贝利在其发表的研究成果阐述了自己的价值投资理念，并发展出一种简单的选股标 准，称为 2-2-2 法则。

麦克•贝利 2-2-2 选股法则的通用版本¶

A. 股票预期市盈率低于市场平均预期市盈率的“2”分之一

B. 公司预期盈利成长率大于市场平均预估盈利成长率的“2”分之一

C. 股票的市净率小于“2”(此处我们选择“股票的市净率小于市场平均预估市净率的‘2’分之一”)

具体回测代码如下¶

In [ ]:

```
%%rqalpha_plus -s 20160101 -e 20170101 --account stock 100000 -fq 1d -p -bm 000001.XSHG
# 上述命令参数可以通过运行 %%rqalpha_plus -h 查看到
import numpy as np
import pandas as pd
import math
from pandas import Series
import statsmodels.api as sm
import talib as tb
# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    # 在context中保存全局变量
    context.num = 20
    # 设置全局计数器，用于调仓次数的计数
    context.count = 0
    #context.stocks = []
    # 实时打印日志
    #logger.info("RunInfo: {}".format(context.run_info))
    if context.count % 4 == 0:
        scheduler.run_monthly(rebalance,1)
        context.count +=1

# before_trading此函数会在每天策略交易开始前被调用，当天只会被调用一次
def before_trading(context):
    all_df = get_fundamentals(
            query(
                 fundamentals.eod_derivative_indicator.pb_ratio, # 市净率
                 fundamentals.eod_derivative_indicator.pe_ratio, # 市盈率
                 fundamentals.financial_indicator.earnings_per_share, # 每股收益 EPS
                 fundamentals.eod_derivative_indicator.market_cap # 总市值
                 #盈利成长率=每股业绩=每股收益EPS
            )
                )
    # dropna 返回一个仅含非空数据和索引值的 Series
    all_df = all_df.dropna(axis = 1,how = 'any')

    # 转置
    all_df = all_df.T

    # shape 用于读取矩阵的长度
    n = max(np.shape(all_df.values))
    #print(sum(all_df.values))
    # 计算（1/2 * 市场平均市净率）
    context.pb = (sum(all_df.values)/(2*n))[0]
    # 计算（1/2 * 市场平均市盈率）
    context.pe = (sum(all_df.values)/(2*n))[1]
    # 计算（1/2 * 市场平均每股收益）
    context.eps = (sum(all_df.values)/(2*n))[2]

    #print('pb='+str(context.pb))
    #print('pe='+str(context.pe))
```

```
    #print('eps='+str(context.eps))


    all_df = all_df[all_df['pb_ratio']<2]
    all_df = all_df[all_df['pe_ratio']<context.pe]
    all_df = all_df[all_df['earnings_per_share']> context.eps]
    all_df = all_df.sort_values(by = 'market_cap',ascending = True)
    #logger.info('all_df'+str(all_df))
    all_df = all_df.head(context.num)




    context.all_df = all_df
    context.stocks = context.all_df.index.values
    #logger.info("选择好的股票列表为: " + str(context.stocks))
# 你选择的证券的数据更新将会触发此段逻辑，例如日或分钟历史数据切片或者是实时数据切片更新
def rebalance(context, bar_dict):
    # 开始编写你的主要的算法逻辑
    # bar_dict[order_book_id] 可以拿到某个证券的bar信息
    # context.portfolio 可以拿到现在的投资组合信息
    # 使用order_shares(id_or_ins, amount)方法进行落单
    # TODO: 开始编写你的算法吧!

    average_weight = 0
    if len(context.stocks) != 0:
        average_weight = 0.999 / len(context.stocks)

        #开始进行调仓
        #先清仓持仓的股票不在新的列表中

    for stock in context.portfolio.positions:   #从已有持仓的股票中选取
        if stock not in context.stocks:         #如果不在新的股票列表里
            order_target_percent(stock,0)       #清仓

    if average_weight != 0:
        for stock in context.stocks:
            order_target_percent(stock,average_weight)

# after_trading函数会在每天交易结束后被调用，当天只会被调用一次
def after_trading(context):
    pass
```

```
[2017-07-17 15:29:20.931011] INFO: system_log: rqdatac use socket ('q-rqdatad', 16003)
2016-01-04 WARN 002582.XSHE 在 2016-01-04 15:00:00 时停牌。
2016-01-04 WARN 002082.XSHE 在 2016-01-04 15:00:00 时停牌。
2016-01-04 WARN 600864.XSHG 在 2016-01-04 15:00:00 时停牌。
2016-02-01 WARN 002582.XSHE 在 2016-02-01 15:00:00 时停牌。
2016-02-01 WARN 002082.XSHE 在 2016-02-01 15:00:00 时停牌。
2016-03-01 WARN 002082.XSHE 在 2016-03-01 15:00:00 时停牌。
2016-05-03 WARN 600665.XSHG 在 2016-05-03 15:00:00 时停牌。
2016-08-01 WARN 000916.XSHE 在 2016-08-01 15:00:00 时停牌。
2016-09-01 WARN 000916.XSHE 在 2016-09-01 15:00:00 时停牌。
2016-10-10 WARN 000916.XSHE 在 2016-10-10 15:00:00 时停牌。
2016-11-01 WARN 000916.XSHE 在 2016-11-01 15:00:00 时停牌。
2016-12-01 WARN 000916.XSHE 在 2016-12-01 15:00:00 时停牌。
2016-12-01 WARN 600480.XSHG 在 2016-12-01 15:00:00 时停牌。
```

获取自定义的回测报告¶

In [ ]:

```
report.summary
```

Out[ ]:


```
{'STOCK': 100000.0,
 'alpha': 0.30499999999999999,
 'annualized_returns': 0.27200000000000002,
 'benchmark': '000001.XSHG',
 'benchmark_annualized_returns': -0.058000000000000003,
 'benchmark_total_returns': -0.058000000000000003,
 'beta': 1.121,
 'cash': 16426.883000000002,
 'downside_risk': 0.067000000000000004,
 'end_date': '2016-12-30',
 'information_ratio': 2.8740000000000001,
 'max_drawdown': 0.19600000000000001,
 'run_type': 'BACKTEST',
 'sharpe': 0.91700000000000004,
 'sortino': 3.5430000000000001,
 'start_date': '2016-01-04',
 'strategy_file': 'strategy.py',
 'strategy_name': 'strategy',
 'total_returns': 0.27000000000000002,
 'total_value': 126976.883,
 'tracking_error': 0.104,
 'unit_net_value': 1.27,
 'units': 100000.0,
 'volatility': 0.26000000000000001}
```


In [ ]:

```
results["sys_analyser"]["trades"][:5]
```

Out[ ]:

	commission	exec_id	last_price	last_quantity	order_book_id	order_id	position_effect	sid
datetime								
2016-01-04 15:00:00	5.0	1500276559	11.52	400	600356.XSHG	1500276558	None	BU
2016-01-04 15:00:00	5.0	1500276560	11.92	400	600051.XSHG	1500276559	None	BU
2016-01-04 15:00:00	5.0	1500276561	7.66	600	002087.XSHE	1500276560	None	BU
2016-01-04 15:00:00	5.0	1500276562	17.04	200	600969.XSHG	1500276561	None	BU
2016-01-04 15:00:00	5.0	1500276563	7.64	600	000789.XSHE	1500276562	None	BU

我们来看一下部分交易记录

In [ ]:

```
report.trades[:5]
```

Out[ ]:

	commission	exec_id	last_price	last_quantity	order_book_id	order_id	position_effect	sid
datetime								
2016-01-04 15:00:00	5.0	1500276559	11.52	400	600356.XSHG	1500276558	None	BU
2016-01-04 15:00:00	5.0	1500276560	11.92	400	600051.XSHG	1500276559	None	BU
2016-01-04 15:00:00	5.0	1500276561	7.66	600	002087.XSHE	1500276560	None	BU

	commission	exec_id	last_price	last_quantity	order_book_id	order_id	position_effect	sid
datetime								
2016-01-04 15:00:00	5.0	1500276562	17.04	200	600969.XSHG	1500276561	None	BU
2016-01-04 15:00:00	5.0	1500276563	7.64	600	000789.XSHE	1500276562	None	BU

In [ ]:

```
from hmmlearn.hmm import GaussianHMM
import numpy as np
from matplotlib import cm, pyplot as plt
import matplotlib.dates as dates
import pandas as pd
import datetime
import warnings
warnings.filterwarnings('ignore')
```

回测功能再认识¶

接下来是一个更为复杂的策略——HMM在股票市场中的应用¶

我们假设隐藏状态数量是6，即假设股市的状态有6种，虽然我们并不知道每种状态到底是什么，但是通过后面的图我们可以看出那种状态下市场是上涨的，哪种是震荡的，哪种是下跌的。可观测的特征状态我们选择了3个指标进行标示，进行预测的时候假设假设所有的特征向量的状态服从高斯分布，这样就可以使用 hmmlearn 这个包中的 GaussianHMM 进行预测了。下面我会逐步解释。

首先导入必要的包:

In [ ]:

```
from hmmlearn.hmm import GaussianHMM
import numpy as np
from matplotlib import cm, pyplot as plt
import matplotlib.dates as dates
import pandas as pd
import datetime
```

测试时间从2005年1月1日到2015年12月31日，拿到每日沪深300的各种交易数据。

In [ ]:

```
beginDate = '2005-01-01'
endDate = '2015-12-31'
n = 6 #6个隐藏状态
data = get_price('CSI300.INDX',start_date=beginDate, end_date=endDate,frequency='1d')
data[0:9]
```

```
WARN: start_date is earlier than 2005-01-04, adjusted
```

Out[ ]:

	open	close	high	low	total_turnover	volume
2005-01-04	994.77	982.79	994.77	980.66	4.431977e+09	7412869.0
2005-01-05	981.58	992.56	997.32	979.88	4.529208e+09	7119109.0
2005-01-06	993.33	983.17	993.79	980.33	3.921015e+09	6288028.0
2005-01-07	983.05	983.96	995.71	979.81	4.737469e+09	7298694.0
2005-01-10	983.76	993.88	993.96	979.79	3.762932e+09	5791697.0
2005-01-11	994.19	997.14	999.55	991.09	3.704076e+09	5849079.0
2005-01-12	996.65	996.75	996.98	989.26	3.093299e+09	5014525.0
2005-01-13	996.08	996.88	999.47	992.70	3.842173e+09	6044065.0
2005-01-14	996.62	988.31	1006.46	987.23	4.162921e+09	7297842.0

拿到每日成交量和收盘价的数据。

In [ ]:

```
volume = data['TotalVolumeTraded']
close = data['ClosingPx']
```

计算每日最高最低价格的对数差值，作为特征状态的一个指标。

In [ ]:

```
logDel = np.log(np.array(data['HighPx'])) - np.log(np.array(data['LowPx']))
logDel
```

Out[ ]:

```
array([ 0.01428574,  0.01764157,  0.01363667, ...,  0.01380317,
        0.01051247,  0.01207808])
```

计算每5日的指数对数收益差，作为特征状态的一个指标。

In [ ]:

```
logRet_1 = np.array(np.diff(np.log(close)))#这个作为后面计算收益使用
logRet_5 = np.log(np.array(close[5:])) - np.log(np.array(close[:-5]))
logRet_5
```

Out[ ]:

```
array([ 0.01449572,  0.00421252,  0.01384836, ..., -0.03007529,
       -0.02652464, -0.02603115])
```

计算每5日的指数成交量的对数差，作为特征状态的一个指标。

In [ ]:

```
logVol_5 = np.log(np.array(volume[5:])) - np.log(np.array(volume[:-5]))
logVol_5
```

Out[ ]:

```
array([-0.23693333, -0.35044388, -0.03957071, ..., -0.57079226,
       -0.67285963, -0.36793342])
```

由于计算中出现了以5天为单位的计算，所以要调整特征指标的长度。

In [ ]:

```
logDel = logDel[5:]
logRet_1 = logRet_1[4:]
close = close[5:]
Date = pd.to_datetime(data.index[5:])
```

把我们的特征状态合并在一起。

In [ ]:

```
A = np.column_stack([logDel,logRet_5,logVol_5])
A
```

Out[ ]:

```
array([[ 0.00849983,  0.01449572, -0.23693333],
       [ 0.00777352,  0.00421252, -0.35044388],
       [ 0.00679663,  0.01384836, -0.03957071],
       ...,
       [ 0.01380317, -0.03007529, -0.57079226],
       [ 0.01051247, -0.02652464, -0.67285963],
       [ 0.01207808, -0.02603115, -0.36793342]])
```

下面运用 hmmlearn 这个包中的 GaussianHMM 进行预测。

In [ ]:

```
model = GaussianHMM(n_components= n, covariance_type="full", n_iter=2000).fit(A)
hidden_states = model.predict(A)
hidden_states
```

Out[ ]:

```
array([5, 5, 5, ..., 0, 0, 0])
```

关于 covariance_type 的参数有下面四种：

spherical：是指在每个马尔可夫隐含状态下，可观察态向量的所有特性分量使用相同的方差值。对应协方差矩阵的非对角为0，对角值相等，即球面特性。这是最简单的高斯分布PDF。 diag：是指在每个马尔可夫隐含状态下，可观察态向量使用对角协方差矩阵。对应协方差矩阵非对角为0，对角值不相等。diag是hmmlearn里面的默认类型。 full：是指在每个马尔可夫隐含状态下，可观察态向量使用完全协方差矩阵。对应的协方差矩阵里面的元素都是不为零。
tied：是指所有的马尔可夫隐含状态使用相同的完全协方差矩阵。

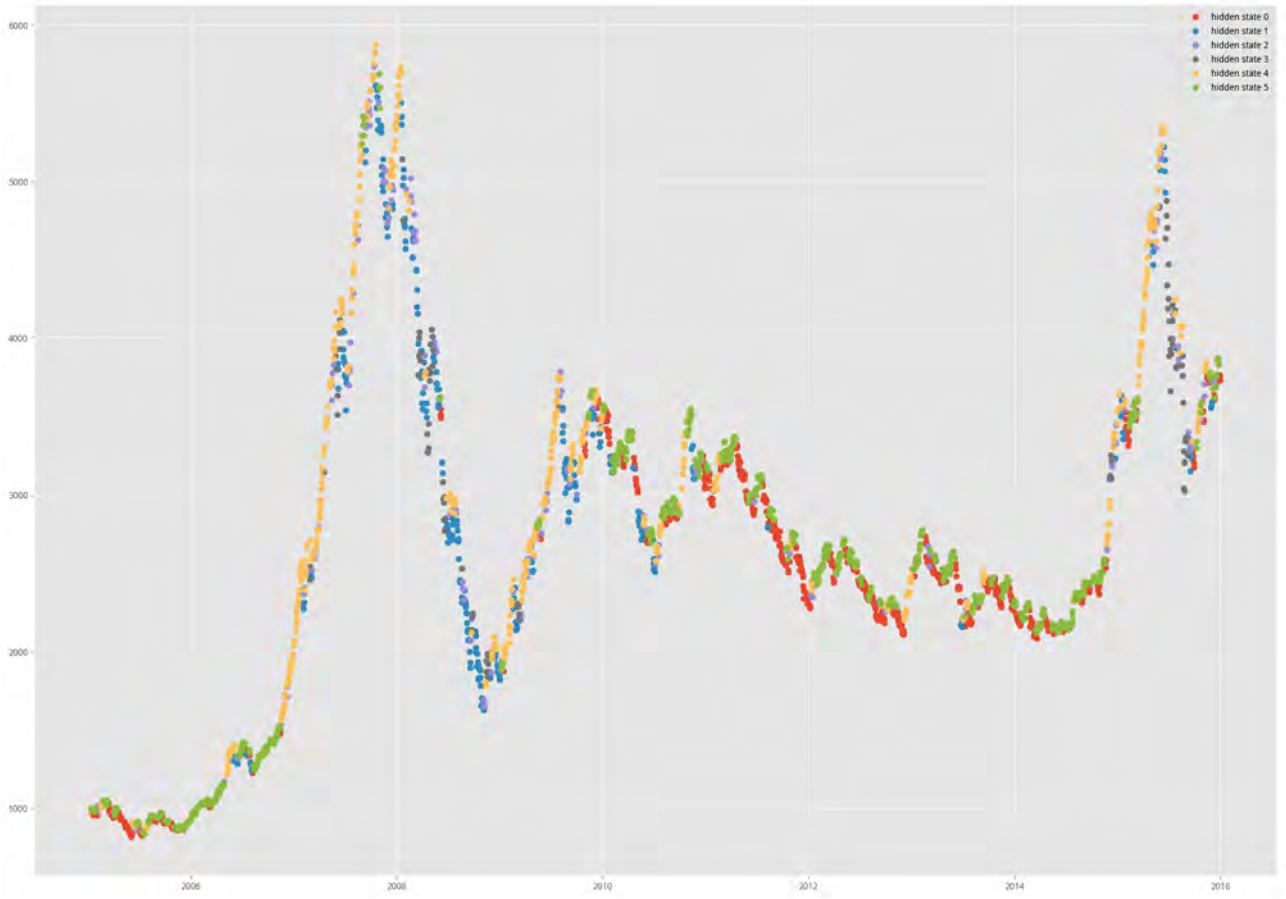
这四种PDF类型里面，spherical, diag和full代表三种不同的高斯分布概率密度函数，而tied则可以看作是GaussianHMM和GMMHMM的特有实现。其中，full是最强大的，但是需要足够多的数据来做合理的参数估计；spherical是最简单的，通常用在数据不足或者硬件平台性能有限的情况之下；而diag则是这两者一个折中。在使用的时候，需要根据可观察态向量不同特性的相关性来选择合适的类型。

转自知乎用户Aubrey Li

我们把每个预测的状态用不同颜色标注在指数曲线上看一下结果。

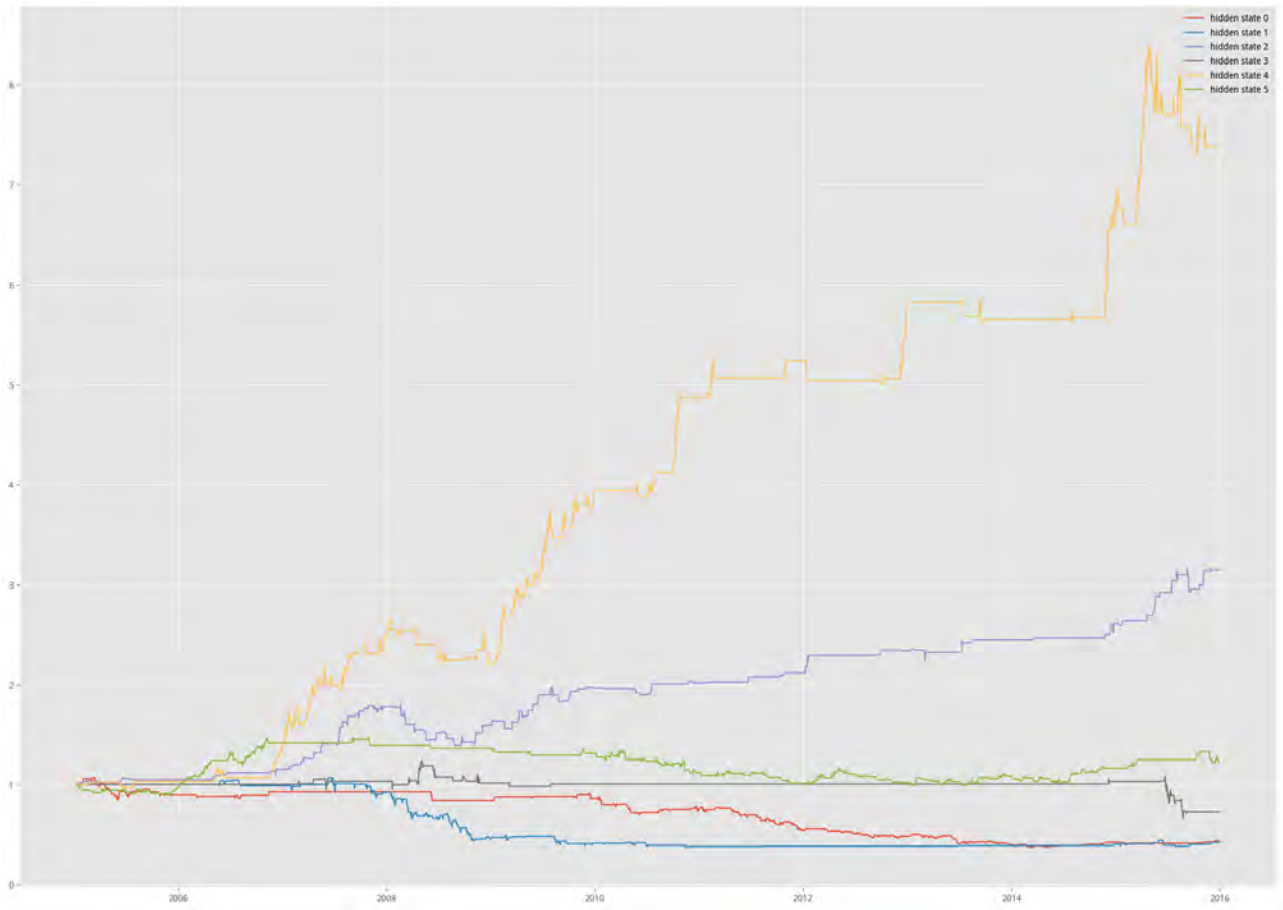
In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(25, 18))
for i in range(model.n_components):
    pos = (hidden_states==i)
    plt.plot_date(Date[pos],close[pos],'o',label='hidden state %d'%i,lw=2)
    plt.legend(loc="left")
```



In [ ]:

```
res = pd.DataFrame({'Date':Date,'logRet_1':logRet_1,'state':hidden_states}).set_index('Date')
plt.figure(figsize=(25, 18))
for i in range(model.n_components):
    pos = (hidden_states==i)
    pos = np.append(0,pos[:-1])#第二天进行买入操作
    df = res.logRet_1
    res['state_ret%s'%i] = df.multiply(pos)
    plt.plot_date(Date,np.exp(res['state_ret%s'%i].cumsum()),'-',label='hidden state %d'%i)
    plt.legend(loc="left")
```



可以看到，隐藏状态1是一个明显的大牛市阶段，隐藏状态0是一个缓慢上涨的阶段(可能对应反弹)，隐藏状态3和5可以分别对应震荡下跌的大幅下跌。其他的两个隐藏状态并不是很明确。由于股指期货可以做空，我们可以进行如下操作：当处于状态0和1时第二天做多，当处于状态3和5第二天做空，其余状态则不持有。

In [ ]:

```
long = (hidden_states==0) + (hidden_states == 1) #做多
short = (hidden_states==3) + (hidden_states == 5)  #做空
long = np.append(0,long[:-1]) #第二天才能操作
short = np.append(0,short[:-1]) #第二天才能操作
```

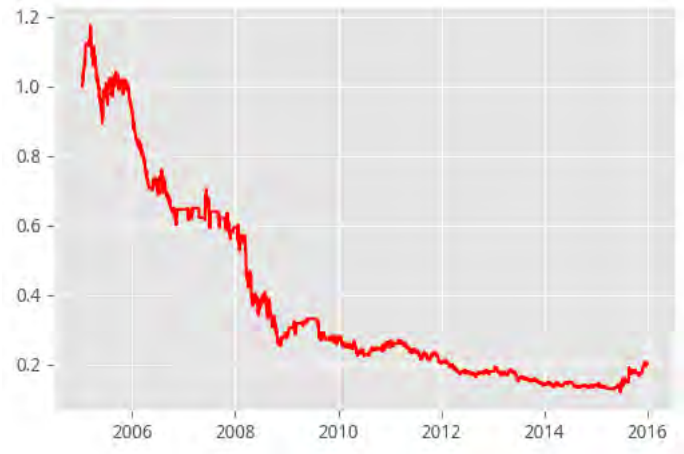
收益曲线图如下：

In [ ]:

```
res['ret'] =  df.multiply(long) - df.multiply(short)
plt.plot_date(Date,np.exp(res['ret'].cumsum()),'r-')
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f5193119470>]
```



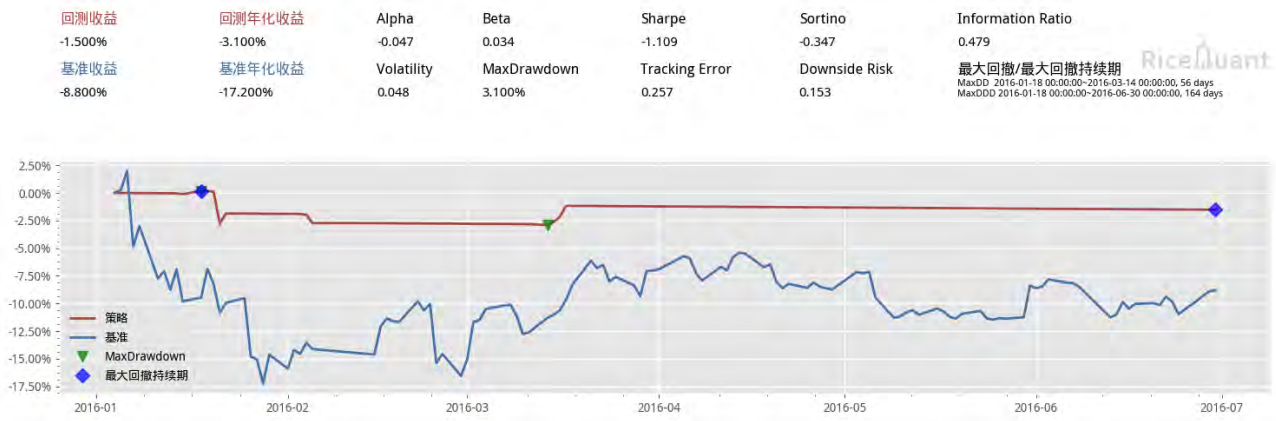
可以看到效果还是很不错的。但事实上该结果是有些问题的。真实操作时，我们并没有未来的信息来训练模型。不过可以考虑用历史数据进行训练，再对之后的数据进行预测。

In [ ]:

```
from rqalpha_plus.api import *
from rqalpha_plus import run_func

from hmmlearn.hmm import GaussianHMM
import numpy as np
from matplotlib import cm, pyplot as plt
import matplotlib.dates as dates
import pandas as pd
import datetime

def init(context):
    # 策略初始化运行
    context.now
    context.stock = '000300.XSHG'
    context.A = 1

def before_trading(context):
    yesterday = (context.now-datetime.timedelta(days = 1)).strftime('%Y-%m-%d')
    ago = (context.now-datetime.timedelta(days = 100)).strftime('%Y-%m-%d')
    data = get_price('000300.XSHG',start_date = ago,end_date = yesterday)
    volume = data['volume']
    close = data['close']
    logDel = np.log(np.array(data['high'])) - np.log(np.array(data['low']))
    logRet_1 = np.array(np.diff(np.log(close)))#这个作为后面计算收益使用
    logRet_5 = np.log(np.array(close[5:])) - np.log(np.array(close[:-5]))
    logVol_5 = np.log(np.array(volume[5:])) - np.log(np.array(volume[:-5]))
    logDel = logDel[5:]
    logRet_1 = logRet_1[4:]
    close = close[5:]
    Date = pd.to_datetime(data.index[5:])
    A = np.column_stack([logDel,logRet_5,logVol_5])
    context.A = A

def handle_bar(context, bar_dict):
    # 每个 bar 数据运行
    hidden_states = model.predict(context.A)
    if hidden_states[-1] == 2:
        order_target_percent(context.stock,1)
    else:
        order_target_percent(context.stock,0)

def after_trading(context):
    # 每日收盘后运行
    pass

config = {
  "base": {
    "start_date": "2016-01-01",
    "end_date": "2016-06-30",
    "benchmark": "000300.XSHG",
    "accounts": {
                'stock': 100000,
            }
  },
  "extra": {
    "log_level": "verbose",
  },
  "mod": {
    "sys_analyser": {
      "enabled": True,
      "plot": True
    }
  }
}

# 您可以指定您要传递的参数
run_func(init=init, before_trading=before_trading, handle_bar=handle_bar, config=config)

from rqdatac import *
```

```
[2017-07-17 15:30:40.251022] DEBUG: system_log:
{'base': {'accounts': {'STOCK': 100000.0},
          'benchmark': '000300.XSHG',
          'data_bundle_path': '/home/user_306344/.rqalpha-plus/bundle',
          'end_date': datetime.date(2016, 6, 30),
          'frequency': '1d',
          'margin_multiplier': 1,
          'persist': False,
          'persist_mode': PERSIST_MODE.REAL_TIME,
```

```
         'resume_mode': False,
         'run_type': RUN_TYPE.BACKTEST,
         'source_code': None,
         'start_date': datetime.date(2016, 1, 1),
         'strategy_file': 'strategy.py'},
 'extra': {'context_vars': None,
          'dividend_reinvestment': False,
          'enable_profiler': False,
          'force_run_init_when_pt_resume': False,
          'is_hold': False,
          'locale': 'zh_Hans_CN',
          'log_level': 'verbose',
          'user_log_disabled': False,
          'user_system_log_disabled': False},
 'mod': {'indicator': {'enabled': True, 'lib': 'rqalpha_mod_indicator'},
        'ricequant_data': {'enabled': True,
                           'lib': 'rqalpha_mod_ricequant_data',
                           'night_trading': False,
                           'priority': 101,
                           'rqdata_client_addr': 'q-rqdatad',
                           'rqdata_client_password': 'research',
                           'rqdata_client_port': 16003,
                           'rqdata_client_retry_cnt': 30,
                           'rqdata_client_username': 'research'},
        'sys_accounts': {'enabled': True},
        'sys_analyser': {'enabled': True, 'plot': True},
        'sys_funcat': {'enabled': False},
        'sys_progress': {'enabled': True},
        'sys_risk': {'enabled': True},
        'sys_simulation': {'enabled': True},
        'sys_stock_realtime': {'enabled': False}},
 'validator': {'cash_return_by_stock_delisted': False, 'close_amount': True},
 'version': '0.1.6',
 'whitelist': ['base', 'extra', 'validator', 'mod']}
[2017-07-17 15:30:40.255155] DEBUG: system_log: 载入 Mod rqalpha_mod_indicator
[2017-07-17 15:30:40.257339] DEBUG: system_log: 载入 Mod rqalpha.mod.rqalpha_mod_sys_simulation
[2017-07-17 15:30:40.259526] DEBUG: system_log: 载入 Mod rqalpha.mod.rqalpha_mod_sys_accounts
[2017-07-17 15:30:40.261536] DEBUG: system_log: 载入 Mod rqalpha.mod.rqalpha_mod_sys_analyser
[2017-07-17 15:30:40.263562] DEBUG: system_log: 载入 Mod rqalpha.mod.rqalpha_mod_sys_risk
[2017-07-17 15:30:40.265675] DEBUG: system_log: 载入 Mod rqalpha.mod.rqalpha_mod_sys_progress
[2017-07-17 15:30:40.267605] DEBUG: system_log: 载入 Mod rqalpha_mod_ricequant_data
[2017-07-17 15:30:40.269890] INFO: system_log: rqdatac use socket ('q-rqdatad', 16003)
```




```
[2017-07-17 15:30:43.601510] DEBUG: system_log: 策略运行成功，正常退出
```

In [ ]:

```
get_price('000300.XSHG',start_date = '2017-07-05',end_date = '2017-07-05')
```

Out[ ]:

	open	close	high	low	total_turnover	volume
2017-07-05	3611.8984	3659.6795	3660.7291	3610.4586	1.203516e+11	8.823242e+09

In [ ]:

```
from rqdatac import *
```

参数调优¶

现在，notebook 不仅可以运行回测，更可以在其中进行参数调优，免去在策略页面中多次修改参数、多次回测的重复劳动。

下面，我们通过一个小例子来一窥究竟。

跨期合约价差研究¶

首先，我们发现 RU1605 和RU1606 的价差始终都处于比较平稳的状态，并且带有剧烈的波动，于是以历史两倍标准差作为动态开平仓线，进行跨期套利。

那么这个两倍标准差究竟是不是最优值呢？我们运用 notebook 的参数优化功能，以步长 0.1 考察标准差从 2 倍到 3 倍的情况。

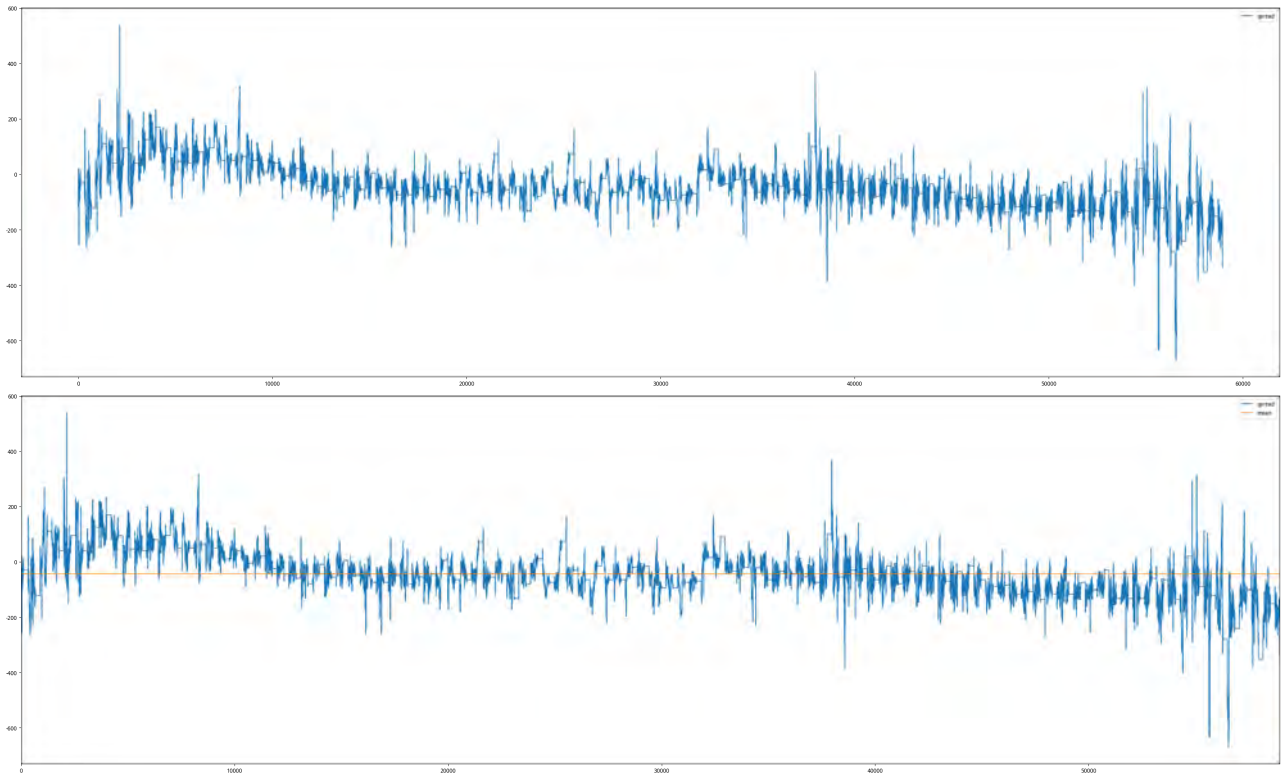
我们把需要调整的参数放入 tasks，回测的主逻辑不变，并发运行回测，最终得到不同参数值下的一系列结果。

In [1]:

```
start_date='20151121'
end_date='20160430'
RU1605=get_price('RU1605',start_date=start_date,end_date=end_date,fields='close',frequency='1m') #当月
RU1606=get_price('RU1606',start_date=start_date,end_date=end_date,fields='close',frequency='1m') # 下月

spread=RU1605.values-RU1606.values

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

plt.figure(figsize=[40,12])
plt.plot(spread)
plt.legend(['spread'])

mean=np.ones(spread.shape)*np.mean(spread)
data=pd.DataFrame([spread,mean],index=['spread','mean']).T
data[['spread','mean']].plot(figsize=[40,12])
```

Out[1]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc3a0073dd8>
```



In [ ]:

```
%timeit
import concurrent.futures
import multiprocessing
from rqalpha_plus import run_func
from rqalpha_plus.api import *
import numpy as np
import warnings

warnings.filterwarnings("ignore")
max_workers = multiprocessing.cpu_count() - 1
max_workers = 2
```

```
config = {
    'extra': {
        'log_level': 'error',
    },
    'base': {
        'securities': 'future',
        'matching_type': 'current_bar',
        'start_date': '2015-12-01',
        'end_date': '2016-04-01',
        'accounts': {
                 'future': 100000,
             },
        'frequency': '1m',
    },
    'mod': {
        'sys_progress': {
            'enabled': False,
            'show': True,
        },
        'sys_analyser': {
            'enabled': True,
        },
    },
}

# 需要调优的参数放在 tasks 中
tasks = []
for entry_score in np.arange(2, 3, 0.1):
    tasks.append((config, entry_score))


def run_bt(config, entry_score):

    #回测代码
    # 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。
    import numpy as np


    # 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
    def init(context):
        context.s1 = 'RU1605'
        context.s2 = 'RU1609'

        # 设置全局计数器
        context.counter = 0

        # 设置滚动窗口
        context.window = 60

        # 设置对冲手数,通过研究历史数据进行价格序列回归得到该值
        context.ratio = entry_score

        context.up_cross_up_limit = False
        context.down_cross_down_limit = False

        # 设置入场临界值
        context.entry_score = entry_score

        # 初始化时订阅合约行情。订阅之后的合约行情会在handle_bar中进行更新
        subscribe([context.s1, context.s2])
        print(entry_score)


    # before_trading此函数会在每天交易开始前被调用，当天只会被调用一次
    def before_trading(context):
        # 样例商品期货在回测区间内有夜盘交易,所以在每日开盘前将计数器清零
        context.counter = 0


    # 你选择的期货数据更新将会触发此段逻辑，例如日线或分钟线更新
    def handle_bar(context, bar_dict):

        # 获取当前一对合约的仓位情况。如尚未有仓位,则对应持仓量都为0
        position_a = context.portfolio.positions[context.s1]
        position_b = context.portfolio.positions[context.s2]

        context.counter += 1
        # 当累积满一定数量的bar数据时候,进行交易逻辑的判断
        if context.counter > context.window:

            # 获取当天历史分钟线价格队列
```

```
        price_array_a = history_bars(context.s1, context.window, '1m', 'close')
        price_array_b = history_bars(context.s2, context.window, '1m', 'close')

        # 计算价差序列、其标准差、均值、上限、下限
        spread_array = price_array_a - context.ratio * price_array_b
        std = np.std(spread_array)
        mean = np.mean(spread_array)
        up_limit = mean + context.entry_score * std
        down_limit = mean - context.entry_score * std

        # 获取当前bar对应合约的收盘价格并计算价差
        price_a = bar_dict[context.s1].close
        price_b = bar_dict[context.s2].close
        spread = price_a - context.ratio * price_b

        # 如果价差低于预先计算得到的下限,则为建仓信号,'买入'价差合约
        if spread <= down_limit and not context.down_cross_down_limit:
            # 可以通过logger打印日志
            # logger.info('spread: {}, mean: {}, down_limit: {}'.format(spread, mean, down_limit))
            # logger.info('创建买入价差中...')

            # 获取当前剩余的应建仓的数量
            qty_a = 1 - position_a.buy_quantity
            qty_b = context.ratio - position_b.sell_quantity

            # 由于存在成交不超过下一bar成交量25%的限制,所以可能要通过多次发单成交才能够成功建仓
            if qty_a > 0:
                buy_open(context.s1, qty_a)
            if qty_b > 0:
                sell_open(context.s2, qty_b)
            if qty_a == 0 and qty_b == 0:
                # 已成功建立价差的'多仓'
                context.down_cross_down_limit = True
                # logger.info('买入价差仓位创建成功!')

        # 如果价差向上回归移动平均线,则为平仓信号
        if spread >= mean and context.down_cross_down_limit:
            # logger.info('spread: {}, mean: {}, down_limit: {}'.format(spread, mean, down_limit))
            # logger.info('对买入价差仓位进行平仓操作中...')

            # 由于存在成交不超过下一bar成交量25%的限制,所以可能要通过多次发单成交才能够成功建仓
            qty_a = position_a.buy_quantity
            qty_b = position_b.sell_quantity
            if qty_a > 0:
                sell_close(context.s1, qty_a)
            if qty_b > 0:
                buy_close(context.s2, qty_b)
            if qty_a == 0 and qty_b == 0:
                context.down_cross_down_limit = False
                # logger.info('买入价差仓位平仓成功!')

        # 如果价差高于预先计算得到的上限,则为建仓信号,'卖出'价差合约
        if spread >= up_limit and not context.up_cross_up_limit:
            # logger.info('spread: {}, mean: {}, up_limit: {}'.format(spread, mean, up_limit))
            # logger.info('创建卖出价差中...')
            qty_a = 1 - position_a.sell_quantity
            qty_b = context.ratio - position_b.buy_quantity
            if qty_a > 0:
                sell_open(context.s1, qty_a)
            if qty_b > 0:
                buy_open(context.s2, qty_b)
            if qty_a == 0 and qty_b == 0:
                context.up_cross_up_limit = True
                # logger.info('卖出价差仓位创建成功')

        # 如果价差向下回归移动平均线,则为平仓信号
        if spread < mean and context.up_cross_up_limit:
            # logger.info('spread: {}, mean: {}, up_limit: {}'.format(spread, mean, up_limit))
            # logger.info('对卖出价差仓位进行平仓操作中...')
            qty_a = position_a.sell_quantity
            qty_b = position_b.buy_quantity
            if qty_a > 0:
                buy_close(context.s1, qty_a)
            if qty_b > 0:
                sell_close(context.s2, qty_b)
            if qty_a == 0 and qty_b == 0:
                context.up_cross_up_limit = False
                # logger.info('卖出价差仓位平仓成功!')



name = '{}'.format(entry_score)
```

```
    try:
        result = run_func(config=config, init=init, handle_bar=handle_bar)
    except Exception as e:
        print(e)
        return

    return result, name


# 并发运行回测
futures = []
with concurrent.futures.ProcessPoolExecutor(max_workers=max_workers) as executor:
    for task in tasks:
        future = executor.submit(run_bt, *task)

        futures.append(future)


#获取回测结果
results = []
for future in futures:

    result, name = future.result()
    results.append((result['sys_analyser'], name))


# 从回测结果中提取分析信息
data = []
for task in tasks:
    result, name = run_bt(*task)
    #print(result,name)
    result = result['sys_analyser']
    summary = result['summary']
    data.append({
        'name': name,
        'annualized_returns': summary['annualized_returns'],
        'sharpe': summary['sharpe'],
        'max_drawdown': summary['max_drawdown'],
    })


# 分析回测
results_df = pd.DataFrame(data)

print('')
print('-' * 50)
print('Sort by sharpe')
print(results_df.sort_values('sharpe', ascending=False)[:10])

print('-' * 50)
print('Sort by annualized_returns')
print(results_df.sort_values('annualized_returns', ascending=False)[:10])

from rqdatac import *
```

结语¶

以上就是本次 notebook 更新的新功能，大家快来试试吧！

多因子模型

数据预处理（上）之离群值处理、标准化

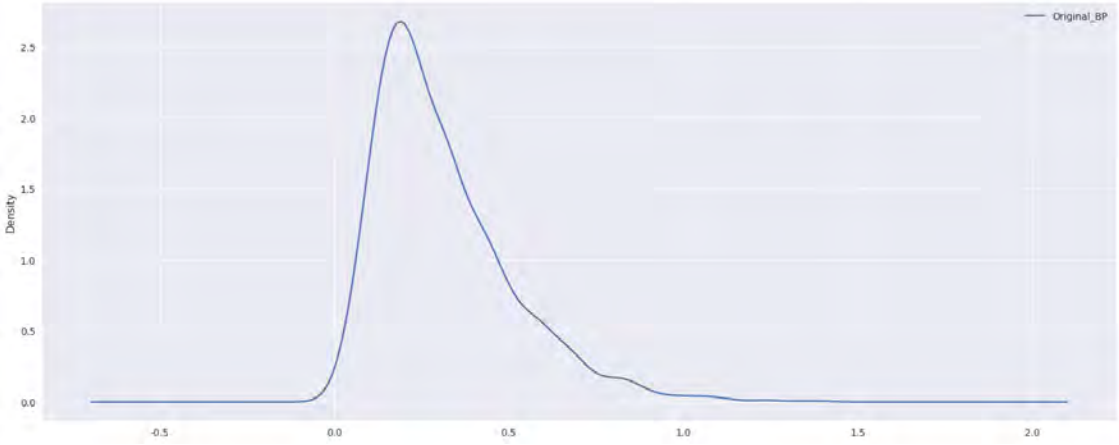
一般的数据预处理中常提及到三类处理：去极值、标准化、中性化。这几个词想必大家都不陌生，也许存在疑问或有自己的一番见解，本文将先对前两个进行解释和总结，欢迎讨论和指正~

一、离群值处理

因为过大或过小的数据可能会影响到分析结果，尤其是在做回归的时候，我们需要对那些离群值进行处理。实际上离群值和极值是有区别的，因为极值不代表异常，但实际处理中这两个所用方法差不多，所以这里也不强行区分了。

处理方法是调整因子值中的离群值至上下限（Winsorzation 处理），其中上下限由离群值判断的标准给出，从而减小离群值的影响力。离群值的判断标准有三种，分别为 MAD、 3σ、百分位法。

下图是2017-10-23日的全市场BP原始数据。



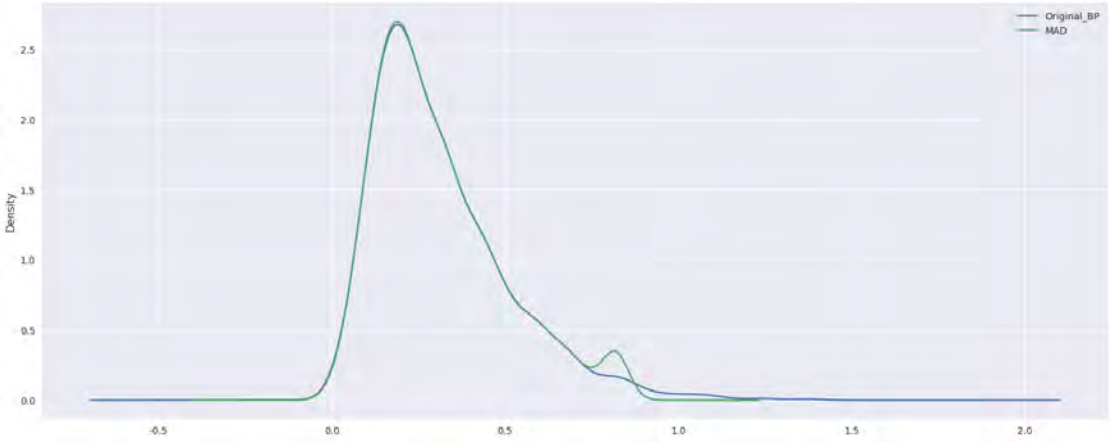
1. MAD法：

MAD又称为绝对值差中位数法（Median Absolute Deviation ）。MAD 是一种先需计算所有因子与平均值之间的距离总和来检测离群值的方法。

处理的逻辑：第一步，找出所有因子的中位数 Xmedian； 第二步，得到每个因子与中位数的绝对偏差值 Xi?Xmedian ； 第三步，得到绝对偏差值的中位数 MAD；最后，确定参数 n，从而确定合理的范围为 [Xmedian?nMAD,Xmedian+nMAD] ，并针对超出合理范围的因子值做如下的调整：

$$X_i' = \begin{cases} X_{median} + nMAD & \text{if } X_i > X_{median} + nMAD \\ X_{median} - nMAD & \text{if } X_i < X_{median} - nMAD \\ X_i & \text{if } X_{median} - nMAD < X_i < X_{median} + nMAD \end{cases}$$

对全市场BP原始数据进行MAD处理后的结果：



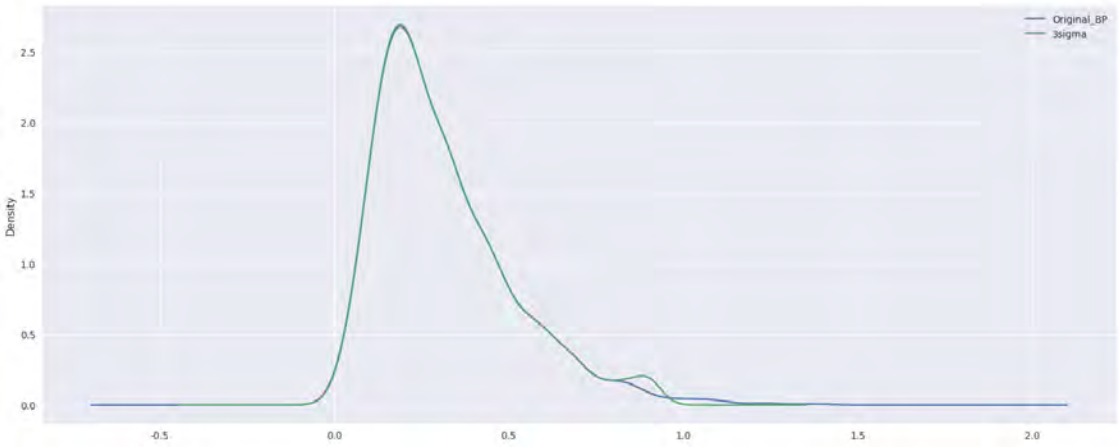
2. 3σ法

又称为标准差法。标准差本身可以体现因子的离散程度，是基于因子的平均值 Xmean而定的。在离群值处理过程中，可通过用 Xmean±nσ 来衡量因子与平均值的距离。

标准差法处理的逻辑与MAD法类似，首先计算出因子的平均值与标准差，其次确认参数 n（这里选定 n = 3），从而确认因子值的合理范围为 [Xmean?nσ,Xmean+nσ] ，并对因子值作如下的调整：

$$X_i' = \begin{cases} X_{mean} + n\sigma & \text{if } X_i > X_{mean} + n\sigma \\ X_{mean} - n\sigma & \text{if } X_i < X_{mean} - n\sigma \\ X_i & \text{if } X_{mean} - n\sigma < X_i < X_{mean} + n\sigma \end{cases}$$

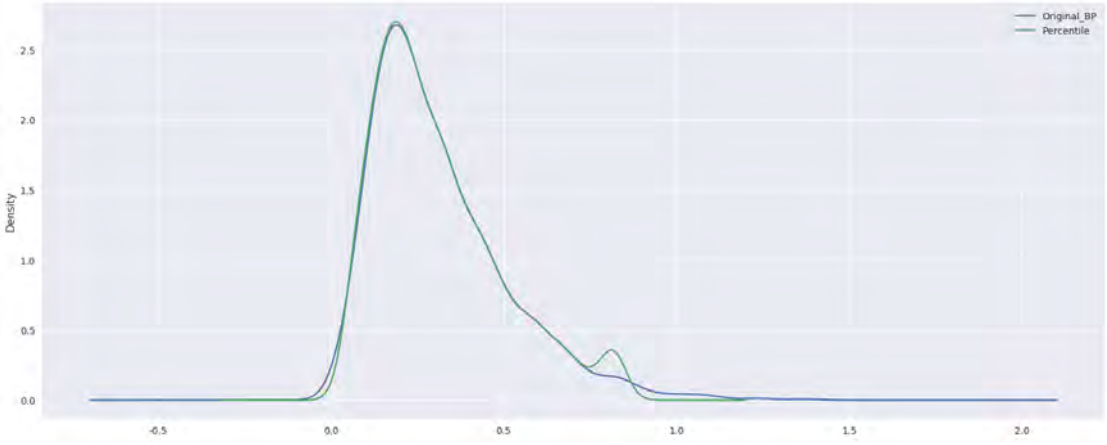
对全市场BP原始数据进行3σ法处理后的结果：



3. 百分位法：

计算的逻辑是将因子值进行升序的排序，对排位百分位高于97.5%或排位百分位低于2.5%的因子值，进行类似于 MAD 、 3σ 的方法进行调整。

对全市场BP原始数据进行百分位法处理后的结果：



通过这几个图表比对，我们可以发现百分位法的结果和MAD很相似。

二、标准化

标准化（standardization ）在统计学中有一系列含义，一般使用z-score 的方法。处理后的数据从有量纲转化为无量纲，从而使得数据更加集中，或者使得不同的指标能够进行比较和回归。

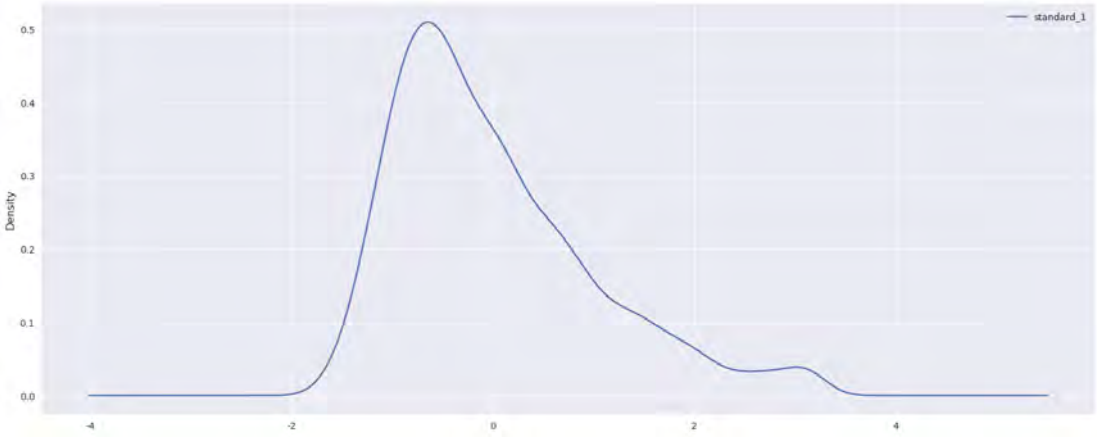
由此可见，标准化应该用于多个不同量级指标之间需要互相比较的时候。讲到这里，我们应该区分一下标准化和中性化。中性化的目的在于消除因子中的偏差和不需要的影响，详细的内容将会在下一个帖子总结~

对因子进行标准化处理的方法主要有以下两种：

1、对原始因子值进行标准化；

方法一可以保留更多的因子分布信息，但是需要去掉极端值，否则会影响到回归结果。回归的方法一般使用z-score ，将因子值的均值调整为0，标准差调整为1。 标准化处理基于原始数据的均值和标准差，处理的逻辑是因子值减去均值后，再除以标准差。

对已经过3σ法去极值后的结果进行标准化：



2、用因子的排序值进行标准化。

方法二只关注原始序列的相对排序关系，所以对原始变量的分布不做要求，属于非参数统计方法，可以适用于更多类型的数据。首先将原始数据的排序值作为参数，再将之带入方法一的标准化计算中。

由于转为排序值之后的分布图像意义不大，就不在此贴出。

在RQPro中的因子研究部分提供了三种离群值处理和方法一的标准化处理（当然我也不知道它们和我的代码是否完全一致...），大家可以试一试哦~

下一个帖子将讨论中性化~

数据预处理之去极值、标准化.ipynb

克隆研究 +3

In [1]:

```
#import seaborn
```

```
import numpy as np
```

```
import pandas as pd
```

```
import math
```

```
from statsmodels import regression
```

```
import statsmodels.api as sm
```

```
stocks = all_instruments(type="CS", date='2017-10-23').order_book_id.tolist()
```

```
data = get_fundamentals(query(fundamentals.eod_derivative_indicator.pb_ratio,fundamentals.eod_derivative_indicator.market_cap
```

```
).filter(fundamentals.income_statement.stockcode.in_(stocks)), '2017-10-23', '1d').major_xs('2017-10-23').dropna()
```

```
data['BP'] = 1/data['pb_ratio']
```

一、离群值处理

一、离群值处理

In [2]:

```
def filter_extreme_MAD(series,n): #MAD:中位数去极值
    median = np.percentile(series,50)
    new_median = np.percentile((series - median).abs(),50)
    max_range = median + n*new_median
    min_range = median - n*new_median
```

```
    return np.clip(series,min_range,max_range)
```

```
?
```

```
def filter_extreme_3sigma(series,n=3): #3 sigma
```

```
    mean = series.mean()
```

```
    std = series.std()
```

```
    max_range = mean + n*std
```

```
    min_range = mean - n*std
```

```
    return np.clip(series,min_range,max_range)
```

```
?
```

```
def filter_extreme_percentile(series,min = 0.025,max = 0.975): #百分位法
```

```
    series = series.sort_values()
```

```
    q = series.quantile([min,max])
```

```
    return np.clip(series,q.iloc[0],q.iloc[1])
```

```
可以看出MAD和百分位法的结果十分相近。
```

可以看出MAD和百分位法的结果十分相近。

In [3]:

```
fig = plt.figure(figsize = (20, 8))
```

```
ax = data['BP'].plot.kde(label = 'Original_BP')
```

```
ax = filter_extreme_percentile(data['BP']).plot.kde(label = 'Percentile')
```

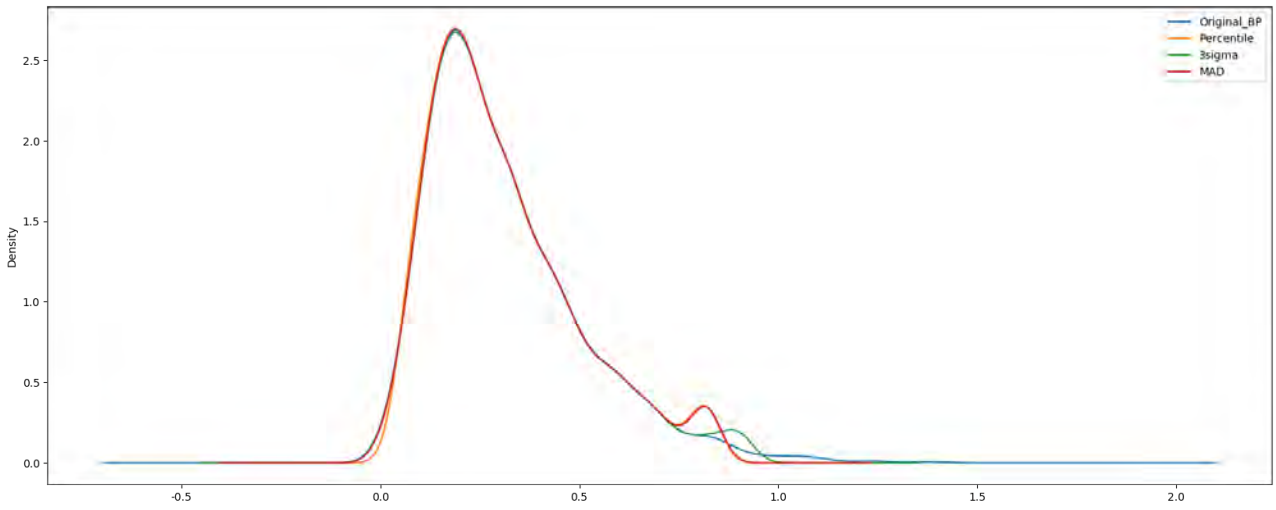
```
ax = filter_extreme_3sigma(data['BP']).plot.kde(label = '3sigma')
```

```
ax = filter_extreme_MAD(data['BP'],5).plot.kde(label = 'MAD')
```

```
ax.legend()
```

Out[3]:

```
<matplotlib.legend.Legend at 0x7fc9c3f58ac8>
```



二、标准化处理

二、标准化处理

In [4]:

```
def standardize_series(series): #原始值法
```

```
    std = series.std()
```

```
    mean = series.mean()
```

```
    return (series-mean)/std
```

```
fig = plt.figure(figsize = (20, 8))
```

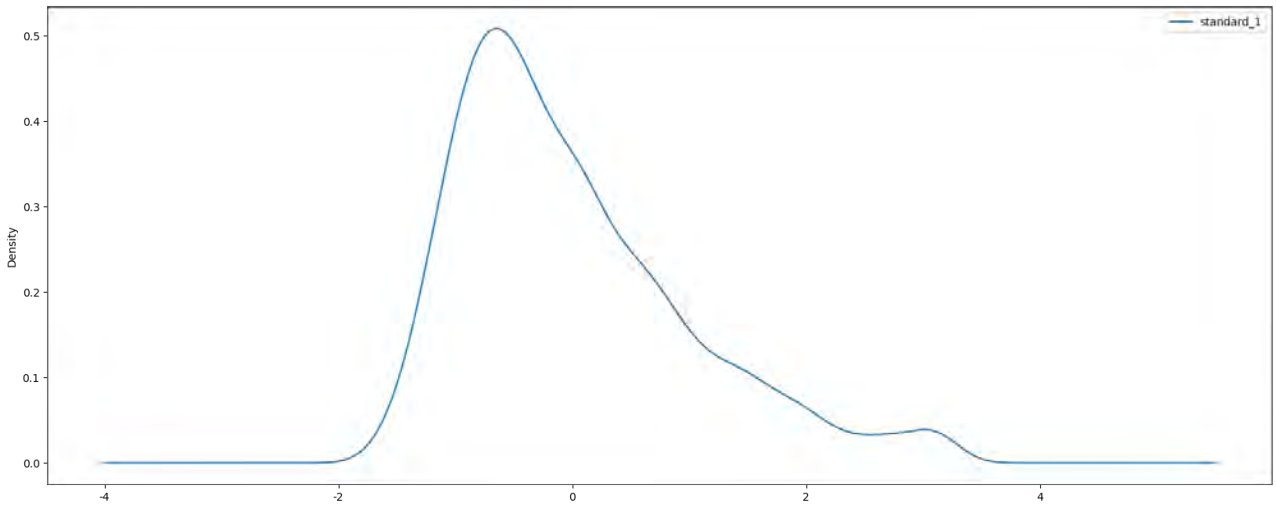
```
new = filter_extreme_3sigma(data['BP'])
```

```
ax = standardize_series(new).plot.kde(label = 'standard_1')
```

```
ax.legend()
```

Out[4]:

```
<matplotlib.legend.Legend at 0x7fc9c16d1da0>
```



In [5]:

```
standard_2 = standardize_series(new.rank())
```

```
standard_2
```

Out[5]:

```
000005.XSHE   -0.197560
600602.XSHG    0.376072
600651.XSHG    0.538256
600652.XSHG    0.042995
600654.XSHG    0.470770
600653.XSHG    0.527371
600601.XSHG    0.957323
000004.XSHE   -1.655042
000002.XSHE    0.773369
000001.XSHE    1.709467
000009.XSHE   -0.143136
600603.XSHG    1.467823
000012.XSHE    0.846298
600604.XSHG    0.373895
000016.XSHE   -0.419611
600605.XSHG   -0.096331
600606.XSHG    1.429726
000011.XSHE   -0.291170
000017.XSHE   -1.730148
000020.XSHE   -1.648511
000008.XSHE   -0.152932
000014.XSHE   -0.359744
000018.XSHE   -1.029163
600610.XSHG   -1.075968
600611.XSHG    1.554902
600612.XSHG   -0.080004
600613.XSHG    0.828882
600614.XSHG   -0.299878
600615.XSHG   -0.524106
600616.XSHG    0.585061
                 ...
603055.XSHG   -1.309992
002901.XSHE   -1.697493
```

```
603963.XSHG   -1.672458
002900.XSHE   -1.462380
603157.XSHG   -0.110481
300654.XSHE   -1.621299
603363.XSHG   -1.639804
300706.XSHE   -1.661573
603136.XSHG   -1.521159
603378.XSHG   -1.499389
601086.XSHG   -1.101003
603367.XSHG   -0.626423
300705.XSHE   -1.549459
300707.XSHE   -1.611503
603103.XSHG   -1.461292
002903.XSHE   -1.501566
603110.XSHG   -1.530955
002906.XSHE   -1.118419
603499.XSHG   -1.192436
300708.XSHE    0.410903
002905.XSHE   -1.350266
603829.XSHG   -1.237064
603396.XSHG   -0.954058
300710.XSHE   -0.757042
300709.XSHE   -1.607149
002907.XSHE   -0.656901
603466.XSHG   -0.688467
002908.XSHE   -0.173613
603683.XSHG   -0.003810
603922.XSHG   -0.836501
Name: BP, dtype: float64
```

In [1]:

```
#import seaborn
import numpy as np
import pandas as pd
import math
from statsmodels import regression
import statsmodels.api as sm
stocks = all_instruments(type="CS", date='2017-10-23').order_book_id.tolist()
data = get_fundamentals(query(fundamentals.eod_derivative_indicator.pb_ratio,fundamentals.eod_derivative_indicator.market_cap
).filter(fundamentals.income_statement.stockcode.in_(stocks)), '2017-10-23', '1d').major_xs('2017-10-23').dropna()
data['BP'] = 1/data['pb_ratio']
```

一、离群值处理

In [2]:

```
def filter_extreme_MAD(series,n): #MAD:中位数去极值
  median = np.percentile(series,50)
  new_median = np.percentile((series - median).abs(),50)
  max_range = median + n*new_median
  min_range = median - n*new_median
  return np.clip(series,min_range,max_range)

def filter_extreme_3sigma(series,n=3): #3 sigma
  mean = series.mean()
  std = series.std()
  max_range = mean + n*std
  min_range = mean - n*std
  return np.clip(series,min_range,max_range)

def filter_extreme_percentile(series,min = 0.025,max = 0.975): #百分位法
  series = series.sort_values()
  q = series.quantile([min,max])
  return np.clip(series,q.iloc[0],q.iloc[1])
```

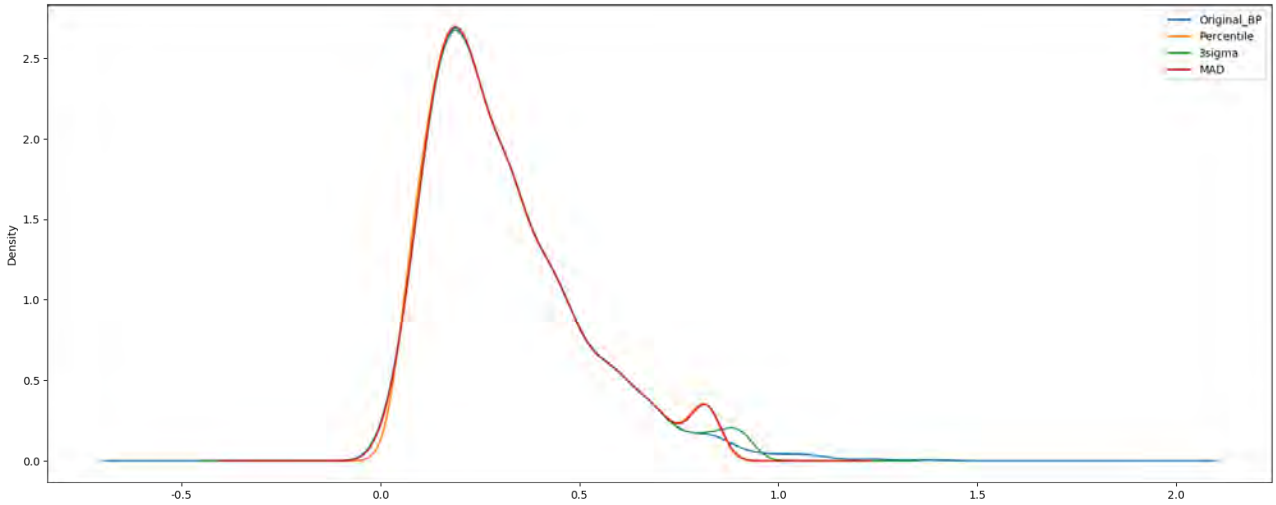
可以看出MAD和百分位法的结果十分相近。

In [3]:

```
fig = plt.figure(figsize = (20, 8))
ax = data['BP'].plot.kde(label = 'Original_BP')
ax = filter_extreme_percentile(data['BP']).plot.kde(label = 'Percentile')
ax = filter_extreme_3sigma(data['BP']).plot.kde(label = '3sigma')
ax = filter_extreme_MAD(data['BP'],5).plot.kde(label = 'MAD')
ax.legend()
```

Out[3]:

```
<matplotlib.legend.Legend at 0x7fc9c3f58ac8>
```



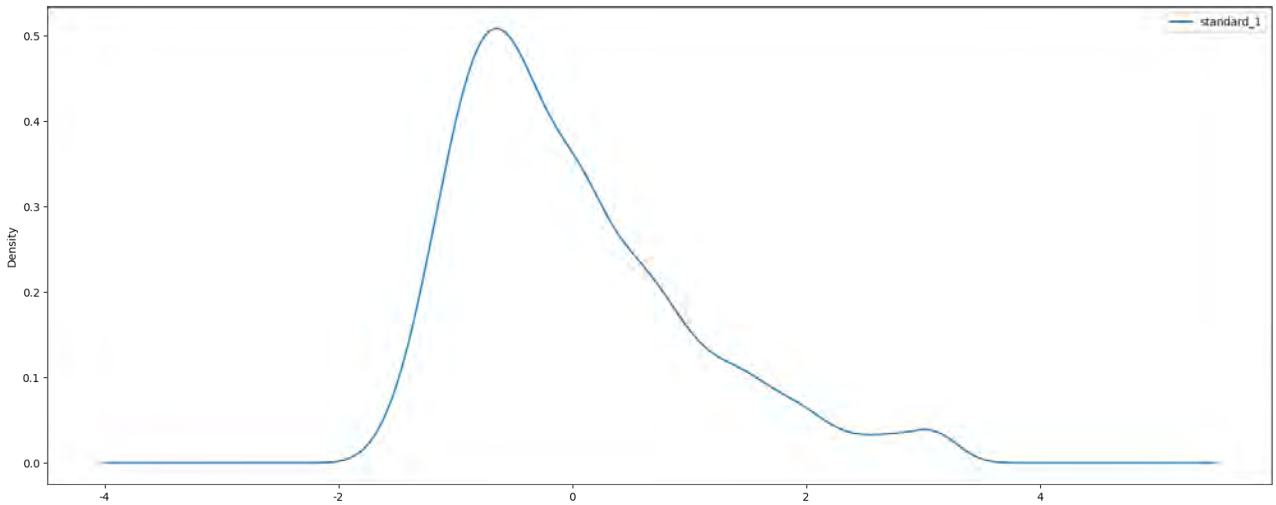
二、标准化处理

In [4]:

```
def standardize_series(series): #原始值法
  std = series.std()
  mean = series.mean()
  return (series-mean)/std
fig = plt.figure(figsize = (20, 8))
new = filter_extreme_3sigma(data['BP'])
ax = standardize_series(new).plot.kde(label = 'standard_1')
ax.legend()
```

Out[4]:

```
<matplotlib.legend.Legend at 0x7fc9c16d1da0>
```



In [5]:

```
standard_2 = standardize_series(new.rank())
standard_2
```

Out[5]:

```
000005.XSHE   -0.197560
600602.XSHG    0.376072
600651.XSHG    0.538256
600652.XSHG    0.042995
600654.XSHG    0.470770
600653.XSHG    0.527371
600601.XSHG    0.957323
000004.XSHE   -1.655042
000002.XSHE    0.773369
000001.XSHE    1.709467
000009.XSHE   -0.143136
600603.XSHG    1.467823
000012.XSHE    0.846298
600604.XSHG    0.373895
000016.XSHE   -0.419611
```

```
600605.XSHG   -0.096331
600606.XSHG    1.429726
000011.XSHE   -0.291170
000017.XSHE   -1.730148
000020.XSHE   -1.648511
000008.XSHE   -0.152932
000014.XSHE   -0.359744
000018.XSHE   -1.029163
600610.XSHG   -1.075968
600611.XSHG    1.554902
600612.XSHG   -0.080004
600613.XSHG    0.828882
600614.XSHG   -0.299878
600615.XSHG   -0.524106
600616.XSHG    0.585061
                 ...
603055.XSHG   -1.309992
002901.XSHE   -1.697493
603963.XSHG   -1.672458
002900.XSHE   -1.462380
603157.XSHG   -0.110481
300654.XSHE   -1.621299
603363.XSHG   -1.639804
300706.XSHE   -1.661573
603136.XSHG   -1.521159
603378.XSHG   -1.499389
601086.XSHG   -1.101003
603367.XSHG   -0.626423
300705.XSHE   -1.549459
300707.XSHE   -1.611503
603103.XSHG   -1.461292
002903.XSHE   -1.501566
603110.XSHG   -1.530955
002906.XSHE   -1.118419
603499.XSHG   -1.192436
300708.XSHE    0.410903
002905.XSHE   -1.350266
603829.XSHG   -1.237064
603396.XSHG   -0.954058
300710.XSHE   -0.757042
300709.XSHE   -1.607149
002907.XSHE   -0.656901
603466.XSHG   -0.688467
002908.XSHE   -0.173613
603683.XSHG   -0.003810
603922.XSHG   -0.836501
Name: BP, dtype: float64
```

Fama-Franch 三因子及其拓展五因子模型¶

In [2]:

```
import pandas as pd
import numpy as np
import datetime
import statsmodels.api as sm

import plotly.plotly as py
import plotly.graph_objs as go
import plotly.tools as tls
from plotly.graph_objs import *
import plotly.graph_objs as go
import cufflinks as cf

py.sign_in('DemoAccount','2qdyfjyr7o')
```

第一节：Fama-French 三因子模型¶

 Fama和French提出了一个和sharpe风格分析在理念伤非常接近的业绩分析方法，他们提出了用如下回归方程进行业绩分析。

$$r_p(t) = \alpha_p + \beta \cdot r_b(t) + \beta \cdot SMB(t) + \beta_v \cdot HML(t) + \epsilon_p(t)$$

这看起来像是增加了额外两项的标准CAPM回归。SMB(t)是一个买入小盘股卖出大盘股的组合收益率。HML(t)是一个买入高净市率股票卖空低净市率股票的投资组合收益率。Fema和Franch的方法在两个维度进行了控制，同时使用了回归的方法。

In [3]:

```
# 拿到基准收益率
index_change_rate = get_price_change_rate('000001.XSHG',start_date='20160514', end_date='20170515')
# 拿到无风险利率
```

```
risk_free_rate = get_yield_curve(start_date='20160514', end_date='20170515', tenor=None, country='cn')['0S']
# 反年化处理
risk_free_rate = (risk_free_rate+1)**(1/244)-1
# 取与基准相同的交易日
risk_free_rate = risk_free_rate[list(index_change_rate.index)]
price_change_rate = pd.DataFrame()
price_change_rate['risk_free_rate'] = risk_free_rate
price_change_rate['000001.XSHG'] = index_change_rate
price_change_rate
```

Out[3]:

	risk_free_rate	000001.XSHG
2016-05-16	0.000070	0.008402
2016-05-17	0.000070	-0.002518
2016-05-18	0.000070	-0.012719
2016-05-19	0.000070	-0.000217
2016-05-20	0.000070	0.006618
2016-05-23	0.000072	0.006428
2016-05-24	0.000072	-0.007729
2016-05-25	0.000074	-0.002332
2016-05-26	0.000076	0.002613
2016-05-27	0.000076	-0.000495
2016-05-30	0.000076	0.000498
2016-05-31	0.000074	0.033363
2016-06-01	0.000075	-0.001066
2016-06-02	0.000075	0.004023
2016-06-03	0.000075	0.004599
2016-06-06	0.000073	-0.001560
2016-06-07	0.000073	0.000664
2016-06-08	0.000073	-0.003026
2016-06-13	0.000073	-0.032143
2016-06-14	0.000071	0.003218
2016-06-15	0.000069	0.015840
2016-06-16	0.000069	-0.004985
2016-06-17	0.000069	0.004277
2016-06-20	0.000067	0.001284
2016-06-21	0.000068	-0.003549
2016-06-22	0.000068	0.009377
2016-06-23	0.000068	-0.004677
2016-06-24	0.000068	-0.013027
2016-06-27	0.000068	0.014510
2016-06-28	0.000072	0.005820
...	...	...
2017-03-30	0.000116	-0.009588
2017-03-31	0.000115	0.003824
2017-04-05	0.000112	0.014830
2017-04-06	0.000112	0.003272
2017-04-07	0.000110	0.001710
2017-04-10	0.000108	-0.005240
2017-04-11	0.000106	0.005987

	risk_free_rate	000001.XSHG
2017-04-12	0.000107	-0.004602
2017-04-13	0.000100	0.000651
2017-04-14	0.000096	-0.009125
2017-04-17	0.000094	-0.007363
2017-04-18	0.000094	-0.007900
2017-04-19	0.000096	-0.008142
2017-04-20	0.000104	0.000446
2017-04-21	0.000105	0.000331
2017-04-24	0.000102	-0.013747
2017-04-25	0.000101	0.001609
2017-04-26	0.000101	0.002003
2017-04-27	0.000103	0.003610
2017-04-28	0.000106	0.000784
2017-05-02	0.000107	-0.003470
2017-05-03	0.000108	-0.002661
2017-05-04	0.000108	-0.002544
2017-05-05	0.000108	-0.007780
2017-05-08	0.000109	-0.007871
2017-05-09	0.000108	0.000622
2017-05-10	0.000108	-0.009006
2017-05-11	0.000106	0.002855
2017-05-12	0.000102	0.007190
2017-05-15	0.000099	0.002178

243 rows × 2 columns

In [4]:

```
# 取2016-05-15的估值数据作为SMB和HML的计算标准
all_stocks_value = get_fundamentals(
    query(
        fundamentals.eod_derivative_indicator.pb_ratio,
        fundamentals.eod_derivative_indicator.market_cap)
    , '2016-05-15', interval = '1d' ,report_quarter = False)
all_stocks_value = all_stocks_value.major_xs(all_stocks_value.major_axis[0])
all_stocks_value['bp_ratio'] = 1/all_stocks_value['pb_ratio']
del all_stocks_value['pb_ratio']
all_stocks_value = all_stocks_value.dropna(how = 'any')
all_stocks_value
```

Out[4]:

	market_cap	bp_ratio
000005.XSHE	7.47327e+09	0.152798
600602.XSHG	1.26845e+10	0.262
600651.XSHG	1.02955e+10	0.317914
600652.XSHG	1.14247e+10	0.182093
600654.XSHG	2.46982e+10	0.119852
600653.XSHG	7.64927e+09	0.291265
600601.XSHG	9.0649e+09	0.395977
000004.XSHE	3.09286e+09	0.0255849
000002.XSHE	2.69686e+11	0.373162
000001.XSHE	1.48095e+11	1.26662

	market_cap	bp_ratio
000009.XSHE	1.95511e+10	0.228493
000012.XSHE	2.21438e+10	0.364804
600608.XSHG	2.9762e+09	0.00107698
600604.XSHG	1.79126e+10	0.162396
000016.XSHE	1.10043e+10	0.256575
600605.XSHG	2.66546e+09	0.209617
600606.XSHG	1.42246e+11	0.384793
000011.XSHE	7.22327e+09	0.290833
000017.XSHE	5.53553e+09	0.00215181
000007.XSHE	5.49467e+09	0.0686625
000006.XSHE	9.77396e+09	0.45436
000020.XSHE	5.66322e+09	0.0556839
000008.XSHE	3.21549e+10	0.174077
000014.XSHE	3.55001e+09	0.200064
000018.XSHE	1.95209e+10	0.0697408
600609.XSHG	4.8733e+09	0.0540941
600610.XSHG	1.45265e+10	0.0793393
600611.XSHG	1.7526e+10	0.506791
600612.XSHG	2.08986e+10	0.227713
600613.XSHG	7.51973e+09	0.297832
...	...	...
300503.XSHE	3.904e+09	0.160994
603520.XSHG	4.9488e+09	0.160671
603919.XSHG	7.8176e+09	0.193431
002789.XSHE	4.22971e+09	0.242442
300505.XSHE	3.73627e+09	0.152061
601020.XSHG	1.16376e+10	0.120376
300484.XSHE	8.79268e+09	0.0563768
603861.XSHG	9.65885e+09	0.182655
300506.XSHE	4.482e+09	0.114185
002792.XSHE	1.035e+10	0.167003
002791.XSHE	9.24963e+09	0.243582
300474.XSHE	1.35543e+10	0.0598795
603028.XSHG	5.04749e+09	0.129406
603798.XSHG	4.838e+09	0.143127
603701.XSHG	4.1552e+09	0.0704265
002793.XSHE	4.499e+09	0.0789129
603868.XSHG	2.09829e+10	0.0595444
300508.XSHE	5.25528e+09	0.0384578
603726.XSHG	3.75091e+09	0.121014
300509.XSHE	4.1096e+09	0.080525
603029.XSHG	3.42651e+09	0.148427
603822.XSHG	3.2208e+09	0.128284
002795.XSHE	4.566e+09	0.0496047
300507.XSHE	5.38027e+09	0.0665531

	market_cap	bp_ratio
603528.XSHG	6.02886e+09	0.120557
300511.XSHE	7.0785e+09	0.0970676
300510.XSHE	2.47899e+09	0.124228
603101.XSHG	4.908e+09	0.133595
002796.XSHE	1.9864e+09	0.133518
002797.XSHE	4.05841e+10	0.153226

2835 rows × 2 columns

In [5]:

```
# 拿到所有股票的日收益率数据
all_price_change_rate = get_price_change_rate(list(all_stocks_value.index),start_date='20160514', end_date='20170515')
all_price_change_rate
```

Out[5]:

order_book_id	000001.XSHE	000002.XSHE	000004.XSHE	000005.XSHE	000006.XSHE	000007.XSHE	000008.X
date							
2016-05-16	0.000966	0.000000	0.000000	-0.004250	0.015193	0.0	0.002573
2016-05-17	-0.005792	0.000000	0.000000	-0.004267	0.004082	0.0	-0.023952
2016-05-18	0.000000	0.000000	0.000000	-0.044286	-0.025745	0.0	-0.041192
2016-05-19	-0.004854	0.000000	0.000000	0.004485	0.001391	0.0	0.003656
2016-05-20	0.004878	0.000000	0.000000	0.013393	0.002778	0.0	-0.002732
2016-05-23	-0.001942	0.000000	0.000000	0.011747	0.015235	0.0	0.050228
2016-05-24	-0.006809	0.000000	0.000000	-0.014514	-0.005457	0.0	0.009565
2016-05-25	0.001959	0.000000	0.000000	-0.002945	-0.009602	0.0	-0.022395
2016-05-26	-0.000978	0.000000	0.000000	0.000000	-0.002770	0.0	0.004405
2016-05-27	0.004892	0.000000	0.000000	0.004431	0.019444	0.0	0.001754
2016-05-30	0.000974	0.000000	0.000000	-0.011764	-0.027248	0.0	-0.009632
2016-05-31	0.026265	0.000000	0.000000	0.034226	0.033614	0.0	0.022104
2016-06-01	-0.006635	0.000000	0.000000	0.011510	0.001355	0.0	0.000000
2016-06-02	-0.001908	0.000000	0.000000	0.012803	0.017591	0.0	-0.000865
2016-06-03	0.003824	0.000000	0.000000	0.000000	-0.001330	0.0	-0.003463
2016-06-06	0.000952	0.000000	0.000000	-0.009832	-0.001332	0.0	0.007819
2016-06-07	0.000951	0.000000	0.000000	0.019858	-0.013333	0.0	0.047414
2016-06-08	-0.001901	0.000000	0.000000	-0.018080	-0.002703	0.0	0.001646
2016-06-13	-0.016190	0.000000	0.000000	-0.048158	-0.044715	0.0	-0.073131
2016-06-14	0.006776	0.000000	0.000000	0.099701	0.001418	0.0	0.028369
2016-06-15	0.003846	0.000000	0.000000	0.082544	0.018414	0.0	0.045690
2016-06-16	-0.000292	0.000000	0.000000	-0.022500	-0.009736	0.0	-0.014015
2016-06-17	0.001167	0.000000	0.000000	-0.007673	0.032856	0.0	0.000836
2016-06-20	0.002331	0.000000	0.000000	0.011598	-0.013831	0.0	-0.047619
2016-06-21	0.001163	0.000000	0.000000	-0.045860	-0.001403	0.0	-0.082456
2016-06-22	0.013937	0.000000	0.000000	0.014687	0.019663	0.0	-0.001912
2016-06-23	-0.008018	0.000000	0.000000	-0.003947	-0.004132	0.0	-0.012452
2016-06-24	-0.010393	0.000000	0.000000	0.007925	0.015214	0.0	-0.023278
2016-06-27	0.004667	0.000000	0.000000	0.009175	0.008175	0.0	0.014896
2016-06-28	0.002323	0.000000	0.000000	0.002598	0.009459	0.0	0.009785

order_book_id	000001.XSHE	000002.XSHE	000004.XSHE	000005.XSHE	000006.XSHE	000007.XSHE	000008.X
date							
...	...	...	...	...	...	...	...
2017-03-30	-0.003293	-0.022846	-0.025297	-0.025039	-0.035377	0.0	0.018868
2017-03-31	0.009912	0.002435	-0.026764	0.004816	0.012225	0.0	0.011574
2017-04-05	0.004362	0.000000	0.000000	0.015974	0.020532	0.0	0.017163
2017-04-06	-0.001086	0.010204	0.000000	0.001573	0.009467	0.0	0.033746
2017-04-07	0.000000	0.000481	0.000000	0.004709	0.001172	0.0	-0.011969
2017-04-10	-0.002174	-0.009615	0.000000	-0.015625	0.001171	0.0	-0.020925
2017-04-11	-0.003268	0.004854	0.000000	0.044444	0.070176	0.0	0.017998
2017-04-12	-0.003279	0.015459	0.000000	0.021278	0.056831	0.0	-0.018784
2017-04-13	0.000000	-0.003806	0.000000	-0.011906	-0.032058	0.0	-0.007883
2017-04-14	-0.004386	-0.014327	0.000000	-0.028614	-0.043803	0.0	0.001135
2017-04-17	0.002203	-0.003876	0.000000	-0.013953	-0.022347	0.0	0.006803
2017-04-18	-0.005494	-0.004377	0.000000	-0.007862	-0.005714	0.0	-0.010135
2017-04-19	-0.015470	0.002931	0.000000	-0.019018	-0.016092	0.0	-0.010239
2017-04-20	0.001122	0.004871	0.000000	-0.004846	-0.016355	0.0	0.000000
2017-04-21	0.005605	0.002424	0.000000	0.004869	-0.002375	0.0	0.005747
2017-04-24	-0.004459	-0.012089	0.000000	-0.029079	-0.036905	0.0	-0.012571
2017-04-25	0.007839	0.005874	0.000000	-0.009984	0.004945	0.0	-0.002315
2017-04-26	-0.001111	-0.009246	0.000000	0.000000	0.011070	0.0	-0.022042
2017-04-27	-0.002225	-0.029470	0.000000	-0.025210	-0.002433	0.0	-0.024911
2017-04-28	0.002230	-0.013158	0.000000	0.003449	0.014634	0.0	-0.017032
2017-05-02	-0.005562	-0.015385	0.000000	-0.017182	-0.006010	0.0	-0.012377
2017-05-03	-0.003356	-0.017708	0.000000	0.010488	0.014510	0.0	-0.007519
2017-05-04	-0.019080	0.008484	0.000000	0.000000	0.004768	0.0	-0.005050
2017-05-05	-0.012586	-0.006835	0.000000	-0.025952	-0.033215	0.0	-0.043147
2017-05-08	-0.006953	-0.022234	0.000000	-0.028418	-0.040491	0.0	-0.030504
2017-05-09	0.008168	0.008121	0.000000	0.001827	0.011509	0.0	0.002736
2017-05-10	0.003472	0.002685	0.000000	-0.038320	-0.020228	0.0	0.004093
2017-05-11	0.003460	0.030530	0.000000	-0.022771	0.020645	0.0	0.008152
2017-05-12	0.022989	0.008316	0.000000	0.013593	0.011378	0.0	-0.009434
2017-05-15	-0.004494	-0.003093	0.000000	-0.013411	-0.017500	0.0	0.001360

243 rows × 2835 columns

In [6]:

```
# 去除nan
test = all_price_change_rate.T.dropna(how = 'any').T
test
```

Out[6]:

order_book_id	000001.XSHE	000002.XSHE	000004.XSHE	000005.XSHE	000006.XSHE	000007.XSHE	000008.X
date							
2016-05-16	0.000966	0.000000	0.000000	-0.004250	0.015193	0.0	0.002573
2016-05-17	-0.005792	0.000000	0.000000	-0.004267	0.004082	0.0	-0.023952
2016-05-18	0.000000	0.000000	0.000000	-0.044286	-0.025745	0.0	-0.041192
2016-05-19	-0.004854	0.000000	0.000000	0.004485	0.001391	0.0	0.003656

order_book_id	000001.XSHE	000002.XSHE	000004.XSHE	000005.XSHE	000006.XSHE	000007.XSHE	000008.X
date							
2016-05-20	0.004878	0.000000	0.000000	0.013393	0.002778	0.0	-0.002732
2016-05-23	-0.001942	0.000000	0.000000	0.011747	0.015235	0.0	0.050228
2016-05-24	-0.006809	0.000000	0.000000	-0.014514	-0.005457	0.0	0.009565
2016-05-25	0.001959	0.000000	0.000000	-0.002945	-0.009602	0.0	-0.022395
2016-05-26	-0.000978	0.000000	0.000000	0.000000	-0.002770	0.0	0.004405
2016-05-27	0.004892	0.000000	0.000000	0.004431	0.019444	0.0	0.001754
2016-05-30	0.000974	0.000000	0.000000	-0.011764	-0.027248	0.0	-0.009632
2016-05-31	0.026265	0.000000	0.000000	0.034226	0.033614	0.0	0.022104
2016-06-01	-0.006635	0.000000	0.000000	0.011510	0.001355	0.0	0.000000
2016-06-02	-0.001908	0.000000	0.000000	0.012803	0.017591	0.0	-0.000865
2016-06-03	0.003824	0.000000	0.000000	0.000000	-0.001330	0.0	-0.003463
2016-06-06	0.000952	0.000000	0.000000	-0.009832	-0.001332	0.0	0.007819
2016-06-07	0.000951	0.000000	0.000000	0.019858	-0.013333	0.0	0.047414
2016-06-08	-0.001901	0.000000	0.000000	-0.018080	-0.002703	0.0	0.001646
2016-06-13	-0.016190	0.000000	0.000000	-0.048158	-0.044715	0.0	-0.073131
2016-06-14	0.006776	0.000000	0.000000	0.099701	0.001418	0.0	0.028369
2016-06-15	0.003846	0.000000	0.000000	0.082544	0.018414	0.0	0.045690
2016-06-16	-0.000292	0.000000	0.000000	-0.022500	-0.009736	0.0	-0.014015
2016-06-17	0.001167	0.000000	0.000000	-0.007673	0.032856	0.0	0.000836
2016-06-20	0.002331	0.000000	0.000000	0.011598	-0.013831	0.0	-0.047619
2016-06-21	0.001163	0.000000	0.000000	-0.045860	-0.001403	0.0	-0.082456
2016-06-22	0.013937	0.000000	0.000000	0.014687	0.019663	0.0	-0.001912
2016-06-23	-0.008018	0.000000	0.000000	-0.003947	-0.004132	0.0	-0.012452
2016-06-24	-0.010393	0.000000	0.000000	0.007925	0.015214	0.0	-0.023278
2016-06-27	0.004667	0.000000	0.000000	0.009175	0.008175	0.0	0.014896
2016-06-28	0.002323	0.000000	0.000000	0.002598	0.009459	0.0	0.009785
...	...	...	...	...	...	...	...
2017-03-30	-0.003293	-0.022846	-0.025297	-0.025039	-0.035377	0.0	0.018868
2017-03-31	0.009912	0.002435	-0.026764	0.004816	0.012225	0.0	0.011574
2017-04-05	0.004362	0.000000	0.000000	0.015974	0.020532	0.0	0.017163
2017-04-06	-0.001086	0.010204	0.000000	0.001573	0.009467	0.0	0.033746
2017-04-07	0.000000	0.000481	0.000000	0.004709	0.001172	0.0	-0.011969
2017-04-10	-0.002174	-0.009615	0.000000	-0.015625	0.001171	0.0	-0.020925
2017-04-11	-0.003268	0.004854	0.000000	0.044444	0.070176	0.0	0.017998
2017-04-12	-0.003279	0.015459	0.000000	0.021278	0.056831	0.0	-0.018784
2017-04-13	0.000000	-0.003806	0.000000	-0.011906	-0.032058	0.0	-0.007883
2017-04-14	-0.004386	-0.014327	0.000000	-0.028614	-0.043803	0.0	0.001135
2017-04-17	0.002203	-0.003876	0.000000	-0.013953	-0.022347	0.0	0.006803
2017-04-18	-0.005494	-0.004377	0.000000	-0.007862	-0.005714	0.0	-0.010135
2017-04-19	-0.015470	0.002931	0.000000	-0.019018	-0.016092	0.0	-0.010239
2017-04-20	0.001122	0.004871	0.000000	-0.004846	-0.016355	0.0	0.000000
2017-04-21	0.005605	0.002424	0.000000	0.004869	-0.002375	0.0	0.005747
2017-04-24	-0.004459	-0.012089	0.000000	-0.029079	-0.036905	0.0	-0.012571
2017-04-25	0.007839	0.005874	0.000000	-0.009984	0.004945	0.0	-0.002315

order_book_id	000001.XSHE	000002.XSHE	000004.XSHE	000005.XSHE	000006.XSHE	000007.XSHE	000008.X
date							
2017-04-26	-0.001111	-0.009246	0.000000	0.000000	0.011070	0.0	-0.022042
2017-04-27	-0.002225	-0.029470	0.000000	-0.025210	-0.002433	0.0	-0.024911
2017-04-28	0.002230	-0.013158	0.000000	0.003449	0.014634	0.0	-0.017032
2017-05-02	-0.005562	-0.015385	0.000000	-0.017182	-0.006010	0.0	-0.012377
2017-05-03	-0.003356	-0.017708	0.000000	0.010488	0.014510	0.0	-0.007519
2017-05-04	-0.019080	0.008484	0.000000	0.000000	0.004768	0.0	-0.005050
2017-05-05	-0.012586	-0.006835	0.000000	-0.025952	-0.033215	0.0	-0.043147
2017-05-08	-0.006953	-0.022234	0.000000	-0.028418	-0.040491	0.0	-0.030504
2017-05-09	0.008168	0.008121	0.000000	0.001827	0.011509	0.0	0.002736
2017-05-10	0.003472	0.002685	0.000000	-0.038320	-0.020228	0.0	0.004093
2017-05-11	0.003460	0.030530	0.000000	-0.022771	0.020645	0.0	0.008152
2017-05-12	0.022989	0.008316	0.000000	0.013593	0.011378	0.0	-0.009434
2017-05-15	-0.004494	-0.003093	0.000000	-0.013411	-0.017500	0.0	0.001360

243 rows × 2833 columns

In [7]:

```
stock_need_drop = list(set(all_price_change_rate.columns).difference(set(test.columns)))
stock_need_drop
```

Out[7]:

```
['600005.XSHG', '000748.XSHE']
```

 有两支股票的日收益率数据含有nan，'000748.XSHE', '600005.XSHG'。查询资料发现：
 电脑(000066)和长城信息(000748)3月9日发布重大资产重组公告,长城电脑拟换股吸收合并长城信息,交易完成后,长城电脑将作为存续方,长城信息将注销法人资格,也将退市。
 武钢股份(600005)此前发布公告称,经上交所上市委员会审核,决定对武钢股份予以终止上市,终止上市日期为2月14日。
 两支含有nan的股票都退市，故没有数据。

In [8]:

```
# 在估值数据中将上述两支股票删除
all_stocks_value = all_stocks_value.T
for stock in stock_need_drop:
    del all_stocks_value[stock]
all_stocks_value = all_stocks_value.T
```

 以市值中位数为界将所有股票分为两类----S类（small，小盘股）和B类（big，大盘类）。同时对所有股票按照净市率排序。最后30%定义为L类，中间40%定义为M类，最高30%定义为H类。两个维度共计6类，计算6类的加权收益率。

In [9]:

```
copy = all_stocks_value.iloc[:,:]
all_stocks_value = all_stocks_value.sort_values('market_cap') #市值升序排布
S = all_stocks_value.index[:int(len(all_stocks_value.index)/2)]
B = all_stocks_value.index[int(len(all_stocks_value.index)/2):]
all_stocks_value = all_stocks_value.sort_values('bp_ratio') #净市率升序排列
L = all_stocks_value.index[:int(len(all_stocks_value.index)*0.3)]
M = all_stocks_value.index[int(len(all_stocks_value.index)*0.3):int(len(all_stocks_value.index)*0.7)]
H = all_stocks_value.index[int(len(all_stocks_value.index)*0.7):]
```

In [10]:

```
portfolio = {}
portfolio['SL'] = list(set(S).intersection(set(L)))
portfolio['SM'] = list(set(S).intersection(set(M)))
portfolio['SH'] = list(set(S).intersection(set(H)))
portfolio['BL'] = list(set(B).intersection(set(L)))
portfolio['BM'] = list(set(B).intersection(set(M)))
portfolio['BH'] = list(set(B).intersection(set(H)))
```

In [11]:

```
# 市值加权求出每个类别的日收益率时间序列
portfolio_return = {}
for por in portfolio:
    portfolio_return[por] = 0
    for stock in portfolio[por]:
        portfolio_return[por] += all_stocks_value['market_cap'][stock]*test[stock]
    portfolio_return[por] = portfolio_return[por]/np.sum(all_stocks_value['market_cap'][portfolio[por]])
```

In [12]:

```
SMB = (portfolio_return['SL']+portfolio_return['SM']+portfolio_return['SH'])/3-(portfolio_return['BL']+portfolio_return['BM']+portfol
HML = (portfolio_return['SH']+portfolio_return['BH'])/2-(portfolio_return['SL']+portfolio_return['BL'])/2
```

In [13]:

```
Rb = price_change_rate['000001.XSHG']-price_change_rate['risk_free_rate']
```

<font size = 3 face = '宋体'>投资组合介绍;
在2016-05-12平均买入去除停牌的股票中市值最小的20支股票。

In [14]:

```
# 构建组合
q = query(
            fundamentals.eod_derivative_indicator.market_cap,
        ).order_by(
            fundamentals.eod_derivative_indicator.market_cap.asc()
        ).limit(21)
pa = get_fundamentals(q,entry_date = '20160512')
stocks = list(pa.major_xs(pa.major_axis[0]).index)
stocks.remove('600656.XSHG')
# 我们拥有200万资金，计算买入时各股的量，分摊到各股每股10万
price = np.array(get_price(stocks,start_date='20160512', end_date='20170515',fields = 'close'))
# 最小买入一手,计算买入的量
position = np.array(1000/price[0],dtype=int)*100
# 计算剩余现金
cash = 2e6-(price[0]*position).sum()

portfolio_value = (price*position).sum(axis = 1)+cash
# np.diff(portfolio_value)后一天组合总价值减去前一天，得到新的序列.然后用差值依次除以前一天的值，求得日收益率序列
portfolio_value_rate = (np.diff(portfolio_value)/(portfolio_value[:-1]))[1:]
portfolio_value_rate
```

Out[14]:

```
array([ 0.04690176,  0.00826282, -0.02241415,  0.03122727,  0.03896055,
        0.0363623 ,  0.0057301 ,  0.00595328,  0.01073387, -0.00453721,
        0.00028715,  0.04368306,  0.00901098,  0.01761658,  0.00756529,
        0.0061413 ,  0.00794898, -0.00466621, -0.06015439,  0.00636494,
        0.03680521,  0.00244591,  0.01676524,  0.00397737, -0.00014629,
        0.00964465, -0.00050164, -0.00938068,  0.02654241,  0.00940173,
        0.02530033, -0.00579989,  0.01571037,  0.03436924,  0.01746873,
       -0.0019608 ,  0.00388429, -0.01320044, -0.03841505,  0.01614038,
        0.00296689,  0.00659967,  0.00394035, -0.01939202,  0.00254277,
        0.00196447, -0.01378129, -0.00854916, -0.00800506,  0.01581144,
       -0.058468  ,  0.00561951, -0.0057696 , -0.03025341,  0.02275274,
        0.01823169,  0.00014883, -0.00404677,  0.00897074,  0.01700534,
       -0.00093012, -0.01713266,  0.01166762,  0.02811279,  0.00815158,
        0.02077147,  0.01349679, -0.00213392, -0.02615654,  0.01625438,
        0.00089884, -0.00257379, -0.00128746,  0.00063466, -0.00386035,
        0.00785068, -0.00887345, -0.00395449,  0.02027403,  0.01631168,
       -0.00641006,  0.0040704 , -0.00293442, -0.03838325,  0.01949461,
        0.01179365,  0.02356316,  0.00249913,  0.00050825,  0.00449336,
        0.00029063, -0.03076373, -0.00023486, -0.00102183,  0.00876477,
        0.00547108,  0.01840432,  0.01025487,  0.01705659,  0.01044656,
       -0.00994089, -0.01672001,  0.01595148, -0.00253827, -0.00142301,
       -0.00262072,  0.00310977,  0.00377454, -0.00666643,  0.0044867 ,
       -0.00980946,  0.00582684,  0.01871992, -0.00297613,  0.00696657,
        0.00371911,  0.00132873,  0.01029234, -0.00692149,  0.01511791,
        0.00788964,  0.01075627,  0.00214834, -0.0003685 , -0.01020754,
        0.01465225,  0.01498591,  0.00297053, -0.00798114, -0.01076315,
       -0.00813203, -0.0080655 , -0.02228117, -0.01079776,  0.01073463,
       -0.02298366, -0.00251261,  0.00582854,  0.013827  , -0.00796133,
       -0.00107606, -0.05417154,  0.00042612,  0.01153702,  0.01803304,
        0.00378542, -0.00740062,  0.01578713,  0.00156955,  0.00261571,
       -0.01721421,  0.01366435,  0.02432611,  0.00220118,  0.00102544,
       -0.01055659,  0.01635489,  0.00197147, -0.00758277, -0.01703242,
       -0.00070481,  0.00241459, -0.01344026, -0.01479331, -0.02315449,
       -0.06334497,  0.00987198, -0.00639078, -0.0064891 ,  0.02462263,
```

```
        0.01482894,  0.00189785,  0.0057214 ,  0.00329171, -0.00944135,
        0.01442143,  0.0053822 ,  0.00476526,  0.00926259, -0.00444799,
        0.00237533, -0.0007594 , -0.00952157,  0.00919402, -0.01368201,
        0.00053713,  0.01208925,  0.0083601 , -0.00704379,  0.01694042,
       -0.00077307,  0.00187013,  0.00738904, -0.0072635 ,  0.00541677,
        0.01679274, -0.00579622,  0.00283572,  0.00586468, -0.00323854,
        0.00255446, -0.00332872,  0.00105644,  0.00896265,  0.00170191,
        0.00381307, -0.00495849, -0.00292552, -0.0021091 , -0.00276153,
       -0.01202422, -0.00315326, -0.01717053, -0.03682951, -0.0021991 ,
        0.02183782,  0.00482149,  0.00141054, -0.00897655,  0.01940595,
       -0.01132633,  0.00602998, -0.00797925, -0.02112556, -0.00117125,
       -0.01371384, -0.00698914, -0.01202393, -0.01712621,  0.02033712,
        0.00786401,  0.0073981 ,  0.02608688,  0.01318002,  0.00128681,
       -0.00818612, -0.02154283, -0.01392595,  0.02213621, -0.01121129,
        0.00521375, -0.00481485,  0.01712803])
```

<font size = 3 face = '宋体'>600656.XSHG在2016-05-12当日处于停牌状态

In [15]:

```
Rp = pd.Series(portfolio_value_rate,index = price_change_rate.index)-price_change_rate['risk_free_rate']
```

<font size = 3 face = '宋体'>开始做回归

In [16]:

```
X = pd.DataFrame()
X['Rb'] = np.array(Rb)
X['SMB'] = np.array(SMB)
X['HML'] = np.array(HML)
X = sm.add_constant(X)
Y = np.array(Rp)
```

In [17]:

```
est=sm.OLS(Y,X).fit()
print(est.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.768
Model:                            OLS   Adj. R-squared:                  0.765
Method:                 Least Squares   F-statistic:                     264.2
Date:                Sat, 22 Jul 2017   Prob (F-statistic):           1.37e-75
Time:                        15:46:36   Log-Likelihood:                843.56
No. Observations:                 243   AIC:                            -1679.
Df Residuals:                     239   BIC:                            -1665.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          0.0015      0.001      3.072      0.002         0.001     0.003
Rb             1.0038      0.074     13.609      0.000         0.859     1.149
SMB            1.2257      0.139      8.831      0.000         0.952     1.499
HML           -0.7168      0.126     -5.668      0.000        -0.966    -0.468
==============================================================================
Omnibus:                       11.937   Durbin-Watson:                   1.627
Prob(Omnibus):                  0.003   Jarque-Bera (JB):               12.513
Skew:                           0.485   Prob(JB):                      0.00192
Kurtosis:                       3.543   Cond. No.                         356.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [45]:

```
est.params
```

Out[45]:

```
const    0.001540
Rb       1.003847
SMB      1.225742
HML     -0.716755
dtype: float64
```

 上表主要看coef和t值。 1、SMB大于0，和过去的事实符合，小盘股具有较好的收益率
2、HML小于0，说明bp大的组合（即pb较小的组合）的组合为负的权重，价值投资不能片面追求较小的pb
3、t的绝对值大于2，我们一般认为系数显著。

$$Rp = 0.0015 + 1.0036Rb + 1.2258SMB - 0.7162HML$$

我们在附录位置详细描述了各个指标具体的含义：请点击

第二节：Fama-French 三因子策略¶

<font size = 3 face = '宋体'> 1、通过SMB和HML将所有股票分为了6份，我们选出其中总收益最好的一份所在的区域
2、选出当期区域内的股票
3、基本面分析继续筛选，缩小股票的范围

In [8]:

```
from rqalpha_plus.api import *
from rqalpha_plus import run_func
########################################################################################################################################################
# 可以自己import我们平台支持的第三方python模块，比如pandas、numpy等。
import numpy as np
import pandas as pd
from pandas import *
import datetime as dt

# 在这个方法中编写任何的初始化逻辑。context对象将会在你的算法策略的任何方法之间做传递。
def init(context):
    context.now
    context.last_month = 13
    context.buy_stocks = []

# before_trading此函数会在每天策略交易开始前被调用，当天只会被调用一次
def before_trading(context):
    # 是否调仓，每日更新为不调仓
    context.change = False
    context.buy_stocks = []
    if context.now.month == context.last_month:
        return None
    context.last_month = context.now.month
    aera = fin_alpha_area(context)
    ready_stocks = fin_stocks(aera,context)
    #stocks = get_buy_stock(ready_stocks,context)
    context.buy_stocks = get_my_stocks(ready_stocks,context)
    context.change = True


# 你选择的证券的数据更新将会触发此段逻辑，例如日或分钟历史数据切片或者是实时数据切片更新
def handle_bar(context, bar_dict):
    if context.change != True:
        return None
    for stock in context.portfolio.stock_account.positions:
        if stock not in context.buy_stocks:
            order_target_percent(stock,0)

    for stock in context.buy_stocks:
        order_target_percent(stock,1/len(context.buy_stocks))

# 彼得林奇
def get_my_stocks(stocks,context):
    df = get_fundamentals(
        query(fundamentals.financial_indicator.cash_flow_from_operations_per_share
        ).filter(
            fundamentals.financial_indicator.debt_to_asset_ratio <= 100 # 资产负债率小于等于25%
        ).filter(
            fundamentals.financial_indicator.cash_flow_from_operations_per_share > 0 # 每股净现金大于0
        ).filter(
            fundamentals.financial_indicator.annual_return_on_equity > 20 # 当前股价与每股自由现金流量比小于11
        ).filter(
            financials.stockcode.in_(stocks))
    ).T
    if len(df.index) <= 2:
        return ['000012.XSHG']
    df = df.sort_values('cash_flow_from_operations_per_share',ascending=False)
    buy_stocks = [stock for stock in df.index if is_suspended(stock) != True]
    return buy_stocks[:20]

# 横盘的10支股票
def get_cheap_stocks(stocks,context):
    start_date = (context.now-dt.timedelta(days = 90)).strftime("%Y-%m-%d")
    end_date = (context.now-dt.timedelta(days = 1)).strftime("%Y-%m-%d")
```

```
    price = get_price(stocks,start_date = start_date,end_date = end_date,fields = ['close'],adjust_type='pre')
    price =np.abs(price.iloc[0,:]/price.iloc[-1,:]-1)
    price = price.sort_values(ascending=True)
    buy_stocks = [stock for stock in price.index if is_suspended(stock) != True]

    return buy_stocks[-20:-10]

# 买入roe最大的10支股票
def get_buy_stock(stocks,context):
    value = get_fundamentals(
        query(
            fundamentals.financial_indicator.annual_return_on_equity,
        ).filter(financials.stockcode.in_(stocks)),
            entry_date = context.now - dt.timedelta(days = 1),interval = '1d', report_quarter = False).T
    value.sort_values('annual_return_on_equity',ascending=False, inplace=True)
    buy_stocks = [stock for stock in value.index if is_suspended(stock) != True]

    return buy_stocks[0:int(len(buy_stocks)/3)]


# after_trading函数会在每天交易结束后被调用，当天只会被调用一次
def after_trading(context):
    pass

def fin_stocks(aera,context):
    # 用查询一年前所有股票的基本面信息
    all_stocks_value = get_fundamentals(
        query(
            fundamentals.eod_derivative_indicator.pb_ratio,
            fundamentals.eod_derivative_indicator.market_cap
        ),
            entry_date = context.now - dt.timedelta(days = 1),interval = '1d', report_quarter = False).T
    all_stocks_value.dropna(how = 'any',inplace = True)
    all_stocks_value['bp_ratio'] = 1/all_stocks_value['pb_ratio']
    del all_stocks_value['pb_ratio']

    # 对股票进行分组
    calue_copy = all_stocks_value.iloc[:,:]
    lengh = len(all_stocks_value.index)
    all_stocks_value = all_stocks_value.sort_values('market_cap') #市值升序排布
    S = all_stocks_value.index[:int(lengh/2)]
    B = all_stocks_value.index[int(lengh/2):]
    all_stocks_value = all_stocks_value.sort_values('bp_ratio') #净市率升序排列
    L = all_stocks_value.index[:int(lengh*0.3)]
    M = all_stocks_value.index[int(lengh*0.3):int(lengh*0.7)]
    H = all_stocks_value.index[int(lengh*0.7):]

    # 分为6类别
    portfolio = pd.Series()
    portfolio['SL'] = list(set(S).intersection(set(L)))
    portfolio['SM'] = list(set(S).intersection(set(M)))
    portfolio['SH'] = list(set(S).intersection(set(H)))
    portfolio['BL'] = list(set(B).intersection(set(L)))
    portfolio['BM'] = list(set(B).intersection(set(M)))
    portfolio['BH'] = list(set(B).intersection(set(H)))

    return portfolio[aera]

def fin_alpha_area(context):
    last_year = context.now - dt.timedelta(days = 150)
    yesterday = context.now - dt.timedelta(days = 1)

    # 得到所有的股票
    sz = index_components('399107.XSHE')
    sh = index_components('000002.XSHG')
    sz.extend(sh)
    all_stocks = sz.copy()

    # 拿到所有股票日收益率变动
    all_price = get_price(all_stocks,start_date=last_year-dt.timedelta(days =1), end_date=yesterday,fields = 'close')
    all_price = ((all_price.T).dropna(how = 'any',inplace = False)).T
    all_stocks = all_price.columns.values
    all_price_change_rate = all_price.pct_change().dropna(how = 'any',inplace = False)

    # 用查询一年前所有股票的基本面信息
    all_stocks_value = get_fundamentals(
        query(
            fundamentals.eod_derivative_indicator.pb_ratio,
            fundamentals.eod_derivative_indicator.market_cap
        ).filter(
            financials.stockcode.in_(all_stocks)),
            entry_date = last_year,interval = '1d', report_quarter = False).T
```

```
    all_stocks_value.dropna(how = 'any',inplace = True)
    all_stocks_value['bp_ratio'] = 1/all_stocks_value['pb_ratio']
    del all_stocks_value['pb_ratio']

    # 对股票进行分组
    calue_copy = all_stocks_value.iloc[:,:]
    lengh = len(all_stocks_value.index)
    all_stocks_value = all_stocks_value.sort_values('market_cap') #市值升序排布
    S = all_stocks_value.index[:int(lengh/2)]
    B = all_stocks_value.index[int(lengh/2):]
    all_stocks_value = all_stocks_value.sort_values('bp_ratio') #净市率升序排列
    L = all_stocks_value.index[:int(lengh*0.3)]
    M = all_stocks_value.index[int(lengh*0.3):int(lengh*0.7)]
    H = all_stocks_value.index[int(lengh*0.7):]

    # 分为6类别
    portfolio = {}
    portfolio['SL'] = list(set(S).intersection(set(L)))
    portfolio['SM'] = list(set(S).intersection(set(M)))
    portfolio['SH'] = list(set(S).intersection(set(H)))
    portfolio['BL'] = list(set(B).intersection(set(L)))
    portfolio['BM'] = list(set(B).intersection(set(M)))
    portfolio['BH'] = list(set(B).intersection(set(H)))

    # 计算每类平均收益
    return_ = pd.Series()
    for i in portfolio:
        return_[i] = np.array(all_price_change_rate[portfolio[i]]).mean()
    return_.sort_values(ascending=False, inplace=True)

    #print(return_)

    return return_.index[0]

config = {
  "base": {
    "start_date": "2016-01-01",
    "end_date": "2017-06-01",
    "benchmark": "000300.XSHG",
    'accounts': {
                'stock': 100000},
  },
  "extra": {
    "log_level": "verbose",
  },
  "mod": {
    "sys_analyser": {
      "enabled": True,
      "plot": True
    }
  }
}

# 您可以指定您要传递的参数
run_func(init=init, before_trading=before_trading, handle_bar=handle_bar, config=config)

from rqdatac import *
```

```
[2017-07-19 17:18:19.911381] DEBUG: system_log:
{'base': {'accounts': {'STOCK': 100000.0},
          'benchmark': '000300.XSHG',
          'data_bundle_path': '/home/user_303036/.rqalpha-plus/bundle',
          'end_date': datetime.date(2017, 6, 1),
          'frequency': '1d',
          'margin_multiplier': 1,
          'persist': False,
          'persist_mode': PERSIST_MODE.REAL_TIME,
          'resume_mode': False,
          'run_type': RUN_TYPE.BACKTEST,
          'source_code': None,
          'start_date': datetime.date(2016, 1, 1),
          'strategy_file': 'strategy.py'},
 'extra': {'context_vars': None,
           'dividend_reinvestment': False,
           'enable_profiler': False,
           'force_run_init_when_pt_resume': False,
           'is_hold': False,
           'locale': 'zh_Hans_CN',
           'log_level': 'verbose',
           'user_log_disabled': False,
```

```
          'user_system_log_disabled': False},
 'mod': {'indicator': {'enabled': True, 'lib': 'rqalpha_mod_indicator'},
         'ricequant_data': {'enabled': True,
                            'lib': 'rqalpha_mod_ricequant_data',
                            'night_trading': False,
                            'priority': 101,
                            'rqdata_client_addr': 'q-rqdatad',
                            'rqdata_client_password': 'research',
                            'rqdata_client_port': 16003,
                            'rqdata_client_retry_cnt': 30,
                            'rqdata_client_username': 'research'},
         'sys_accounts': {'enabled': True},
         'sys_analyser': {'enabled': True, 'plot': True},
         'sys_funcat': {'enabled': False},
         'sys_progress': {'enabled': True},
         'sys_risk': {'enabled': True},
         'sys_simulation': {'enabled': True},
         'sys_stock_realtime': {'enabled': False}},
 'validator': {'cash_return_by_stock_delisted': False, 'close_amount': True},
 'version': '0.1.6',
 'whitelist': ['base', 'extra', 'validator', 'mod']}
[2017-07-19 17:18:19.916930] DEBUG: system_log: 载入 Mod rqalpha_mod_ricequant_data
[2017-07-19 17:18:19.919563] DEBUG: system_log: 载入 Mod rqalpha_mod_indicator
[2017-07-19 17:18:19.922337] DEBUG: system_log: 载入 Mod rqalpha.mod.rqalpha_mod_sys_analyser
[2017-07-19 17:18:19.924950] DEBUG: system_log: 载入 Mod rqalpha.mod.rqalpha_mod_sys_accounts
[2017-07-19 17:18:19.927580] DEBUG: system_log: 载入 Mod rqalpha.mod.rqalpha_mod_sys_simulation
[2017-07-19 17:18:19.930364] DEBUG: system_log: 载入 Mod rqalpha.mod.rqalpha_mod_sys_risk
[2017-07-19 17:18:19.933089] DEBUG: system_log: 载入 Mod rqalpha.mod.rqalpha_mod_sys_progress
[2017-07-19 17:18:19.936268] INFO: system_log: rqdatac use socket ('q-rqdatad', 16003)
2016-04-01 WARN 600870.XSHG 在 2016-04-01 15:00:00 时停牌。
2016-05-03 WARN 600870.XSHG 在 2016-05-03 15:00:00 时停牌。
```

```
No record found
```

回测收益	回测年化收益	Alpha	Beta	Sharpe	Sortino	Information Ratio
7.600%	5.300%	0.069	1.19	0.241	0.499	0.374
基准收益	基准年化收益	Volatility	MaxDrawdown	Tracking Error	Downside Risk	最大回撤/最大回撤持续期
0.800%	0.600%	0.276	28.700%	0.184	0.133	MaxDD 2016-01-04 00:00:00~2016-01-28 00:00:00, 24 days MaxDDD 2016-07-12 00:00:00~2017-06-01 00:00:00, 324 days


```
[2017-07-19 17:21:28.812819] DEBUG: system_log: 策略运行成功，正常退出
```

第三节：Fama-French 因子风险暴露Factor Risk Exposure ¶

 运用多因子模型分析一个组合中风险和收益的来源，多因子模型对收益的分解如下：

$$R_i = a_i + b_{i1}F_1 + b_{i2}F_2 + \ldots + b_{iK}F_K + \epsilon_i$$

在风险分析中，我们常常对超额收益（相对基准的收益情况）建模，风险用超额收益（active Return）的标准误来体现。也叫做tracking error。

例如因子对超额收益风险平方的边际贡献（factors' marginal contributions to active risk squared，FMCAR ） 对于因子 j, 这个指标是这样计算的：

$$\text{FMCAR}_j = \frac{b_j^a \sum_{i=1}^{K} b_i^a Cov(F_j, F_i)}{(\text{Active risk})^2}$$

b_i^a 是组合对因子i的风险暴露。这项指标告诉我们了，假设其他条件不变，暴露在因子j下我们增加了多少风险。

<font size = 3 face = '宋体'> 在上文的基础商，我们计算SMB和HML的累计收益率

In [ ]:

```
SMB_cu = pd.Series(index = SMB.index)
HML_cu = SMB_cu.copy()

for i in range(len(SMB)):
    if i == 0:
        print("ok")
        SMB_cu[i] = 1+SMB[i]
```

```
        HML_cu[i] = 1+HML[i]
    else:
        SMB_cu[i] = SMB_cu[i-1]*(SMB[i]+1)
        HML_cu[i] = HML_cu[i-1]*(HML[i]+1)
df = pd.DataFrame()
df['SMB_cu'] = SMB_cu
df['HML_cu'] = HML_cu
df.iplot(kind='scatter', filename='累计收益率')
```

 SMB(t)是一个买入小盘股卖出大盘股的组合收益率。HML(t)是一个买入高净市率股票(低pb)卖空低净市率(高pb)股票的投资组合收益率。

计算风险暴露程度¶

 下面我们运用多因素模型和线性回归工具来计算某只股票的回报率相对于这些因子的风险暴露程度。我们以某个资产组合的超额收益作为被解释变量，对因子做回归，一个因子对超额收益贡献越大，那么这个资产组合的超额收益对于该因子的暴露程度也越高。我们前面已经存在了一个小市值的股票组合了

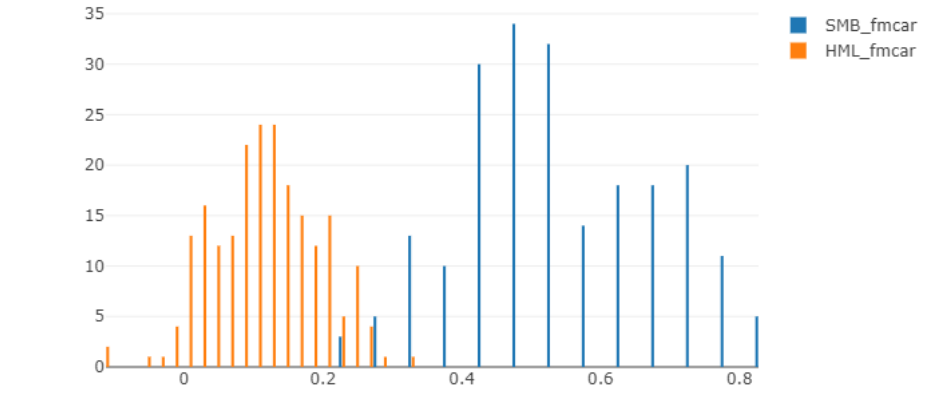
In [ ]:

```
def FMACR(SMB,HML,Rb,Rp):
    X = pd.DataFrame()
    X['Rb'] = np.array(Rb)
    X['SMB'] = np.array(SMB)
    X['HML'] = np.array(HML)
    X = sm.add_constant(X)
    Y = np.array(Rp)
    est=sm.OLS(Y,X).fit()
    active = Rp-est.params['Rb']*Rb

    cov = np.cov(SMB, HML)
    ar_squared = (active.std())**2
    SMB_fmcar = (est.params['SMB']*(est.params['HML']*cov[0,1] + est.params['SMB']*cov[0,0]))/ar_squared
    HML_fmcar = (est.params['HML']*(est.params['SMB']*cov[0,1] + est.params['HML']*cov[1,1]))/ar_squared

    return SMB_fmcar,HML_fmcar

fmcar = pd.DataFrame(index = Rp.index[30:],columns = ['SMB_fmcar','HML_fmcar'])
for i in range(30,len(Rp)):
    fmcar['SMB_fmcar'][i-30],fmcar['HML_fmcar'][i-30] = FMACR(SMB.iloc[i-30:i],HML.iloc[i-30:i],Rb.iloc[i-30:i],Rp.iloc[i-30:i])
fmcar.iplot(kind='scatter', filename='FMACR')
```

<font size = 3 face = '宋体'> 了解历史数据中组合对各个因子的暴露程度是很有趣的，但只有将它用在对未来收益率的预测上时，它才有用武之地。但我们不是总能够放心地认为未来的情况与现在相同，由于随时间会变化，对风险暴露程度取平均值也很容易出现问题。我们可以给均值加上一个置信区间，但只有当收益率的分布是正态分布或者表现很稳健才行。我们来看看Jarque-Bera测验的结果。

In [ ]:

```
data = [go.Histogram(x=np.array(fmcar['SMB_fmcar'])),go.Histogram(x=np.array(fmcar['HML_fmcar']))]
data[0].name = 'SMB_fmcar'
data[1].name = 'HML_fmcar'
py.iplot(data, filename='SMB_FMCAR')
```



In [21]:

```
from statsmodels.stats.stattools import jarque_bera
JB_SMB, _, _, _ = jarque_bera(fmcar['SMB_fmcar'])
JB_HML, _, _, _ = jarque_bera(fmcar['HML_fmcar'])

print ('JB SMB_FMCAR is normally distributed', JB_SMB)
print ('JB HML_FMCAR is normally distributed', JB_HML)
```

```
JB SMB_FMCAR is normally distributed 4.628242740982365
JB HML_FMCAR is normally distributed 0.06063453785474304
```

第四节：Fama五因素模型¶

<font face = '宋体' size = 3> 由Eugene F. Fama和Kenneth R. French撰写Journal of Financial Economics2015年第4期论文"A five-factor asset pricing model"对原有的Fama-French（1993）三因素模型进行了改进，在原有的市场、公司市值（即SML，small minus large）以及账面市值比（即HML，high minus low）三因子的基础上，加入了盈利能力（profitability）因子（即RMW，robust minus weak）和投资模式（investment patterns）因子（即CMA，conservative minus aggressive），从而能够更好地解释股票横截面收益率的差异。 对于这个新的模型，其基本特性和一些要素总结如下：

- 定义盈利能力因子（profitability）为高/低盈利股票投资组合的回报之差（Robust Minus Weak）

- 定义投资模式因子（investment pattern）为高/低投资比例公司股票投资组合的回报之差（Conservative Minus Aggressive）
- 五因子模型比三因子模型能更好地解释股票的回报
- 五因子模型存在的问题是对于高投资比例低盈利小盘股的回报解释不甚理想
- 加上了盈利能力和投资模式两个新因子之后，原先的价值因子似乎变得多余

In [23]:

```
# 取2016-05-15的估值数据作为SMB和HML的计算标准
all_stocks_value = get_fundamentals(
    query(
        fundamentals.eod_derivative_indicator.pb_ratio,
        fundamentals.eod_derivative_indicator.market_cap,
        fundamentals.financial_indicator.annual_return_on_equity,
        fundamentals.financial_indicator.inc_total_asset)
    , '2016-05-15', interval = '1d' ,report_quarter = False)
all_stocks_value = all_stocks_value.major_xs(all_stocks_value.major_axis[0])
all_stocks_value['bp_ratio'] = 1/all_stocks_value['pb_ratio']
del all_stocks_value['pb_ratio']
all_stocks_value = all_stocks_value.dropna(how = 'any')
all_stocks_value.head()
```

Out[23]:

	market_cap	annual_return_on_equity	inc_total_asset	bp_ratio
000005.XSHE	7.47327e+09	-2.3836	-3.7934	0.152798
600602.XSHG	1.26845e+10	3.6792	-2.5571	0.262
600651.XSHG	1.02955e+10	2.5172	40.3337	0.317914
600652.XSHG	1.14247e+10	1.5756	3.2331	0.182093
600654.XSHG	2.46982e+10	7.4944	-7.3252	0.119852

In [24]:

```
# 拿到所有股票的日收益率数据
all_price_change_rate = get_price_change_rate(list(all_stocks_value.index),start_date='20160514', end_date='20170515')
all_price_change_rate.head()
```

Out[24]:

order_book_id	000001.XSHE	000002.XSHE	000004.XSHE	000005.XSHE	000006.XSHE	000007.XSHE	000008.X
date							
2016-05-16	0.000966	0.0	0.0	-0.004250	0.015193	0.0	0.002573
2016-05-17	-0.005792	0.0	0.0	-0.004267	0.004082	0.0	-0.023952
2016-05-18	0.000000	0.0	0.0	-0.044286	-0.025745	0.0	-0.041192
2016-05-19	-0.004854	0.0	0.0	0.004485	0.001391	0.0	0.003656
2016-05-20	0.004878	0.0	0.0	0.013393	0.002778	0.0	-0.002732

5 rows × 2823 columns

In [25]:

```
# 去除nan
test = all_price_change_rate.T.dropna(how = 'any').T
test.head()
```

Out[25]:

order_book_id	000001.XSHE	000002.XSHE	000004.XSHE	000005.XSHE	000006.XSHE	000007.XSHE	000008.X
date							
2016-05-16	0.000966	0.0	0.0	-0.004250	0.015193	0.0	0.002573
2016-05-17	-0.005792	0.0	0.0	-0.004267	0.004082	0.0	-0.023952
2016-05-18	0.000000	0.0	0.0	-0.044286	-0.025745	0.0	-0.041192
2016-05-19	-0.004854	0.0	0.0	0.004485	0.001391	0.0	0.003656
2016-05-20	0.004878	0.0	0.0	0.013393	0.002778	0.0	-0.002732

5 rows × 2821 columns

In [26]:

```
stock_need_drop = list(set(all_price_change_rate.columns).difference(set(test.columns)))
stock_need_drop
```

Out[26]:

```
['600005.XSHG', '000748.XSHE']
```

In [27]:

```
# 在估值数据中将上述两支股票删除
all_stocks_value = all_stocks_value.T
for stock in stock_need_drop:
    del all_stocks_value[stock]
all_stocks_value = all_stocks_value.T
```

In [28]:

```
short = int(len(all_stocks_value.index)*0.3)
long = int(len(all_stocks_value.index)*0.7)

all_stocks_value.sort_values('market_cap',ascending=True, inplace=True)
SMB = all_price_change_rate[all_stocks_value.index[:short]].T.mean()-all_price_change_rate[all_stocks_value.index[long:]].T.mean()

all_stocks_value.sort_values('bp_ratio',ascending=False, inplace=True)
HML = all_price_change_rate[all_stocks_value.index[:short]].T.mean()-all_price_change_rate[all_stocks_value.index[long:]].T.mean()

all_stocks_value.sort_values('annual_return_on_equity',ascending=False, inplace=True)
RMW = all_price_change_rate[all_stocks_value.index[:short]].T.mean()-all_price_change_rate[all_stocks_value.index[long:]].T.mean()

all_stocks_value.sort_values('inc_total_asset',ascending=False, inplace=True)
CMA = all_price_change_rate[all_stocks_value.index[:short]].T.mean()-all_price_change_rate[all_stocks_value.index[long:]].T.mean()
```

In [ ]:

```
SMB_cu = pd.Series(index = SMB.index)
HML_cu = SMB_cu.copy()
RMW_cu = SMB_cu.copy()
CMA_cu = SMB_cu.copy()

for i in range(len(SMB)):
    if i == 0:
        print("ok")
        SMB_cu[i] = 1+SMB[i]
        HML_cu[i] = 1+HML[i]
        RMW_cu[i] = 1+RMW[i]
        CMA_cu[i] = 1+CMA[i]
    else:
        SMB_cu[i] = SMB_cu[i-1]*(SMB[i]+1)
        HML_cu[i] = HML_cu[i-1]*(HML[i]+1)
        RMW_cu[i] = RMW_cu[i-1]*(RMW[i]+1)
        CMA_cu[i] = CMA_cu[i-1]*(CMA[i]+1)
df = pd.DataFrame()
df['SMB_cu'] = SMB_cu
df['HML_cu'] = HML_cu
df['RMW_cu'] = RMW_cu
df['CMA_cu'] = CMA_cu
df.iplot(kind='scatter', filename='累计收益率')
```

<font face = '宋体' size = 3>　对比上述的图，我们可以发现，同样是SMB平均仓位买入和按庄重买入有完全不同效果，按权重买入，我们在大市值的公司暴露度较好，而平均仓位买入在17年4月份，我们面临一个极大的回调，与此相对因的低pb的公司则收益见好。当然，使用模型不必拘谨于模型本身。

附录： ¶

<font face = '宋体' size = 3> Dep.variable: 因变量，随自变量的变动而变动

Model: 模型类型（此处为线性回归模型)

Method: 参数估计方法（此处为最小二乘法）

No.observation:样本数量

Df.residuals:残差的自由度，也就是说当用样本来估计整体残差的时候，样本中独立能自由变化的数据的个数

Df.model: 统计模型的自由度，也就等于能自由取值的自变量的个数

Covariance type: 协方差类型

R-squared: 决定系数，即R^2，意味着模型的拟合度的好坏, R-squared误差取值范围为0到1，这个值越接近1说明模型的拟合度越好。 $R-squared=SSR/SST$(SSR是预测数据与原始数据均值之差的平方和，SST是原始数据和均值之差的平方和)

Adj.R-squared:调整后的决定系数，

$$Adj.\,R-squared = 1 - \frac{(1-R^2)(n-1)}{n-k-1}$$

（n为自变量数目，k为解释变量的数目）

F-statistique: F 检验统计量，是模型总体显著性检验的指标,它越大,模型越好。$F=\frac{R^2/k}{(1-R^2)/(n-k-1)}$

Prob(F-statistique): F检验统计量对应的概率

Log likelihood: 对数似然比检验的值，值越高，说明模型越精准。其公式为$F=\frac{-n}{2}log(2π) - \frac{n}{2} logσ^2- \frac{n}{2}$

AIC: 赤池信息量即Akaike information criterion，是衡量统计模型拟合优良性的一种标准。
在一般的情况下，AIC可以表示为： $AIC=2k-2ln(L)$，其中k是参数数量，L是似然函数。假设条件是模型的误差服从独立正态分布。 让n为观察数，SSR(SUM SQAURE OF RESIDUE)为残差平方和，那么AIC变为： $AIC=2k+nln(SSR/n)$。增加自由参数的数目提高了拟合的优良性，AIC鼓励数据拟合的优良性但是尽量避免出现过度拟合(Overfitting)的情况。所以优先考虑的模型应是AIC值最小的那一个。

BIC: Bayesian information criterion，贝叶斯信息度量。 $BIC=kln(n)-2ln(L)$。其中n是模型中未知参数的数量。 BIC相比AIC在大数据量时对模型参数惩罚得更多，导致BIC更倾向于选择参数少的简单模型。
Coef: 回归系数的估计值。 $β^=（X^T X）^{-1}$ X^TY

Std err: 回归系数估计量的标准差

t: T检验的统计量，即样本平均数与总体平均数的离差统计量。因为样本x符合正太分布，那么 $t=\frac{X¯-μ}{σ/√(n-1)}$ 符合自由度为n-2的T分布,其中t为样本平均数与总体平均数的离差统计量，X¯为样本平均数，μ为总体平均数，σ为样本标准差，n为样本容量.。当t大于自由度为n-2的T分别的临界分位点时，我们拒绝零假设

P>|t|: t对应的T分布的概率，当此概率小于临界值的概率（通常是5%），我们拒绝零假设

Omnibus: 模型的总的全局检验，为似然比检验.

Prob(omnibus):

Skew: 偏度系数。当分布对称时，偏度系数为0.当偏度系数为正值，可以判断为右偏。反之判断为左偏。 $Skew=\frac{1}{n-1}$ $\sum_{i=1}^n\frac{(x_i-X¯)^3}{S^3}$，其中S为标准差

Kurtosis: 峰度系数。是对数据分布平峰或尖峰程度的测度。峰度是针对正态分布而言的。峰度系数为0，表明数据为正态分布。若峰度系数大于0，则数据为尖峰分布；否则为平峰分布。$Kurtosis=\frac{1}{n-1}$ $\sum_{i=1}^n\frac{(x_i-X¯)^4}{S^4}-3$
Durbin-Watson: 检验残差中是否存在自相关。自相关表示相邻观测值是相关的。 如果它们是相关的，那么最小二乘回归低估了系数的标准误；此时，您的预测变量似乎非常显著，其实可能是不显著的。例如，来自每日股票价格数据回归的残差可能取决于以前的观测值，因为前一天的股票价格会影响第二天的价格。 Durbin-Watson 统计量以观测值的顺序（行）为条件。Minitab 假设观测值遵循有意义的顺序（如时间顺序）。Durbin-Watson 统计量确定相邻误差项之间的相关性是否为零。要从检验中得出结论，需要将显示的统计量与表中的上下限进行比较。如果 D > 上限，表示不存在相关性；如果 D < 下限，表示存在正相关性；如果 D 在上下限之间，则无法从检验中得出结论。

Jarque-Bera(JB): Jarque-Bera检验基于数据样本的偏度和峰度，评价给定数据服从未知均值和方差正态分布的假设是否成立。
$JB=\frac{n-k}{6}($S^2$+$\frac{（k-3）^2}{4}$)，其中n为样本容量，k为自由度， S为标准差。

Prob(JB)： JB统计量的相伴概率，若小于设定的概率水平，则拒绝原假设，不认样本概率服从正态分布。

验证Fama-French 三因子分解的有效性¶

本文将介绍著名经济学家尤金 法码与其Fama-French 三因素模型，并利用三因素模型对多因子打分模型进行因子收益率分解。

尤金·法玛（Eugene F. Fama），著名经济学家、金融经济学领域的思想家，芝加哥经济学派代表人物之一、芝加哥大学教授，2013年诺贝尔经济学奖得主。

1990年代初，法玛与麻省理工学院的肯尼斯·弗伦奇(Kenneth French)教授合作，检验了几种可选择的财务数据能够提高股票回报与经济活动的预测度。他们在宏观经济层面上检验这些数据，同时又在公司水平上检验，例如股利的产生；又在股票回报与商业波动中检验他们的相互关系。除了考虑股票价格相对指数的波动外，他们还考虑了公司的规模以及市净率分类等因素，构建了包含市场因子、规模因子和价值因子的三因素模型。三因素模型的提出的确解释了CAPM模型不能解释的“异常”问题；三因素模型还可用来测度基金的业绩，以考察基金经理的投资能力。

模型认为，一个投资组合(包括单个股票)的超额回报率可由它对*三个因子的暴露*来解释，这三个因子是：市场资产组合($R_m - R_f$)、市值因子(SMB)、账面市值比因子(HML)。这个多因子均衡定价模型可以表示为：

$$E(R_i) - R_f = \beta(E(R_m) - R_f) + sSMB + hHMI$$

其中，

- R_f表示无风险收益率；
- R_m为市场收益率；
- R_i为投资组合收益率
- SMB为市值因子收益率
- HMI为账面市值比因子收益率

1、理论假设

在探讨Fama—French三因子模型的应用时，是以“有限理性”理论假设为基础。并在此基础上得出若干基本假定：

(1) 存在着大量投资者；

(2)所有投资者都在同一证券持有期计划自己的投资资产组合；

(3)投资者投资范围仅限于公开金融市场上交易的资产；

(4)不存在证券交易费用(佣金和服务费用等)及税赋；

(5)投资者们对于证券回报率的均值、方差及协方差具有相同的期望值；

(6)所有投资者对证券的评价和经济局势的看法都一致。

2、统计假设

从模型的表达式可以看出，FF模型属于多元回归模型。其基本假设为：

(1)$(R_m - R_f)、SMB、HML$与随机误差项u不相关；

(2)零均值假定：$E(u) = 0$；

(3)同方差假定，即u的方差为一常量：$Var(u) = \sigma ^2$；

(4)无自相关假定：$cov(u_i,u_j)=0,i!=j$；

(5)解释变量之间不存在线性相关关系。即两个解释变量之间无确切的线性关系；

(6)假定随机误差项u服从均值为零，方差为σ^2正态分布，即$u-N(0,\sigma^2)$。

因子收益率的计算¶

In [1]:

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

In [2]:

```
Factor_booktoshare_high=rqportal.portfolio(1643340) #高账面市值比
Factor_booktoshare_low=rqportal.portfolio(1643347)# 低账面市值比
```

In [3]:

```
Factor_market_cap_small=rqportal.portfolio(1643385)
Factor_market_cap_big=rqportal.portfolio(1643384)
```

In [4]:

```
#账面市值比因子收益率
HML=Factor_booktoshare_high.daily_returns.values-Factor_booktoshare_low.daily_returns.values
```

In [5]:

```
#市值因子收益率
SMB=Factor_market_cap_small.daily_returns.values-Factor_market_cap_big.daily_returns.values
```

In [6]:

```
#市场因子收益率
RM=Factor_market_cap_small.benchmark_daily_returns.values
```

In [22]:

```
plt.figure(figsize=[20,8])
plt.plot(np.cumprod(HML+1))
plt.plot(np.cumprod(SMB+1))
plt.plot(np.cumprod(RM+1))
plt.legend(['HML_return_curve','SMB_return_curve','RM_return_curve'])
```

Out[22]:

```
<matplotlib.legend.Legend at 0x7f7b7d145fd0>
```

三因子相关性热力图：

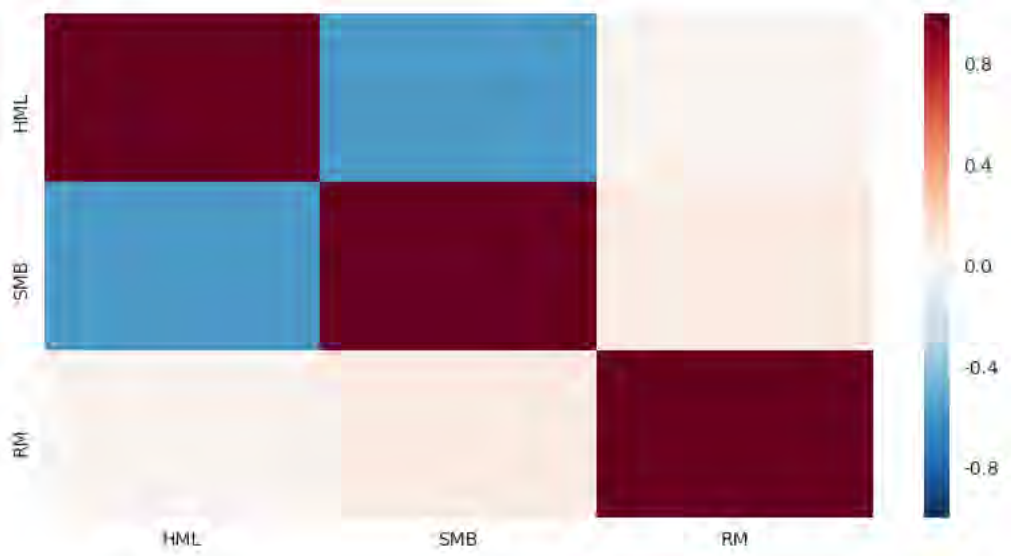
In [48]:

```
import seaborn as sns

cov_matrix=np.cov(X.T)

plt.figure(figsize=[10,5])
sns.heatmap(np.corrcoef(X.T),xticklabels=['HML','SMB','RM'],yticklabels=['HML','SMB','RM'])
```

```
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```

Out[48]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7b7b749550>
```



多因子打分模型回顾¶

所选因子：

- 股票池：沪深300
- 打分规则：个股在每个因子上的排名求算术平均（因子等权重）
- 股票选取：得分排名最高的10只
- 调仓频率：月度

In [12]:

```
#获取多因子模型收益率
MultiFactor=rqportal.portfolio(1604099).daily_returns.values
```

In [13]:

```
len(HML)==len(SMB)==len(RM)==len(MultiFactor)
```

Out[13]:

```
True
```

In [24]:

```
#回归分析
import statsmodels.api as sm
```

In [25]:

```
X=np.column_stack((HML,SMB,RM))
y=MultiFactor
```

In [26]:

```
X1 = sm.add_constant(X)
```

In [27]:

```
model = sm.OLS(y, X1)
results = model.fit()
```

In [28]:

```
results.summary()
```

Out[28]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.867
Model:	OLS	Adj. R-squared:	0.867
Method:	Least Squares	F-statistic:	3170.
Date:	Fri, 12 May 2017	Prob (F-statistic):	0.00
Time:	13:07:10	Log-Likelihood:	5439.8
No. Observations:	1458	AIC:	-1.087e+04
Df Residuals:	1454	BIC:	-1.085e+04
Df Model:	3		
Covariance T ype:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
const	0.0004	0.000	2.830	0.005	0.000 0.001
x1	0.1353	0.018	7.677	0.000	0.101 0.170
x2	0.3383	0.014	24.260	0.000	0.311 0.366
x3	0.8851	0.010	90.903	0.000	0.866 0.904

Omnibus:	142.737	Durbin-W atson:	1.913
Prob(Omnibus):	0.000	Jarque-Bera (JB):	393.667
Skew:	0.522	Prob(JB):	3.28e-86
Kurtosis:	5.322	Cond. No.	133.

可以看到，回归系数在统计上(95%置信度)都不为0

In [29]:

```
paras=results.params
```

In [30]:

```
y_predic=np.dot(X1,paras)
```

In [31]:

```
plt.figure(figsize=[20,8])
plt.plot(np.cumprod(y_predic+1))
plt.plot(np.cumprod(y+1))
plt.legend(['y_predicted','y_true'])
```

Out[31]:

```
<matplotlib.legend.Legend at 0x7f7b7d051f98>
```

此次分享到此为止，后续会基于三因子模型进行风险分解与评估，甚至将三因子模型扩展到五因子模型，敬请期待！

残差风险计算¶

由前文可知存在如下分解：

$$E(R_i) - R_f = \beta(E(R_m) - R_f) + sSMB + hHMI$$

与之对应的回归式为：

$$r_i = \alpha + \beta r_{mi} + ssmb_i + hhmi_i + u_i$$

波动率分解：

$$\sigma_r^2 = cov(\beta r_{mi} + ssmb_i + hhmi_i + u_i, \beta r_{mi} + ssmb_i + hhmi_i + u_i)$$

$$\sigma_r^2 = cov(\beta r_{mi} + ssmb_i + hhmi_i, \beta r_{mi} + ssmb_i + hhmi_i) + \sigma_u^2$$

我们主要关注残差风险(不可解释的风险来源)，即$\frac{\sigma_u}{\sigma_r}$

In [50]:

```
print(u'残差风险占比为%f'%(np.std(y_predic-y)/np.std(y)))
```

```
残差风险占比为0.364165
```

In [ ]:

```
```

套利定价模型¶

套利定价模型是金融领域中非常重要的资产定价理论。对于某个资产，该模型把其的收益使用线性因子模型来表示：

$$R_i = a_i + b_{i1}F_1 + b_{i2}F_2 + \ldots + b_{iK}F_K + \epsilon_i$$

该理论认为，如果我们可以按照上面的式子来表示收益的话，那么资产的预期收益就应该符合：

$$E(R_i) = R_F + b_{i1}\lambda_1 + b_{i2}\lambda_2 + \ldots + b_{iK}\lambda_K$$

其中R_F表示无风险收益，λ_j表示因子j的风险溢价。风险溢价的出现时因为人们需要更高的收益去补偿其受到的更高风险。知道资本资产定价模型（CAPM）的读者可以很明显的发现套利定价模型模型是其的推广形式，CAPM使用了市场回报作为其唯一的因子。

我们可以构建一个投资组合来计算λ_j，这个投资组合对因子j的弹性是1，对于其他因子的弹性为0（因子j的纯因素组合），之后得到超出无风险收益的回报。

套利¶

实际上我们可以选择许多许多股票来构建我们的投资组合。如果我们使用不同的股票来计算λs，我们的结果会一致吗？如果不一致的话，那么好了，就存在着套利空间。 比如，下面这种情况就存在套利空间： 某资产未来一年的预期回报是0.3，和市场的β是1.1，市场的预期收益是0.15，无风险利率是0.05。那么根据APT模型，这个资产的预期收益应该是

$$R_F + \beta\lambda = 0.05 + 1.1(0.15 - 0.05) = 0.16$$

这个和资产未来一年的预期回报是0.3不相符。所以如果我们购买100元的此资产，卖空110元的市场组合，买10元的无风险资产，这样我们没有任何投入而且没有承受任何市场风险，但是我们的预期收入是

$$0.3 \cdot 100 - 0.15 \cdot 110 + 10 \cdot 0.05 = 14$$

APT模型假设所以的机会被人们发现并利用，直到价格发生变化导致套利机会消失，也就是说市场上的套利者有足够的资金和耐心。这也给使用经验因子模型提供了理由：如果模型结果是不一致的，就存在套利机会，价格也会随之变化。

我们需要多少个因子？¶

因子的数量越多会对收益解释的越多，但是也会对数据中噪音适应的越多。如果想要发现一个好的信号，就需要参数越少越好，但是同时还要能解释收益里面大部分的变动。

下面我们就来计算一下资产的预期收益。

首先导入必要的包

In [1]:

```
import numpy as np
import pandas as pd
from statsmodels import regression
import matplotlib.pyplot as plt
import datetime
```

获取股票、市场及无风险资产（这里的无风险利率是每日中国固定利率国债收益率曲线上国债的年化到期收益率的spot rate）一年的数据。

In [2]:

```
start_date = '2015-01-01'
end_date = '2016-01-01'

# 我们会看标的资产未来一个月的收益去构建未来收益模型
offset_start_date = '2015-02-01'
offset_end_date = '2016-02-01'

# 获得资产收益
asset1 = get_price('601857.XSHG', start_date=offset_start_date, end_date=offset_end_date, fields='ClosingPx').pct_change()[1:]
# 获得市场收益
benchmark = get_price('000300.XSHG', start_date=start_date, end_date=end_date, fields='ClosingPx').pct_change()[1:]
date = benchmark.index
temp = pd.read_csv("2015.csv")
rf = temp['rate']
rf.index = pd.to_datetime(temp['Date'], format="%Y%m%d")
rf = rf[rf.index >= date[0]]
rf = rf[rf.index <= date[-1]]
# 转换成每日的收益率
rf = (1+rf/100)**(1/365)-1
```

需要定义一个常量用来计算截距。

In [3]:

```
constant = pd.TimeSeries(1, index = asset1.index)
df = pd.DataFrame({'A1': asset1, 'HS300': benchmark, 'RF': rf, 'Constant': constant})
df = df.dropna()
df
```

Out[3]:

	A1	Constant	HS300	RF
2015-02-03	0.041889	1	0.024891	0.000064
2015-02-04	-0.023952	1	-0.010379	0.000064
2015-02-05	-0.022787	1	-0.010236	0.000062
2015-02-06	-0.023318	1	-0.016194	0.000061
2015-02-09	0.012856	1	0.010114	0.000060
2015-02-10	0.018132	1	0.018238	0.000061
2015-02-11	-0.006233	1	0.007978	0.000063

	A1	Constant	HS300	RF
2015-02-12	0.001792	1	0.002548	0.000067
2015-02-13	0.000000	1	0.007829	0.000068
2015-02-16	-0.009839	1	0.008547	0.000070
2015-02-17	0.000903	1	0.006526	0.000076
2015-02-25	0.013538	1	-0.012376	0.000074
2015-02-26	0.009795	1	0.025172	0.000073
2015-02-27	0.007055	1	0.001836	0.000074
2015-03-02	-0.008757	1	0.007955	0.000074
2015-03-03	-0.034452	1	-0.025926	0.000074
2015-03-04	-0.000915	1	0.006534	0.000074
2015-03-05	-0.019231	1	-0.009765	0.000073
2015-03-06	0.005602	1	-0.005098	0.000073
2015-03-09	0.011142	1	0.017027	0.000072
2015-03-10	-0.011019	1	-0.004845	0.000071
2015-03-11	-0.001857	1	0.001148	0.000071
2015-03-12	0.026047	1	0.019347	0.000074
2015-03-13	0.002720	1	0.006906	0.000074
2015-03-16	0.012658	1	0.024329	0.000073
2015-03-17	0.025000	1	0.013884	0.000072
2015-03-18	0.014808	1	0.023671	0.000072
2015-03-19	-0.000858	1	-0.001642	0.000071
2015-03-20	0.003436	1	0.013760	0.000070
2015-03-23	0.010274	1	0.020420	0.000068
...	...	...	...	...
2015-11-20	-0.005531	1	-0.000155	0.000043
2015-11-23	-0.007786	1	-0.005577	0.000043
2015-11-24	-0.001121	1	0.000147	0.000043
2015-11-25	0.007856	1	0.007384	0.000043
2015-11-26	-0.001114	1	-0.005865	0.000043
2015-11-27	-0.056856	1	-0.053848	0.000043
2015-11-30	-0.007092	1	0.002648	0.000043
2015-12-01	0.001190	1	0.007089	0.000043
2015-12-02	0.027348	1	0.036267	0.000043
2015-12-03	0.020833	1	0.007347	0.000044
2015-12-04	-0.027211	1	-0.019125	0.000045
2015-12-07	-0.003497	1	0.002723	0.000045
2015-12-08	-0.019883	1	-0.017515	0.000045
2015-12-09	-0.003580	1	0.003566	0.000045
2015-12-10	-0.002395	1	-0.003535	0.000045
2015-12-11	-0.003601	1	-0.004147	0.000045
2015-12-14	0.015663	1	0.028620	0.000045
2015-12-15	-0.007117	1	-0.004563	0.000045
2015-12-16	0.010753	1	-0.002422	0.000045
2015-12-17	0.002364	1	0.019115	0.000045
2015-12-18	0.000000	1	0.003202	0.000045

	A1	Constant	HS300	RF
2015-12-21	0.005896	1	0.026023	0.000046
2015-12-22	-0.001172	1	0.002785	0.000047
2015-12-23	0.002347	1	-0.002670	0.000048
2015-12-24	-0.007026	1	-0.009564	0.000048
2015-12-25	0.000000	1	0.002298	0.000048
2015-12-28	-0.012972	1	-0.028808	0.000048
2015-12-29	0.005974	1	0.009186	0.000048
2015-12-30	-0.001188	1	0.000877	0.000048
2015-12-31	-0.007134	1	-0.009075	0.000048

223 rows × 4 columns

OLS的回归结果。

In [4]:

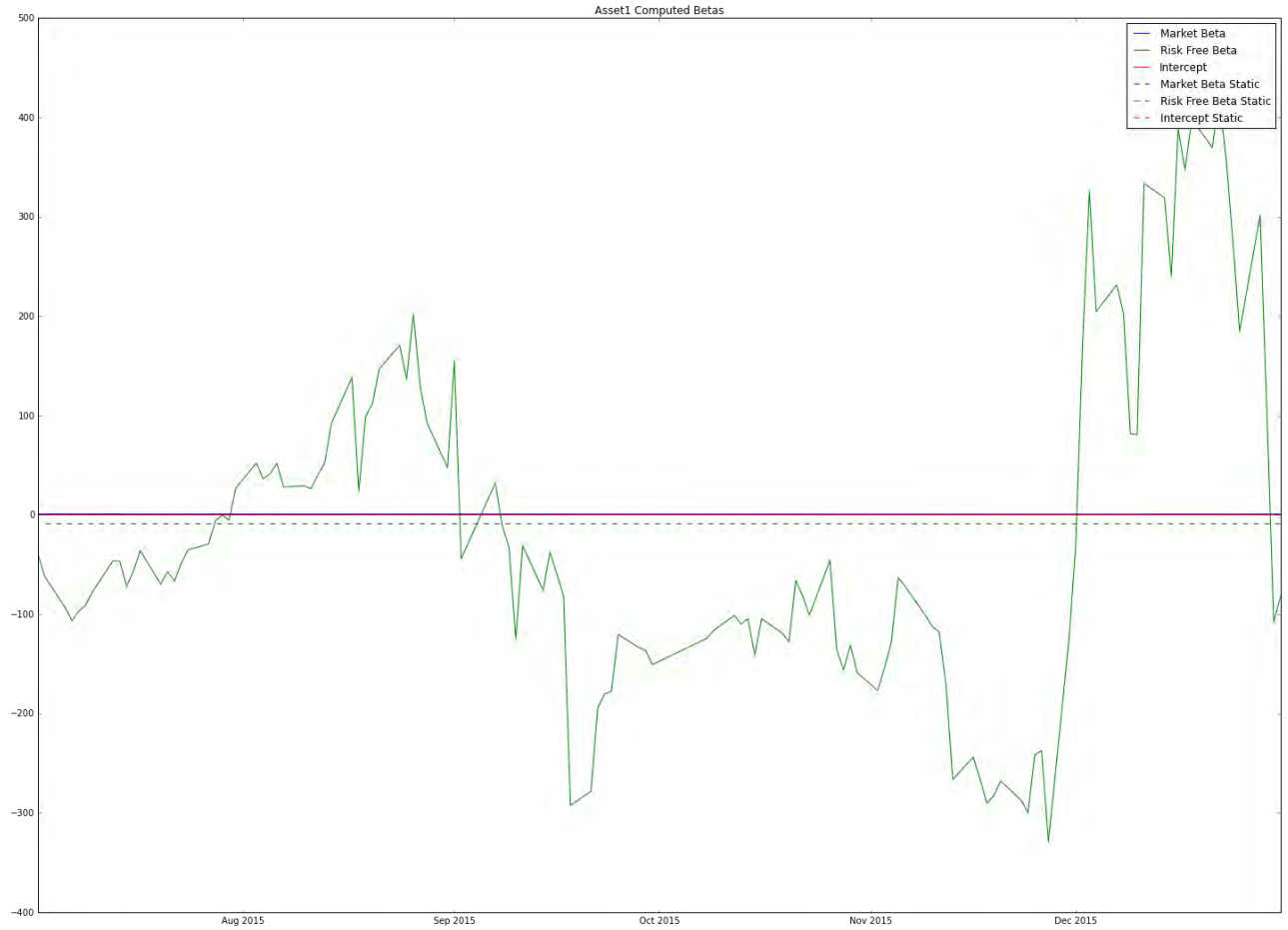
```
OLS = regression.linear_model.OLS(df['A1'], df[['HS300', 'RF', 'Constant']])
fitted_model = OLS.fit()
print('p-value', fitted_model.f_pvalue)
print(fitted_model.params)
A1_params = fitted_model.params
```

```
p-value 8.17578293389e-22
HS300       0.689253
RF         -8.686192
Constant   -0.001100
dtype: float64
```

仅仅通过这个结果并不能说明什么问题，我们需要看一看预测系数的分布以及其是否稳定。我们来看一下一个滑动的100天的回归结果。

In [5]:

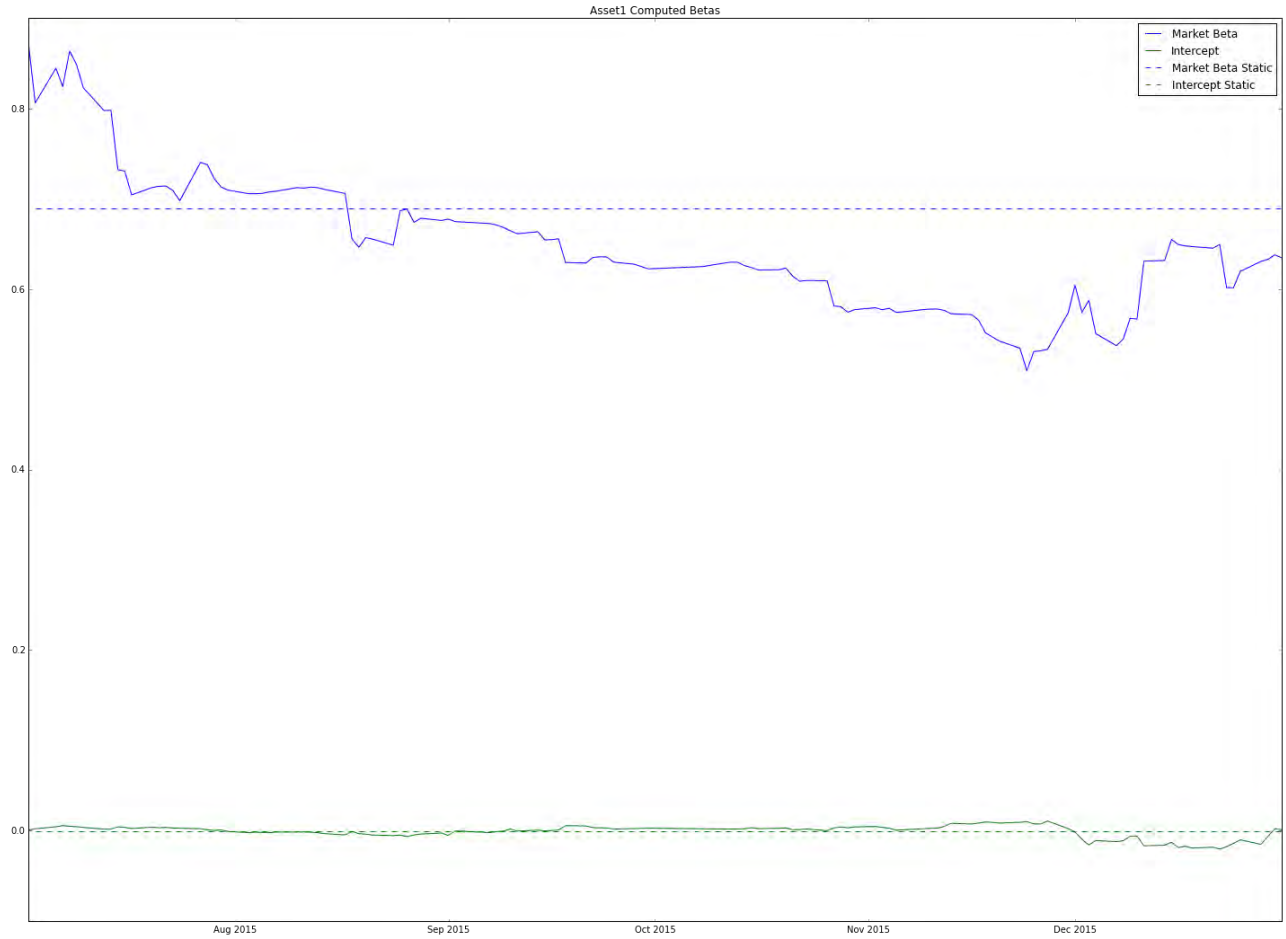
```
model = pd.stats.ols.MovingOLS(y = df['A1'], x=df[['HS300', 'RF']],
                               window_type='rolling',
                               window=100)
rolling = model.beta

plt.figure(figsize=(25, 18))
plt.plot_date(rolling.index,rolling,'-')
plt.hlines(A1_params['HS300'], df.index[100], df.index[-1], linestyles='dashed', colors='blue')
plt.hlines(A1_params['RF'], df.index[100], df.index[-1], linestyles='dashed', colors='green')
plt.hlines(A1_params['Constant'], df.index[100], df.index[-1], linestyles='dashed', colors='red')

plt.title('Asset1 Computed Betas');
plt.legend(['Market Beta', 'Risk Free Beta', 'Intercept', 'Market Beta Static', 'Risk Free Beta Static', 'Intercept Static']);
```



由于无风险收益的β波动范围很大，导致我们无法观测到市场β是否稳定，下面我们在图中去掉无风险收益的β来看一下。

In [6]:

```
rolling = rolling.drop('RF', 1)
plt.figure(figsize=(25, 18))
plt.plot_date(rolling.index,rolling,'-')
plt.hlines(A1_params['HS300'], df.index[100], df.index[-1], linestyles='dashed', colors='blue')
plt.hlines(A1_params['Constant'], df.index[100], df.index[-1], linestyles='dashed', colors='green')

plt.title('Asset1 Computed Betas');
plt.legend(['Market Beta', 'Intercept', 'Market Beta Static', 'Intercept Static']);
```



可以发现市场β还是有一定的波动的。

预测未来走势¶

下面我们使用APT来预测一下股票未来的走势。

In [7]:

```
start_date = '2015-01-01'
end_date = '2015-11-01'

# 我们会看标的资产未来一个月的收益去构建未来收益模型
offset_start_date = '2015-02-01'
offset_end_date = '2015-12-01'

# 获得资产收益
asset1 = get_price('000030.XSHG', start_date=offset_start_date, end_date=offset_end_date, fields='ClosingPx').pct_change()[1:]
# 获得市场收益
benchmark = get_price('000300.XSHG', start_date=start_date, end_date=end_date, fields='ClosingPx').pct_change()[1:]
date = benchmark.index
temp = pd.read_csv("2015.csv")
rf = temp['rate']
rf.index = pd.to_datetime(temp['Date'], format="%Y%m%d")
rf = rf[rf.index >= date[0]]
rf = rf[rf.index <= date[-1]]
# 转换成每日的收益率
rf = (1+rf/100)**(1/365)-1
constant = pd.TimeSeries(1, index=date)
df = pd.DataFrame({'A1': asset1, 'HS300': benchmark, 'RF': rf, 'Constant': constant})
df = df.dropna()
```

In [8]:

```
OLS = regression.linear_model.OLS(df['A1'], df[['HS300', 'RF', 'Constant']])
fitted_model = OLS.fit()
print('p-value', fitted_model.f_pvalue)
print(fitted_model.params)

b_HS300 = fitted_model.params['HS300']
b_RF = fitted_model.params['RF']
a = fitted_model.params['Constant']
```

```
p-value 7.97319026658e-150
HS300       1.058529
RF          3.633765
Constant   -0.000448
dtype: float64
```

获取过去一个月因子数据用来预测未来一个月的数据。

In [9]:

```
start_date = '2015-11-01'
end_date = '2015-12-01'

# 获得市场收益
last_benchmark = get_price('000300.XSHG', start_date=start_date, end_date=end_date, fields='ClosingPx').pct_change()[1:]
date = last_benchmark.index
temp = pd.read_csv("2015.csv")
last_rf = temp['rate']
last_rf.index = pd.to_datetime(temp['Date'], format="%Y%m%d")
last_rf = last_rf[last_rf.index >= date[0]]
last_rf = last_rf[last_rf.index <= date[-1]]
# 转换成每日的收益率
last_rf = (1+last_rf/100)**(1/365)-1
```

预测未来收益。

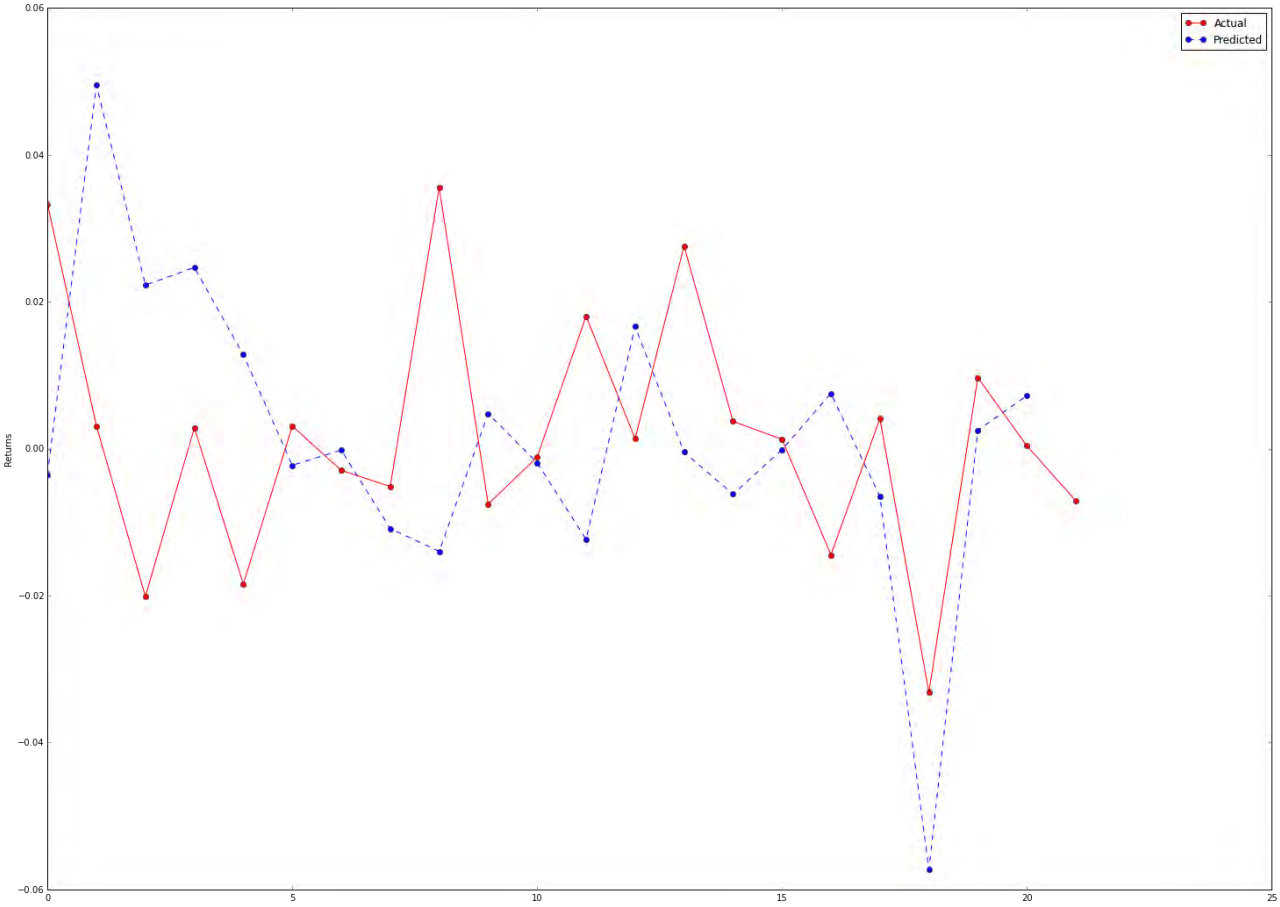
In [10]:

```
predictions = b_HS300 * last_benchmark + b_RF * last_rf + a
```

预测结果和实际对比见下图:

In [11]:

```
offset_start_date = '2015-12-01'
offset_end_date = '2016-01-01'
asset1_act = get_price('000030.XSHG', start_date=offset_start_date, end_date=offset_end_date, fields='ClosingPx').pct_change()[1:]
plt.figure(figsize=(25, 18))
plt.plot(asset1_act.values, 'ro-')
plt.plot(predictions, 'bo--')
plt.ylabel('Returns')
plt.legend(['Actual', 'Predicted']);
```



沪深300指数的风格因子暴露度分析

作者：江嘉键

当你老了，头白了，睡意昏沉，

炉火旁打盹，请取下这部诗歌，

慢慢读，回想你过去眼神的柔和，

回想它们昔日浓重的阴影；

多少人爱你青春欢畅的时辰，

爱慕你的美丽，假意或真心，

只有一个人爱你那朝圣者的灵魂，

爱你衰老了的脸上痛苦的皱纹；

垂下头来，在红光闪耀的炉子旁，

凄然地轻轻诉说那爱情的消逝，

在头顶的山上它缓缓踱着步子，

在一群星星中间隐藏着脸庞。

——《当你老了》

叶芝 /诗 袁可嘉 /译

1 概述

Barra 结构化风险模型是全球知名的投资组合表现和风险分析工具。最近一段时间，我们米筐科技量化策略研究团队对该模型进行了系统研究，并在米筐科技公司的策略研究平台上进行了实现。接下来一段时间，我们将以系列专题的形式展示我们的研究成果。在这一份报告里，我们将对 Barra 结构化模型作简单介绍，并对因子的构建及暴露度的计算进行探讨。为了对因子的有效性作简单的测试，我们对沪深 300 组合 从 2014 年 5 月到 2016 年 3 月共 23 期的因子暴露度进行了计算和分析。

2 均值-方差模型（Markowitz Mean-V ariance Model ）

1952 年，马柯维茨发表了《证券组合选择》，建立起现代投资组合理论的框架。马科维茨认为，投资者可以用预期收益率 $E[R_p]$，以及收益率的标准差 σ_p 来完全构建和衡量一个投资组合，因此，该模型又称为均值-方差模型。依据他的观点，对于一个资产数目为 N ，且各资产头寸相同的投资组合，如果已知每一个资产收益率的方差 $Var(R_p)$ 和资产两两之间的协方差 $Cov(R_iR_j)$ ，则我们可以计算这个投资组合的方差：

$$Var(R_p) = Var(R_1 + R_2 + \cdots + R_N) = \sum_{i=1}^{N} Var(R_i) + \sum_{i=j=1}^{N} Cov(R_iR_j)$$

对于包含 N 个资产的投资组合，我们需要计算 $N(N-1)$ 个协方差，通常以协方差矩阵来表示：

$$\begin{pmatrix} Var(R_1) & Cov(R_1R_2) & \cdots & Cov(R_1R_N) \\ Cov(R_2R_1) & Var(R_2) & \cdots & Cov(R_2R_N) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(R_NR_1) & Cov(R_NR_2) & \ldots & Var(R_N) \end{pmatrix}$$

此时，资产的协方差矩阵包含了我们投资组合的一切风险信息。在实际计算中，我们需要通过历史数据来计算经验协方差矩阵 (empirical covariance matrix)，作为协方差矩阵的估计。然而，使用经验协方差矩阵存在以下问题：

（1）数据量要求大。要对协方差矩阵实现较为准确的估计，需要保证观测值数目大于矩阵的维数。考虑以沪深 300 的 300 个成分股作为投资组合，以月度数据计算经验协方差矩阵，则需要至少 300/12 = 25 年的数据，因此缺乏现实可行性；

（2）依据历史数据进行协方差估计无法反映投资组合中资产的结构性变化（例如并购）；

（3）大量资产两两之间的协方差计算，容易出现多重比较谬误（multiple comparison fallacy）的问题，因而引起资产之间相关性的错误判断。

（4）历史数据中包含大量的噪音，因此简单使用资产的协方差矩阵进行预测会造成较大的偏差。

2 结构化风险模型 (Structural Risk Model, SRM ）

针对以上用资产的协方差矩阵来衡量投资组合风险所存在的缺陷，国际著名的投资组合表现分析研究机构 MSCI Barra 使用结构化风险模型（也称多因子模型，以下简称 SRM）来衡量投资组合的表现和风险。其核心思想是，我们可以选取一系列公共因子（common factors）和特异因子 （idiosyncratic factors）来描述一个投资组合的风险。常用的公共因子有所属行业，成长性，市盈率等，特异因子则是和公共因子相对的概念，用于解释每个资产的收益率中不能用公共因子解释的部分。基于这个思路，投资组合的收益率 R_p 可以用资产的头寸 W，因子暴露（factor exposure）矩阵 X，因子收益率 R_X 和特异因子（idiosyncratic factors）收益率 μ 的线性组合来表示：

$$R_p = W^T(R_X + \mu)$$

其中，因子暴露 X 又称因子载荷（factor loading），用于衡量因子对投资组合收益率的贡献。当因子为行业因子时，X 为取值只有 0 和 1 的哑变量（dummy variable），表示该资产是否属于该行业；当因子为市盈率等连续变量时，为减少回归建模中残差的异方差性，通常要进行剔除离群值和标准化的处理。

另外，SRM 给出了以下的两个假设：

（1）对于同一个资产 i，因子收益率和特异收益率不存在线性相关，即 $Cov(R_i, \mu_i) = 0$；

（2）对于两个不同的资产 i 和 j，它们的特异收益率也不存在线性相关，即 $Cov(\mu_i, \mu_j) = 0$。

基于这两个假设，我们可以推导出 SRM 的投资组合风险表达式：

$$\sigma_p = \sqrt{W^T[XCov(R_X)X^T + \Delta]W}$$

其中 σ_p 为投资组合收益率的标准差，$Cov(R_X)$ 为因子收益率的协方差矩阵。W 和 Δ 分别为权重向量和特异因子收益率方差矩阵。

上述两个表达式即为结构化风险模型的核心。虽然它们的形式上稍显复杂，但它们的意义是明确的：投资组合的风险可以用因子收益率的协方差矩阵，而非投资组合中资产的收益率的协方差矩阵来描述。从数据处理的角度来看，SRM是一种数据降维技术。因此，它具有数据降维通常的优点：

（1）去除数据中的噪音；

（2）它能够大大减少计算量，因此也降低了出现多重比较谬误的可能性。例如，一个包含 500 个资产的投资组合，如果要构建其相关系数矩阵，则需要计算 500*(500-1)/2 = 124,750 个相关系数，如果选用 50 个因子的相关系数来描述，则只需要计算 50*(50-1)/2 = 1225 个相关系数；

（3）因子的统计量通常比资产的统计量有更好的稳定性，因此基于SRM能给出更精确的长期预测；

（4）因子暴露度的调整可以捕捉资产的结构性变化；

（5）因子本身有清晰的经济学涵义，在对 SRM 的因子进行筛选的过程中，也会加深我们对于投资组合风险来源的认识。

3 公共因子的选择

在一般的 SRM 实现中，因子被分为两大类：行业因子和风格因子。参照国家统计局的行业分类方法，我们选定了 19 个行业因子（表1），以取值为 0 或 1 的哑变量（dummy variable）表示。而风格因子则对应一些选择投资组合常见的主题和标准，包括9个类型：贝塔值（beta），动量（momentum），规模（size），盈利率（earnings yield），波动率（volatility），成长性（growth），价值（value），杠杆（leverage）和流动性（liquidity）。因此在目前的建模中，我们一共使用28个因子。

4 风格因子的实现

行业因子的定义见表1。

表1：依据国家统计局的分类选定的行业因子

1	农、林、牧、渔业	11	房地产业
2	采矿业	12	租赁和商务服务业
3	制造业	13	科学研究和技术服务业
4	电力、热力、燃气及水生产和供应业	14	水利，环境和公共设施管理业
5	建筑业	15	居民服务、修理和其他服务业
6	批发和零售业	16	教育
7	交通运输、仓储和邮政业	17	卫生和社会工作
8	住宿和餐饮业	18	文化、体育和娱乐业
9	信息传输、软件和信息技术服务业	19	综合
10	金融业		

接下来我们将介绍如何构建风格因子。在部分风格因子的构建中，我们使用了多个细分因子（一些资料称其为 atom descriptors）来组成一个因子，以表征该因子不同的特征。例如，在盈利率因子的构建中，我们使用了市盈率，每股经营活动产生的现金流量净额和每股收益（扣除/稀释）三个描述变量。如果在线性回归模型中直接使用这些细分因子，因为它们所属的类型相同，因此可能会导致有多个细分因子所属的因子类型对模型参数估计的影响力过大，且容易引起共线性的问题。对此，我们把属于同一类型的多个因子进行加权组合得到该类型对应的暴露度。而因子的权重可通过对上一期数据进行随机森林（random forest）回归分析获得。

另外我们需要对一些稳定性较差（自相关性较低）的因子（动量和波动率）进行指数加权处理，给予时间较近的交易日数据较大的权重。具体地，我们采用半衰期为30天的指数系数（$\lambda = \sqrt[30]{0.5} = 0.977$），此时在风险暴露度矩阵的估计中，近 30 个交易日的数据将会占一半的权重。我们在 200 个交易日处做截断（更早的交易日的权重非常小，已可忽略不计）。在下面的因子计算中，我们将会统一使用 W_λ 表示归一化指数权重向量。

最后，在风格因子的计算中，除了在 RiceQuant 策略研究平台上调用的变量外，我们还需要定义以下的衍生变量和运算：

衍生变量:

$R_{a,e}$和 $R_{b,e}$ 分别表示股票和基准组合的 200 个交易日的每日超额收益率时间序列；R_a和 R_b 分别表示股票和基准组合中资产的 200 个交易日的每日收益率时间序列；W_λ 表示归一化指数权重；ε_a 是贝塔值计算后得到的残差+截距项。outstanding_shares 表示流通股本，TotalVolumeTraded/outstanding_shares 计算得到的即我们通常说的换手率。

运算:

$Cov(\cdot)$ 表示计算变量的协方差；$Var(\cdot)$表示计算变量的方差；$A \cdot B$ 表示计算向量 A 和向量 B 的内积；$ln(\cdot)$ 表示对变量取对数；$std(\cdot)$ 表示计算变量的标准差；$\mu(\cdot)$ 表示计算变量均值，$max(\cdot)$和$min(\cdot)$ 分别表示取一组数据中的最大值和最小值。

基于以上的讨论，我们的风格因子构建所需的变量见表2，具体计算见表3。

get_price	ClosingPx	
	TotalVolumeTraded	
get_fundamentals	eod_derivative	pe_ratio
		pb_ratio
		market_cap
	financial_indicator	operating_cash_flow_per_share
		adjusted_fully_diluted_earnings_per_share
		inc_revenue
		inc_total_asset
		inc_operating_revenue
		inc_gross_profit
	balance_sheet	total_equity
		total_liabilities
		total_assets

表2：RiceQuant 策略研究平台上调用的变量

贝塔值	$\beta = \frac{Cov(R_{b,e}, R_{a,e})}{Var(R_{b,e})}$
动量	$W_\lambda \cdot ln\,(1 + R_a)$
规模	$size = ln\,(Cap_a)$
盈利率	（1）pe_ratio （2）$operating\_cash\_flow\_per\_share$ （3）$adjusted\_fully\_diluted\_earnings\_per\_share$
波动率	（1）$[W_\lambda \cdot (R_a - \mu(R_a))^2]^{1/2}$ （2）$std(W_\lambda \cdot \varepsilon_{a,e})$ $(\varepsilon_{a,e} = R_{a,e} - \beta \cdot R_{b,e})$ （3）$ln\,[1 + max\,\{Z(i)\}] - ln\,[1 + min\{Z(i)\}]$ $Z(i) = \sum_{j=1}^{20} ln(1 + R_a(j)), i = 1, 2, \cdots, 6$（i表示过去 6 个月的序号）
成长性	（1）$inc\_revenue$ （2）inc_total_asset （3）$inc\_operating\_revenue$ （4）inc_gross_profit
价值	$total\_equity/market\_cap$
杠杆	$(total\_liabilities + total\_assets)/total\_assets$
流动性	（1）$ln\{\sum_{t=1}^{20}[1 + TotalVolumeTraded)/outstanding\_shares]/20\}$ (1 month) （2）$ln\,[\sum_{t=1}^{60}[1 + TotalVolumeTraded)/outstanding\_shares]/60\}$ (2 months) （3）$ln\,[\sum_{t=1}^{120}[1 + TotalVolumeTraded)/outstanding\_shares]/120\}$ (3 months)

表3：风格因子的的定义和计算

5 沪深300的因子暴露度分析

接下来，我们将把沪深300指数和各个风格因子的变化趋势进行比较分析（图1和图2）。在这里，我们首先给出沪深300指数的基本计算公式：

报告期指数=报告期成份股的调整市值/基日成份股的调整市值×1000

其中基日为2004年12月31日。由于可以看出，沪深300指数主要决定于其成分股的市值。基于这个认识，我们对各个风格因子的变化进行分析。

5.1 规模因子，盈利率因子和价值因子

基于风格因子的定义，规模因子的暴露度是直接对市值取对数，而盈利率因子暴露度中的市盈率和市值成正比，因此这两个因子和沪深300的变化趋势基本一致（图1）；而对于价值因子的暴露度，其和市值成反比，因此和沪深300的变化呈相反趋势（图2）。

5.2 贝塔值因子

贝塔值是衡量一个投资组合对基准组合的敏感性的指标。在这里，虽然我们使用的投资组合是沪深300的300个成分股，但贝塔值并不恒等于1。其原因在于，我们进行每一期因子暴露度计算时，都使用月底的成分股权重，和过去 200 个交易日沪深 300 指数成分股的权重稍有不同。在图 1 中，我们可以看到当指数出现大幅震荡时，贝塔值因子暴露度会明显偏离于 1，即依据每期月末的权重构建的沪深 300 成分股组合，和过去 200 个交易日的沪深 300 指数的敏感性显著提高或是降低（图1）。

5.3 动量因子

在动量因子的构建过程中，我们使用了 200 个交易日的加权平均值来展示个股的长期动量，对比沪深 300 指数的和动量因子的变化趋势，可以发现当市场出现大幅震荡时，这个因子暴露度的变化呈现出一定的滞后性。2015年下半年市场出现大幅下跌，但2016年2月动量才从正变为负（图1）。

5.4 波动率因子

波动率因子暴露度的走势和沪深300指数的走势基本一致，较好地反映了市场的波动情况（图1）。

5.5 成长性因子

成长性因子的构建使用了四个变量的同比增长率，和企业的股票市值并没有直接联系。有意思的是，沪深300的成长性因子暴露度出现三个峰值，这三个峰值均出现在市场波动较小的阶段，可能反映出在市场行情较为平淡时，投资者较为倾向于投资具有较好成长性的股票（图2）。

5.6 杠杆因子

杠杆因子的计算公式是：（负债合计+总资产）/总资产，和企业的股票市值没有直接联系，但其暴露度变化和沪深300指数的变化呈现相反的趋势。可能反映在牛市当中，杠杆低的企业的股票更受投资者青睐（图2）。

5.7 流动性因子

流动性因子的走势和沪深300指数的走势基本一致，较好地反映了市场的整体交易情况（图2）。

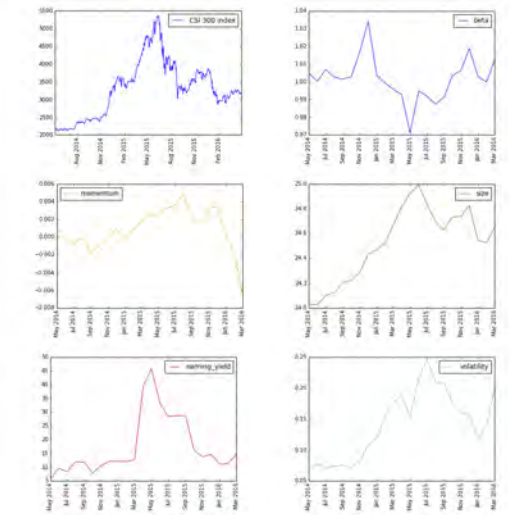


图1 沪深300指数变化和风格因子（贝塔值，动量，规模，盈利率和波动率）的对比

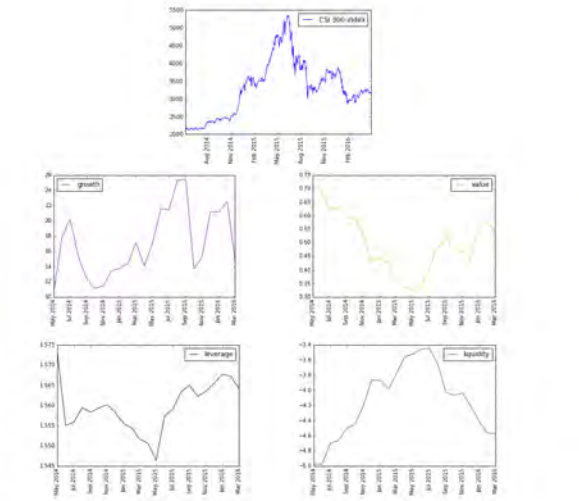


图2 沪深300指数变化和风格因子（成长性，价值，杠杆和流动性）的对比

6 总结

在这一份报告里，我们简要介绍了Barra 结构化风险模型，并对一系列风格因子进行了构建和分析。我们以沪深 300 成分股作为投资组合进行测试。测试结果表明，大部分因子均较好地反映了沪深 300 指数的变化特征，以及在不同的市场环境下投资者的一些投资倾向。在动量因子的分析中，我们发现因子只能反映市场的长期动量变化，而当市场出现大幅震荡的行情时，其变化出现了一定的滞后性。所以在下一步的建模中，我们将进一步添加表征短期和中期动量的细分因子，使其能够捕捉不同情况下的市场变化趋势。

多因子模型水平测试题试答

文章来自江嘉键

因子部分

前一段时间，李腾、陈烨、邓岳、陈志岗几位老师在知乎上发布了一份多因子模型的测试题，其中囊括了多因子建模过程中大部分需要考虑的理论和实践问题：

多因子模型水平测试题 - 科学投资 - 知乎专栏

在过去几个月时间，米筐科技（RiceQuant）量化策略团队对这套经久不衰的量化模型进行了系统研究，相关的产品项目也即将发布。我打算通过回答这份测试题的形式，分享这个过程中获得一些经验和想法。由于题目众多，在这里先对测试题的第一部分的问题进行试答，希望对大家有所帮助。

********** 多因子模型测试题回答开始 **********

问题2： 常见的因子类别?

答：我个人倾向于以下简单的分类方式：量价因子、基本面因子和外部响应因子。

量价因子包含一些简单的指标，例如成交量、换手率和日内最高/最低价等；有一些量价因子则较为复杂，例如一些技术分析因子和数据挖掘得到的统计因子；

基本面因子通常是一些反映企业经营状况（市盈率、负债率等）和经营特征（行业因子）的变量。其包括来自于企业自身公布的报表，也可以是其它专业机构发布的一致预期预测（Consensus Forecast）。

外部响应因子则取决于交易选择的证券池。例如如果一个机构是只做股票类证券的交易，那么和债券市场相关的指标对他们而言就是外部响应因子；如果一个机构是做全球资产配置，涉及多个市场和不同类型的证券，那么像某些国家和地区的通胀率、GDP等，对这个机构来说才是外部响应因子。

问题1： A***股市场驱动因子能分为哪几大类***?

答：因子分类同问题2。

以上三类因子应用于A股市场各自存在优缺点：

量价因子通常对行情变化非常敏感，能够有效捕捉市场的趋势，但简单因子因为广为人知，其影响早已体现在证券价格中（有效市场理论）；一部分技术分析因子由于流传甚广，可能存在一定的自我实现效应，但同样地，其影响可能已体现在证券价格中；而另一部分技术分析的描述接近于玄学，存在一定的模糊性；通过数据挖掘获得的，意义不明的统计因子容易出现多重比较谬误（multiple comparison fallacy）的问题。

基本面因子通常被做长期资产配置或者信奉价值投资理念的投资者采用。而在应用中过程中，存在以下困难：1 基本面因子数据获得时间点不统一（取决于企业发布报表的时间和机构发布预测的时间）；2 相比量价因子，其数据噪声大、缺失值多，数据质量取决于审计机构的专业性和企业的诚信程度；3 因子取值的分布通常远远偏离正态分布，对于离群值的判断不能用标准差这个常用的统计量； 4 一致预期的数据采集难度较大。

外部响应因子在某些情况下有很强的解释力，但在建模过程中，由于不同因子的量纲千差万别，通常要先估计其响应系数，因此会引起变量误差（error-in-variables）的问题，另外，部分外部响应因子来源于政府发布的数据，其可能存在采样质量差，而且数据被人工删改等问题。

问题3：除了最常用的回归法，还有没有其他方法可以进行单因子测试？各自优劣是什么？

答：在不使用回归的前提下，如果我们希望测试单因子对于预期收益率的预测能力：

1 直接计算其信息系数（Information Coefficient，IC），并应用精炼预测公式（refined forecast）进一步预测其效果；

2 构建因子对应的投资组合，其构建方法可以简单也可以复杂，例如选择不同的证券池和不同的中性化处理方法，计算其累计收益率；

3 如果希望捕捉非线性关系，也可以考虑使用距离相关系数（distance correlation）或者最大信息系数（Maximum Information Coefficient）等统计量。但任何非线性关系的分析和应用都存在过拟合的风险。

在不使用回归的前提下，如果我们希望测试该因子是否适合于风险预测模型：

可以计算该因子的自相关系数、和模型其它因子的共线性程度、和基准组合的相关程度等。

问题4：单因子测试是否需要纠正版块、市值偏离等问题？如何纠正？

答：对于收益预测而言，是否需要纠正取决于你对该因子的判断。如果你认为该因子对于行业的预期收益率没有预测能力，就应该实现行业中性，反之不应该实现行业中性；市值偏离是否需要纠正亦是同理。

对于基于净值的业绩归因而言，中性化处理是必须的。市场和市值和影响力很大，不进行中性化处理的话，因子之间的共线性会导致系数估计的方差变大。

问题5：行业归属因子是否应选择动态变化的数据？

答：应该。答案如几位老师所言，不使用动态数据会出现使用未来数据的问题。

问题6：混业经营的上市公司，其行业因子有哪些处理方式？利弊？

答：对于混业经营的上市公司，其行业因子不能简单使用取值为 0 或 1 的哑变量（dummy variable）来表示。此时对行业因子暴露度的处理有两个思路：

1 继续使用哑变量代表行业因子，而虚拟变量的权重可以直接使用等权重，或通过一些财务数据分析来确定；

2 不再使用哑变量，而使用连续变量表示行业因子。其暴露度可以通过该上市公司的收益率对行业收益率做多元回归决定。

第 1 种方法受数据噪音干扰减少，但可能存在欠拟合（underfitting）；第 2 种方法受数据噪音干扰较大，可能存在过拟合的问题。而且选择什么变量进行回归存在主观性。

问题7：行业因子采用GICS、证监会、申万、中信等第三方数据更好？还是利用相关性、聚类分析等算法来动态确定更好？各自利弊？

答：就其使用而言，行业因子一般都是模型中风险控制的维度。它的选择应当符合市场上大部分投资者的认知。我个人的观点是，用复杂统计分析或机器学习来构建行业因子一来没有必要，二来由于引入额外的数据和分析步骤，模型的误差也会随之增加。

问题8：规模因子（Size，也叫市值因子）为什么在中国具有如此重大的影响？选择长期暴露小盘股有哪些利弊？

答：就一般而言，中国A股市场的四个特点导致小盘股更受青睐，市值因子影响巨大：

1 证券市场不规范，内幕交易盛行。部分机构投资者（所谓庄家）和公司股东存在不当的合作关系，利用小盘股股价易于操纵的特点获利；

2 相比成熟的证券市场，中国股票市场个人投资者比例偏高，个人投资者而且对股票市场的认识不成熟，希望通过投资股价变化更快的小盘股短时间内获利；

3 大盘股对应的通常为大型国有企业，其分红意愿不强，投资者难以通过价值投资的方式获得收益；

4 此外，《主动投资组合管理》的一个理论分析和模拟给出了一个很有意思的角度：如果做空个股受限，只能通过基准组合实现对冲的话，阿尔法对冲策略的优化结果通常是在小盘股上持有正头寸，而在大盘股上持有负头寸。这个结论也适用于A股市场。

问题9：市值因子应该怎么取？取市值本身、市值对数、市值平方根有什么区别，哪种更好？你认为流通市值和市值哪个信号更强？

答：取市值平方根计算或市值对数，会减少个股之间市值暴露度的差距。具体哪一种因子计算方式更好，流通市值还是市值哪一个更好，可以通过因子测试验证。就直觉而言，流通市值是一个更能精确地反映股票市值变化的变量，效果可能更好。

问题10：换手率应该怎么计算？如遇长时间停牌，如何处理？

答：换手率一般定义为 交易量/流通股本。由于换手率一般是衡量股票流动性的指标，所以根据多因子模型的调整频率，可以用周换手率、月换手率或年换手率作为指标或者流动性因子。

对于长时间停牌的股票，我个人认可的处理方法是剔除该股票。因为停牌通常意味着有重大消息要发布，其复牌后的收益出现异常波动的可能性较大。这种消息面因素导致的特异收益 (specific return) 通常是无法被多因子模型解释的，所以会给模型的参数估计带来额外噪音。

问题11：若某一因子包含长期平均数据（比如5年平均净利润），而中间有数据缺失的片段（比如最近5年中有2年的年报缺失），应该如何处理？现有两种参考方法：设为空值，或取现有数据的平均值充作长期均值。哪种更好？还是无所谓？

答：作为例子，假设我们在处理一个盈利因子，其第2，3年的数据缺失，考虑三种情况：1 存在缺失值的企业盈利线性增加；存在缺失值的企业盈利线性减少；存在缺失值的股票经营情况来回波动。

如果企业盈利线性增加，剔除缺失值再取平均会导致盈利因子暴露度被高估；

如果企业盈利线性减少，剔除缺失值再取平均会导致盈利因子暴露度被低估；

如果在5年时间区间内企业盈利大幅波动，剔除缺失值再取平均对盈利因子暴露度的暴露度误差的影响无法估计。

对于缺失值的处理没有一劳永逸的方法。其合适的处理方式依赖于数据特点、缺失情况（个别缺失还是大段缺失）、我们计算的目标是什么，等等。因此必须要对具体的问题和数据仔细斟酌，才能找到相对合适的处理方法。

问题12：财务数据应该在哪个时点进行更新？比如月频的多因子模型，年报公布时间可能为3月或4月，是在3月底的时候即时更新那些已出的数据，还是在4月底统一更新使用？

答：在用历史数据对基本面因子取值进行计算的时候，我们应当保证因子所包含的信息和当时市场上投资者获得的信息一致。对于题目中的例子，如果企业是在 3 月份公布年报，则这些股票的因子暴露度应当在 3 月底进行更新；对于没有公布年报的企业，因子暴露度则维持原来的取值。

问题13：有哪些指标可以用来衡量单因子测试的结果？

答：如果希望判断但因子的预测能力，最常见的基本指标是信息率（Information Ratio, IR）。一些业绩评价的常用指标，例如夏普比率，最大回撤等也可以用于单因子效果判断。

问题14：依据单因子测试结果，如何对因子的有效程度进行排序？或者说，如何用单一指标衡量因子有效性？

答：用问题13的指标对因子进行排序则可。在排序的时候，更常见的方法是采用多个股票池对因子进行多次测试 —— 橘生淮南则为橘，生于淮北则为枳。

问题15：所谓的“alpha ***因子***”***和***“***风险因子***”***，应该怎么进行区分？***

答：alpha 因子通常是对特定证券的预期收益率有预测能力的因子；而风险因子通常是投资者认为他们的信息或者策略对于该维度没有预测能力，并希望实现中性化处理的因子 (先验)，或对投资组合的风险存在显著边际贡献的因子 (后验)。由此可见，一个因子是 alpha 因子还是风险因子，部分取决于投资者的信息和经验—— 此之蜜糖，彼之砒霜。

问题16：你理解中有效且有逻辑的因子应该包括哪些？有逻辑但效果较差的因子应该包括哪些？如果采用某种方法组合出一个古怪的因子解释力很强，但是看不出因子的经济意义，你该怎么办？

答：如果我知道有效又有逻辑的因子，那当然不能告诉你了；有逻辑但效果差的话，可能是因为该因子的效应已经体现在证券价格中（有效市场理论）。对于表达式非常复杂难解的因子，可通过计算其对特征组合的暴露度来判断其经济学意义。

问题17***：如何打磨旧的因子，提高其有效性？***

答：一个思路是，把旧因子和新因子进行组合。在旧因子的预测能力未完全衰减至0，且和新因子存在相关性的前提下，旧因子可以增强新因子的预测能力，或者对冲其风险。

问题18***：***构建因子的新信息源如何寻找？有哪些思路？

答：略。

问题19***：现在常用的因子都是易于量化的因子，对于基本面因子、事件驱动因子、市场情绪因子等不易量化的因子，有无合适的处理方法？***

答：在我看来，这些因子不难量化，尽管量化的方式可能因人而异。

问题20***：本专栏的上一篇文章封面人物是谁？***

答：迷糊的乙姬睦美和机智的小蛋。

********** 多因子模型测试题回答结束 **********

小新一

嗯？

你很受欢迎哦。

嗯。。。你说的是步美吧。

不对，是小哀。

呃。。。她不可能吧。

不是哦。如果一个女生用这种眼神看着一个男生，那只有两个可能：1 他脸上黏了饭粒；2 她喜欢这个男生啦。

呃。。。

基于组合权重优化的风格中性多因子选股策略框架

重新发一遍因为刚才图片没有传好。

最近对于多因子都比较感兴趣，之前所做的工作多在于单个因子的测试，但是后来发现这样的工作已经被许多许多人和机构（券商，信息提供商）详尽的做过了，可能更重要的还是如何把这些因子组合起来，形成一个稳定的知道哪里挣钱哪里亏钱的策略，所以今天给大家分享一个风格中性的多因子策略框架，大家可以根据自己的判断加入和删除自己喜欢的因子。接触python和多因子不是很久，希望这个框架可以给大家提供一些帮助。

本模型基于国泰君安在2015.04.06的研报，大部分内容也摘录于这篇研报。本模型并未对因子选择部分进行优化，只是作为一个框架性结构，大家可以自行定义和尝试自己选择的因子和想要中性化的因子。在中性化其他因子之后，我们可以得到这个因子的'pure return'，这个结果可以作为IC，t值等传统检测方法的一个补充。

接下来是一些老生常谈。

多因子选股作为量化投资研究领域的经典模型,在海内外各类投资机构 均受到广泛研究和实践应用。在国内,自 2010 年沪深 300 股指期货上 线以来,以多因子选股为代表的阿尔法对冲策略也逐渐走入了公众的视 野。然而在 2014 年 12 月的市场行情中,阿尔法对冲策略却遭遇了重大 挫折,究其原因不难发现,组合带有过于明显的市值风格特征是导致策 略收益大幅波动的主要原因。

本模型有别于传统的多因子研究,并未将重点放在阿尔法因子的挖掘上,而是着重研究了股票组合的权重优化对策略风格特征的影响并给出了一个自定义因子的回测模型。

在多因子模型中,决定策略收益稳健性的关键步骤正在于股票组合的权重配置。因此,从量化对冲策略追求收益稳定性的角度而言,组合权重优化对多因子模型起着至关重要的作用。

从具体的研究思路而言,我们从结构化多因子风险模型的角度出发,利 用 BARRA 风险因子有效性的检验方法,构建了基于 30 类行业因子、9 类 风格因子的结构化多因子风险模型,奠定了预测股票组合波动率的基 础。之后,我们通过对纯因子股票组合的研究,考察了各类因子阿尔法 性质的强弱,并解释了因子背后的经济、金融学逻辑。最后,我们通过 股票组合的权重优化计算,得到了市值中性、行业中性、风格中性约束 下的最优投资组合。

结构化风险因子模型利用一组共同因子和一个仅与该股票有关的特质因子解释股票的收益率,并利用共同因子和特质因子的波动来解释股票收益率的波动。结构化多因子风险模型的优势在于,通过识别重要的因子,可以降低问题的规模,只要因子个数不变,即使股票组合的数量发生变化,处理问题的复杂度也不会发生变化。

结构化多因子风险模型首先对收益率进行简单的线性分解,分解方程中 包含四个组成部分:股票收益率、因子暴露、因子收益率和特质因子收 益率。那么,第 j 只股票的线性分解如下所示:

$$r_j = x_1 f_1 + x_2 f_2 + x_3 f_3 + x_4 f_4 \ldots x_K f_K + u_j$$

那么对于一个包含 N 只股票的投资组合,假设组合的权重为（w1 , w2 ,..., wN)T ,那么组合收益率可以表示为:

$$R_P = \sum_{j=1}^{N} w_n \cdot (\sum_{k=1}^{K} x_{jk} f_{jk} + u_j)$$

波动率为:

$$\sigma_P = \sqrt{w^T (XFX^T + \Delta) w}$$

X作为因子载荷矩阵，k个因子n个股票的载荷矩阵可以表示如下:

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,k} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,k} \\ \ldots & \ldots & \ldots & \ldots \\ x_{n,1} & x_{n,2} & \ldots & x_{n,k} \end{bmatrix}$$

因子收益率的协方差矩阵可表示如下:

$$F = \begin{bmatrix} Var(f_1) & Cov(f_1, f_2) & \ldots & Cov(f_1, f_k) \\ Cov(f_1, f_2) & Var(f_2) & \ldots & Cov(f_2, f_k) \\ \ldots & \ldots & \ldots & \ldots \\ Cov(f_k, f_1) & Cov(f_k, f_2) & \ldots & Var(f_k) \end{bmatrix}$$

delta为股票的特异波动率。

最大化经风险调整后的收益为目标函数,同时考虑了预期收益与预期风 险的作用,并且在马克维茨的均值方差理论框架下,引入了风险厌恶系数lambda,具体权重优化表达为:

$$\Delta = \begin{bmatrix} Var(u_1) & 0 & \ldots & 0 \\ 0 & Var(u_2) & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & Var(u_k) \end{bmatrix}$$

$$
\begin{aligned}
Max \quad & R_p - \lambda \sigma_p^2 - TC(w) \\
s.t. \quad & \forall k' \ (w^T - w_{bench}{}^T) X_{k'} = 0 \\
& w^T H = h^T \\
& w \geq 0 \\
& \sum_{i=1}^{N} w_i = 1
\end{aligned}
$$

其中 $TC(w)$ 表示以权重 w 构建组合的换仓成本，

$R_p = \sum_{j=1}^{N} w_n \cdot (\sum_{k=1}^{K} x_{jk} f_{jk} + u_j)$，$\sigma_p = \sqrt{w^T (XFX^T + \Delta) w}$。

使用该策略时，大家只需更改以下几处即可：

1. 第339行的index。

2. 第342行的因子名称，当然，你需要在上面定义你的因子的函数。

3. 第426行，你想让哪些因子与对冲所用的指数因子载荷一样。

夹带一点私活，最近也在网上爬一些有趣的东西，刚写了一个小爬虫每日爬取Yahoo finance上的分析师预期，看看能不能产生alpha。欢迎大家来star或者fork：

https://github.com/SunJiaxuan/Web-Crawlers/tree/master/YahooAnalystExpecation

https://www.ricequant.com/community/topic/2153/

阿里巴巴与101个alpha

根据worldquant 发表的论文《101 Formulaic Alphas 》，其中公式化地给出了101个alpha 因子。论文地址在这：https://arxiv .org/pdf/1601.00991.pdf 他们根据数据挖掘的方法发掘了101个alpha，据说里面80%的因子仍然还行之有效并运行在他们的production 中。

对于worldquant 的websim 回测系统进行了模拟，可改变alpha 表达式自行编写expression 形式的alpha 进行回测。如：alpha=1/close.

对其中的函数进行如下解释：

- 可分为横截面函数和时间序列函数两大类，其中时间序列函数名多为以`ts_`开头
- 大部分函数命名方式较为直观
- `abs(x) log(x)`分别表示x的绝对值和x的自然对数
- `rank(x)`表示某股票x值在横截面上的升序排名序号，并将排名归一到[0,1]的闭区间
- `delay(x,d)`表示x值在d天前的值
- `delta(x,d)`表示x值的最新值减去x值在d天前的值
- `correlation(x,y,d) covariance(x,y,d)`分别表示x和y在长度为d的时间窗口上的Pearson相关系数和协方差
- `ts_min(x,d) ts_max(x,d) ts_argmax(x,d) ts_argmin(x,d) ts_rank(x) sum(x,d) stddev(x,d)`等均可以通过函数名称了解其作用
- 具体函数详情见Available Operators

alpha值意义：在每个迭代或者说在每一个交易日，通过计算表达式得到的 每一只股票的Alpha 值，可以得出进行买卖的工具（股票）数量。Alpha 不是表示想要花多少钱买卖，而是在当日的资金比例。

Neutralization （中性化）： 中性化是一种将我们的策略按照市场、行业或子行业进行中性化的操作。当设置 Neutralization="market" （中性化=整个市场）时，它会进行以下操作：

```
alpha = alpha - mean(alpha)
```

基本上，它使 Alpha 向量的均值为零，使得市场上的"净"头寸为零。换句话说，多头头寸和空头头寸相互完全抵消，使我们的策略完全市场中性化。

当设置 Neutralization="industry" 或"sub-industry" （中性化=行业或子行业），那么所有在 Alpha 向量中的金融工具将会按照行业或者子行业分成数个小组，然后对每个小组进行中性化。

if i = **Subindustry 中性化：将i这只股票的alpha 值减去子行业alpha 的均值

alpha(i) = alpha(i) – mean((alpha.subindustry(i)))

中性化（Neutralization ）指的是将原始的Alpha 值放入不同组，然后在各组内进行标准化（Normalization ，指用每个数值减去平均值）的一种操作。 一个组可以是整个市场。 或者根据其他分类来定义组，例如行业或者子行业（基于申银万国行业）。 这样做是为了避免收益受所选组的走向影响，而只注重于股票未来的相对收益。 中性化操作之后，整个投资组合处于中性仓位（多头和空头各占一半）。 这种做法可防止投资组合受市场剧烈波动的影响，还能过滤一些虚假信号。

这里有一个简单的例子，假设 Alpha = close （收盘价格）。
假设您的样本空间里面有 5 只股票（A、B、C、D、E），而在某个特定的日期（20100104）它们的收盘价分别如下（以美元计）：

金融工具（股票）： A B C D E

收盘价： 6 5 2 8 4

现在，您想要使用这些收盘价来计算下一个交易日的 Alpha 权重。 您首先要做的就是写出 Alpha 表达式，在这里即 Alpha = close 。因此您首先要有一个构建一个“close” （收盘价）的向量，即(6, 5, 2, 8, 4)。【注：如果您的表达式是 Alpha = 1 / close ，那么您需要构建的向量也必须是“1 / close” ，即(1/6, 1/5, 1/2, 1/8, 1/4) = (0.167, 0.2, 0.5, 0.125, 0.25)。】

现在您有了向量（6，5，2，8，4），但这不并是权重向量。权重的向量需要经过标准化处理到1。因此我们将每个元素除以它们的总和（= 25），得到的元素总和等于 1。
于是得到了新的向量：(6 / 25, 5 / 25, 2 / 25, 8 / 25, 4 / 25) = (0.24, 0.20, 0.08, 0.32, 0.16)。此向量的总和是 1。这就是我们的投资组合。我们将每个元素再乘以资金规模（2000 万），就能得到我们需要在每只股票上投资的金额。

这是我们将 Neutralization （中性化）设定为“None” 得到的结果。不过这种方法会让我们的策略受到市场风险的影响。于是，我们设置 Neutralization = "Market" （中性化 = “ 市场”）。在这种情况下，我们首先再一次构建一个“close” （收盘价）的向量 = (6, 5, 2, 8, 4)。
现在我们使其 "mean-neutral （均值中性化）"，也就是将每个元素减去他们的均值（=5），使得向量的总和为 0。
于是得到了均值中性化的向量：(1, 0, -3, 3, -1)。您会发现此时所有元素的总和为零。

现在，若要对其进行标准化处理，我们忽略数值的正负，使元素总和为 1。首先我们计算元素绝对值的总和（= 1 + 0 + 3 + 3 + 1 = 8）。现在我们的将每个元素除以它们的总和，得到 (1 / 8, 0 / 8, -3 / 8, 3 / 8, -1 / 8) = (+0.125, 0, -0.375, +0.375, -0.125)。
现在，这就是标准化均值中性的权重向量。我们把它乘以资金规模 2000 万美元，就得到了我们在每只股票上需要投资的金额，如果数值为正，则表明对其建立多头头寸，如果数值为负，则表明对其建立空头头寸。此外，因为所有的正值相加等于 0.5，而负值相加等于 -0.5，这表明我们在多头头寸上投资了 1000 万美元，在空头头寸上也投资了 1000 万美元，满足了中性化策略的要求。

https://www.ricequant.com/community/topic/2129/

因子分析¶

In [1]:

```
import pandas as pd
import numpy as np
from scipy.stats import mstats
from scipy import stats
from datetime import datetime
from pandas import Series
from pandas import DataFrame
from alphalens import utils
from alphalens import plotting
from alphalens import tears
from alphalens import performance
import matplotlib as mpl
import matplotlib.pyplot as plt

# mpl.rcParams['font.family'] = 'sans-serif'
# mpl.rcParams['font.sans-serif'] = [u'SimHei']
# plt.rcParams['axes.unicode_minus'] = False
```

0、数据准备¶

In [2]:

```
# 选择日期
trading_dates = get_trading_dates('2017-01-01','2017-08-01')
trading_dates[0]
```

Out[2]:

```
datetime.date(2017, 1, 3)
```

In [3]:

```
# 获得某段时间内hs300的pe因子数据,将其拼接成有multiindex的dataframe，再转成Series
df_facs_datas = DataFrame()

factor_name = 'pe_ratio'
stocks_sets = index_components('000300.XSHG')
q = query(fundamentals.eod_derivative_indicator.pe_ratio).filter(fundamentals.eod_derivative_indicator.stockcode.in_(stocks_sets))
for i in range(len(trading_dates)):
    daily_fac_data = get_fundamentals(q,trading_dates[i])[0,0,:]
```

```
    df_daily_fac_data = DataFrame(daily_fac_data)
    df_daily_fac_data.columns = ['fac_value']

    df_daily_fac_data['date'] = trading_dates[i]
    df_facs_datas = pd.concat([df_facs_datas,df_daily_fac_data])

df_facs_datas = df_facs_datas.set_index(['date',df_facs_datas.index])
series_facs_datas = df_facs_datas['fac_value']
```

In [4]:

```
# 查看前几行数据
series_facs_datas.head()
```

Out[4]:

```
date
2017-01-03  000001.XSHE      6.885
            000002.XSHE     11.719
            000008.XSHE    85.5301
            000009.XSHE    70.3581
            000060.XSHE    94.7214
Name: fac_value, dtype: object
```

In [5]:

```
# 除去异常值和标准化
def winsorize_series(series):
#
    q = series.quantile([0.02,0.98])
    if isinstance(q,pd.Series) and len(q) == 2:
        series[series<q.iloc[0]] = q.iloc[0]
        series[series>q.iloc[1]] = q.iloc[1]
    return series
def standardize_series(series):
    std = series.std()
    mean = series.mean()
    return (series-mean)/std


```

In [6]:

```
# 看一下pe因子的分布情况
series_facs_datas = series_facs_datas.groupby(level = 'date').apply(winsorize_series)
series_facs_datas = series_facs_datas.groupby(level = 'date').apply(standardize_series)
series_facs_datas.hist(figsize=(12,6),bins=20)
# bins指定有多少个柱，可自行调整
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb37c982630>
```

```
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```

In [7]:

```
# 获取收盘价数据：
price = get_price(stocks_sets, start_date='2017-01-01',end_date = '2017-08-01').close
price.index.name = 'date'
price.columns.name = 'code'
# 看一下价格数据的前几行
price.head()
```

Out[7]:

code	002049.XSHE	600023.XSHG	000768.XSHE	600999.XSHG	600703.XSHG	600100.XSHG	600118.XSHG
date							
2017-01-03	33.0699	5.2800	22.0792	16.3240	13.8010	13.7030	31.7795
2017-01-04	32.9901	5.2992	22.1291	16.4724	13.7714	13.7816	31.8193
2017-01-05	32.5309	5.4624	22.5180	16.4329	14.0385	13.7030	32.1081
2017-01-06	32.2015	5.4144	22.3883	16.3141	13.6230	13.5950	32.0484
2017-01-09	32.2814	5.5296	23.5751	16.2943	13.6428	13.7030	33.3431

5 rows × 300 columns

In [8]:

```
# 获取市值数据,按照上面的方式进行拼接..
df_facs_datas_mc = DataFrame()
q = query(fundamentals.eod_derivative_indicator.market_cap).filter(fundamentals.eod_derivative_indicator.stockcode.in_(stocks_sets))
for i in range(len(trading_dates)):
    daily_fac_mc_data = get_fundamentals(q,trading_dates[i])[0,0,:]

    df_daily_fac_mc_data = DataFrame(daily_fac_mc_data)
    df_daily_fac_mc_data.columns = ['market_value']

    df_daily_fac_mc_data['date'] = trading_dates[i]
    df_facs_datas_mc = pd.concat([df_facs_datas_mc,df_daily_fac_mc_data])

df_facs_datas_mc = df_facs_datas_mc.set_index(['date',df_facs_datas_mc.index])
df_facs_datas_mc.index.names= ['date','code']
series_facs_datas_mc = df_facs_datas_mc['market_value']
series_facs_datas_mc.tail()
```

Out[8]:

```
date        code
2017-08-01  601997.XSHG      3.595e+10
            601998.XSHG    3.29821e+11
```

```
            603160.XSHG    4.19326e+10
            603858.XSHG    4.59601e+10
            603993.XSHG    1.52275e+11
Name: market_value, dtype: object
```

In [9]:

```
port = [1,2,3,4,5]
# 分位数处理，进行分组
def division(series):
    q = series.quantile([0.2,0.4,0.6,0.8])
    if isinstance(q,pd.Series) and len(q) == 4:
        series[series<q.iloc[0]] = port[0]
        series[(series>=q.iloc[0]) & (series<q.iloc[1])] = port[1]
        series[(series>=q.iloc[1]) & (series<q.iloc[2])] = port[2]
        series[(series>=q.iloc[2]) & (series<q.iloc[3])] = port[3]
        series[(series>=q.iloc[3])] = port[4]
    return series
```

In [10]:

```
# 将市值因子的数据每天都进行分组
mc_group = series_facs_datas_mc.groupby(level='date').apply(division)
# 标签
mc_label = {1:'very_small_MC',2:'small_MC',3:'mid_MC',4:'big_MC',5:'very_big_MC'}
mc_group.tail()
```

Out[10]:

```
date        code
2017-08-01  601997.XSHG    2
            601998.XSHG    5
            603160.XSHG    3
            603858.XSHG    3
            603993.XSHG    5
Name: market_value, dtype: object
```

下面就进行分析了,一共有¶

（Quantiles Statistics ，Returns Analysis ，Information Analysis ，Turnover Analysis ）¶

整理数据成规定的格式：¶

In [11]:

```
utils.get_clean_factor_and_forward_returns?
```

参数	类型	注释
factor	pd.Series	A MultiIndex Series indexed by date (level 0) and asset (level 1), containing the values for a single alpha factor.
prices	pd.DataFrame	A wide form Pandas DataFrame indexed by date with assets in the columns. It is important to pass the correct pricing data in depending on what time of period your signal was generated so to avoid lookahead bias, or delayed calculations. Pricing data must span the factor analysis time period plus an additional buffer window that is greater than the maximum number of expected periods in the forward returns calculations.
groupby	pd.Series	Either A MultiIndex Series indexed by date and asset, containing the period wise group codes for each asset, or a dict of asset to group mappings. If a dict is passed, it is assumed that group mappings are unchanged for the entire time period of the passed factor data.
by_group	bool	If True, compute statistics separately for each group.
quantiles	int or sequence[float]	Number of equal-sized quantile buckets to use in factor bucketing. Alternately sequence of quantiles, allowing non-equal-sized buckets e.g. [0, .10, .5, .90, 1.] or [.05, .5, .95] Only one of 'quantiles' or 'bins' can be not-None
bins	int or sequence[float]	Number of equal-width (valuewise) bins to use in factor bucketing. Alternately sequence of bin edges allowing for non-uniform bin width e.g. [-4, -2, -0.5, 0, 10] Only one of 'quantiles' or 'bins' can be not-None
periods	sequence[int]	periods to compute forward returns on.
filter_zscore	int or float	Sets forward returns greater than X standard deviations from the the mean to nan. Caution: this outlier filtering incorporates lookahead bias.
groupby_labels	dict	A dictionary keyed by group code with values corresponding to the display name for each group.

Return： A MultiIndex Series indexed by date (level 0) and asset (level 1), containing the values for a single alpha factor, forward returns for each period, The factor quantile/bin that factor value belongs too, and (optionally) the group the asset belongs to.

大致意思就是将上面得到的数据有:'因子数据','价格','市值数据','市值分组的标签'等数据（要符合规格）代入get_clean_factor_and_forward_returns这个函数就可以得到 一个多重索引的dataframe，包含了alpha（在factor那列），每个时期的预期收益(1,5,10)，因子分组的组号（factor_quantile），可能还会有按另一个因子（此处是市值）的分组（group）

In [12]:

```
facs_data_analysis  = utils.get_clean_factor_and_forward_returns(series_facs_datas,price,groupby=mc_group,groupby_labels=mc_label)
facs_data_analysis.head()
```

Out[12]:

		1	5	10	factor	group	factor_quantile
date	asset						
2017-01-03	002049.XSHE	-0.002413	-0.022032	-0.083006	0.332023	very_small_MC	5
	600023.XSHG	0.003636	0.029091	0.040000	-0.66929	big_MC	1
	000768.XSHE	0.002260	0.045618	0.020775	1.13325	big_MC	5
	600999.XSHG	0.009091	-0.000606	0.001213	-0.503345	very_big_MC	2
	600703.XSHG	-0.002145	-0.003579	0.002869	-0.199861	big_MC	4

In [13]:

```
# 由于factor那列是object类型，转换成float方便下面继续分析
facs_data_analysis['factor'] = np.float128(facs_data_analysis['factor'])
```

1、 查看一下summary ¶

In [14]:

```
tears.create_summary_tear_sheet?
```

参数	类型	注释
factor_data	pd.DataFrame	pd.DataFrame - MultiIndex A MultiIndex DataFrame indexed by date (level 0) and asset (level 1), containing the values for a single alpha factor, forward returns for each period, The factor quantile/bin that factor value belongs too, and (optionally) the group the asset belongs to.
long_short	bool	Should this computation happen on a long short portfolio?

factor_data:里面放上面通过utils.get_clean_factor_and_forward_returns的dataframe

返回的是一个简易的summary包含（Quantiles Statistics，Returns Analysis，Information Analysis，Turnover Analysis）

In [15]:

```
tears.create_summary_tear_sheet(facs_data_analysis)
```

```
Quantiles Statistics
```

	min	max	mean	std	count	count %
factor_quantile						
1	-0.958126	-0.447158	-0.732824	0.108024	7597	20.192435
2	-0.702939	-0.329963	-0.525532	0.075143	7512	19.966510
3	-0.459914	-0.115072	-0.304591	0.072588	7475	19.868166
4	-0.224833	0.482232	0.035476	0.164809	7512	19.966510
5	0.084564	5.632394	1.531205	1.294542	7527	20.006379

```
Returns Analysis
```

	1	5	10
Ann. alpha	-0.211	-0.227	-0.230
beta	0.186	0.261	0.371
Mean Period W ise Return T op Quantile (bps)	-12.003	-64.680	-121.262

	1	5	10
Mean Period W ise Return Bottom Quantile (bps)	9.705	52.127	97.083
Mean Period W ise Spread (bps)	-21.634	-23.290	-21.798

```
Information Analysis
```

	1	5	10
IC Mean	-0.046	-0.100	-0.140
IC Std.	0.190	0.207	0.190
t-stat(IC)	-2.768	-5.524	-8.464
p-value(IC)	0.006	0.000	0.000
IC Skew	-0.059	0.092	0.341
IC Kurtosis	-0.467	-0.603	-0.292
Ann. IR	-3.839	-7.662	-11.739

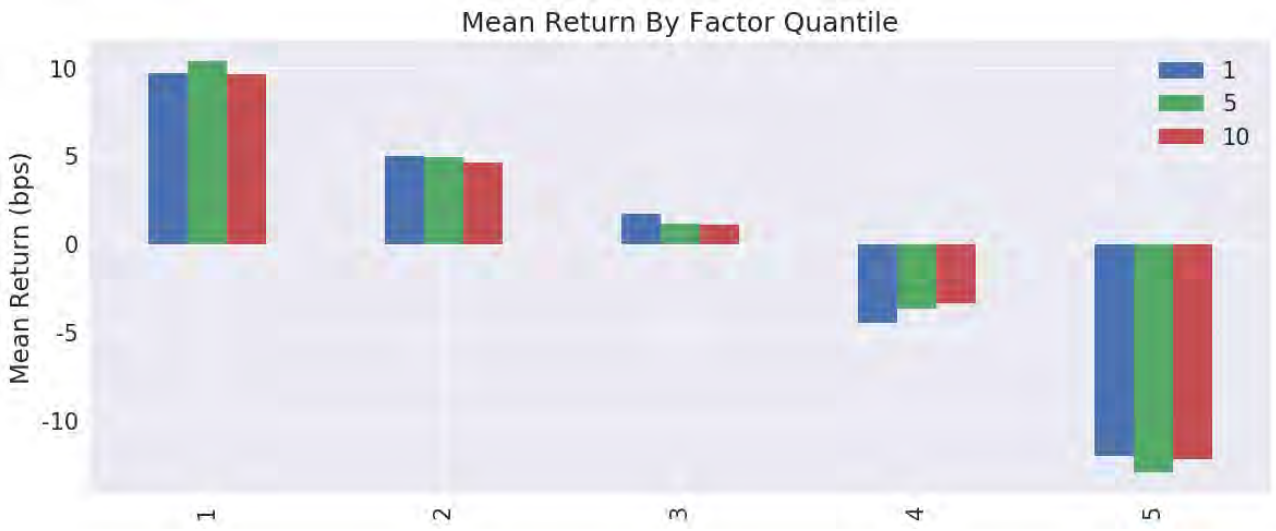
```
Turnover Analysis
```

	1	5	10
Quantile 1 Mean T urnover	0.016	0.038	0.054
Quantile 2 Mean T urnover	0.036	0.085	0.119
Quantile 3 Mean T urnover	0.042	0.103	0.145
Quantile 4 Mean T urnover	0.034	0.086	0.124
Quantile 5 Mean T urnover	0.013	0.037	0.061

	1	5	10
Mean Factor Rank Autocorrelation	0.999	0.995	0.989

```
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```

```
<matplotlib.figure.Figure at 0x7fb37c2d9e48>
```



2、 Returns Analysis ¶

In [16]:

```
tears.create_returns_tear_sheet?
```

参数	类型	注释
factor_data	pd.DataFrame - MultiIndex	A MultiIndex DataFrame indexed by date (level 0) and asset (level 1), containing the values for a single alpha factor, forward returns for each period, The factor quantile/bin that factor value

		belongs too, and (optionally) the group the asset belongs to.
long_short	bool	Should this computation happen on a long short portfolio?
by_group	bool	If True, perform calcuations, and display graphs separately for each group.

收益率分析： 分析每个预期收益在每组的情况、以及每个forward Period每组的累积收益；

factor_data:里面放上面通过utils.get_clean_factor_and_forward_returns的dataframe

by_group:如果是True，会每个组都展示图标

可以对于pe看出第一组应该是好于第五组的 在每个Forward Period的高组减去低组的平均收益 每种市值的收益率分析

In [17]:

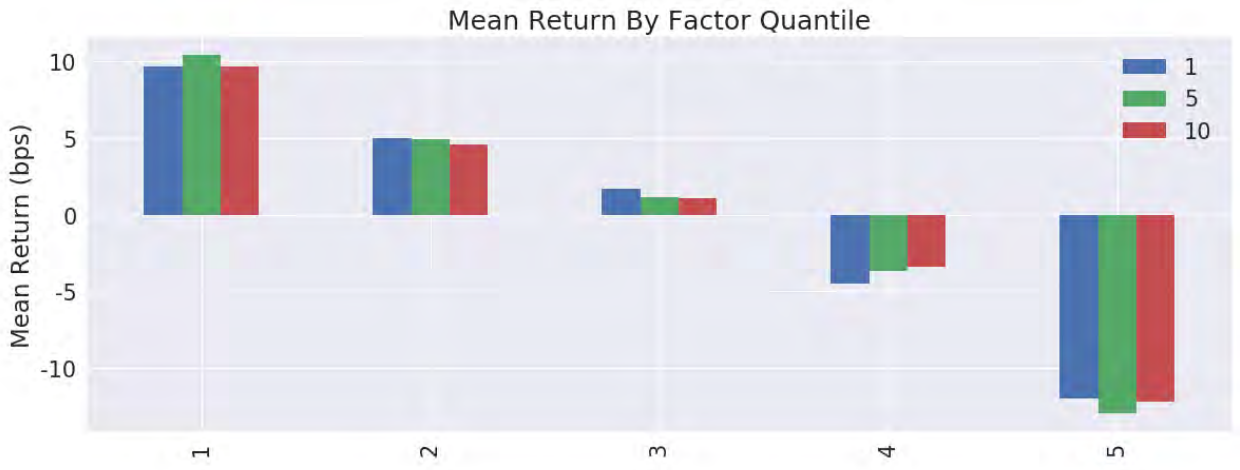
```
tears.create_returns_tear_sheet(facs_data_analysis,by_group=True)
```

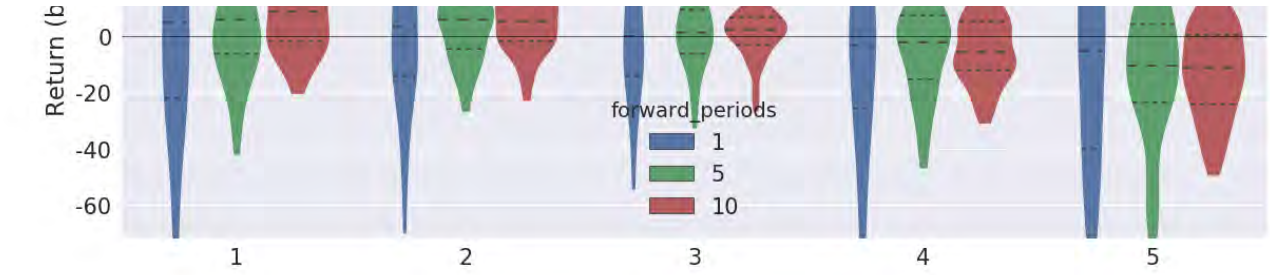
```
Returns Analysis
```

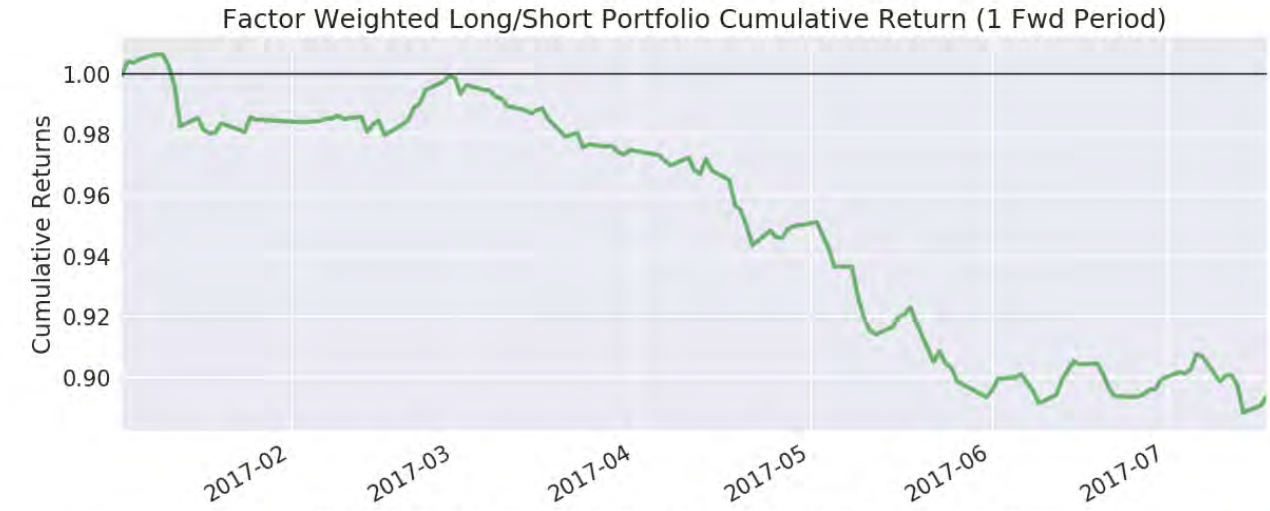
	1	5	10
Ann. alpha	-0.211	-0.227	-0.230
beta	0.186	0.261	0.371
Mean Period W ise Return T op Quantile (bps)	-12.003	-64.680	-121.262
Mean Period W ise Return Bottom Quantile (bps)	9.705	52.127	97.083
Mean Period W ise Spread (bps)	-21.634	-23.290	-21.798

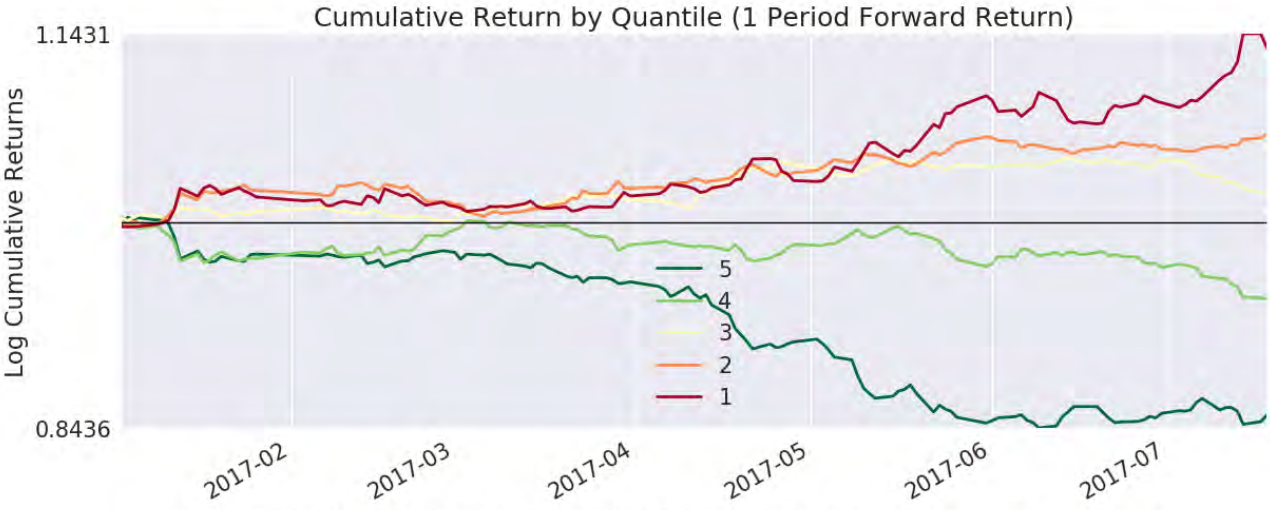
```
/srv/env/lib64/python3.4/site-packages/alphalens/plotting.py:727: FutureWarning: pd.rolling_apply is deprecated for Series and will b
        Series.rolling(min_periods=1,window=5,center=False).apply(func=<function>,kwargs=<dict>,args=<tuple>)
  min_periods=1, args=(period,))
/srv/env/lib64/python3.4/site-packages/alphalens/plotting.py:767: FutureWarning: pd.rolling_apply is deprecated for DataFrame and wil
        DataFrame.rolling(min_periods=1,window=5,center=False).apply(func=<function>,kwargs=<dict>,args=<tuple>)
  min_periods=1, args=(period,))
/srv/env/lib64/python3.4/site-packages/alphalens/plotting.py:727: FutureWarning: pd.rolling_apply is deprecated for Series and will b
        Series.rolling(min_periods=1,window=10,center=False).apply(func=<function>,kwargs=<dict>,args=<tuple>)
  min_periods=1, args=(period,))
/srv/env/lib64/python3.4/site-packages/alphalens/plotting.py:767: FutureWarning: pd.rolling_apply is deprecated for DataFrame and wil
        DataFrame.rolling(min_periods=1,window=10,center=False).apply(func=<function>,kwargs=<dict>,args=<tuple>)
  min_periods=1, args=(period,))
/srv/env/lib64/python3.4/site-packages/alphalens/plotting.py:519: FutureWarning: pd.rolling_mean is deprecated for Series and will be
        Series.rolling(window=22,center=False).mean()
  pd.rolling_mean(mean_returns_spread_bps, 22).plot(color='orangered',
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```

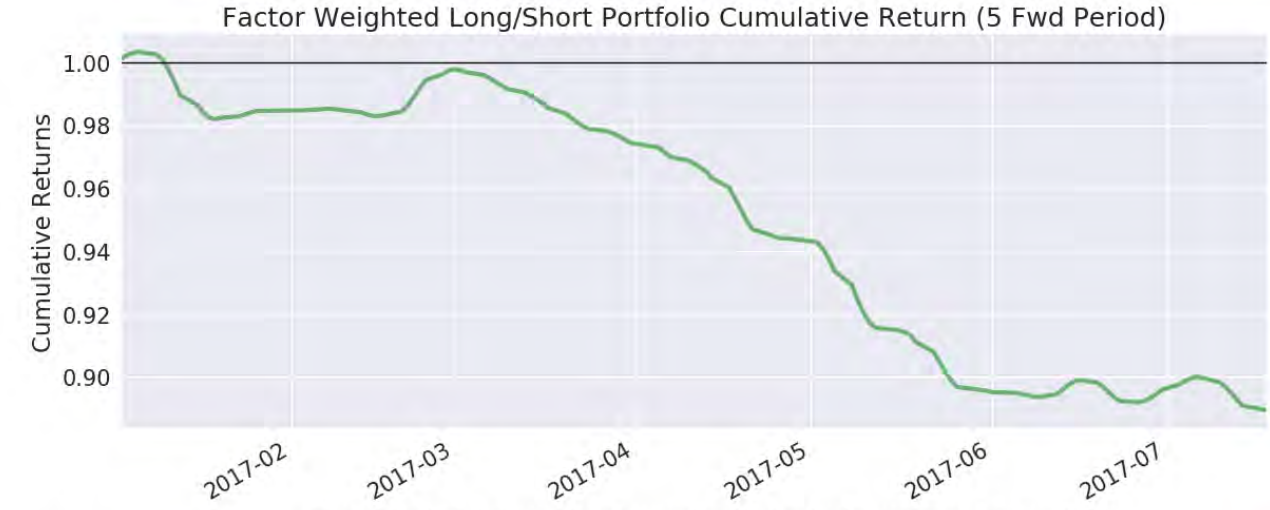
```
<matplotlib.figure.Figure at 0x7fb365e38550>
```

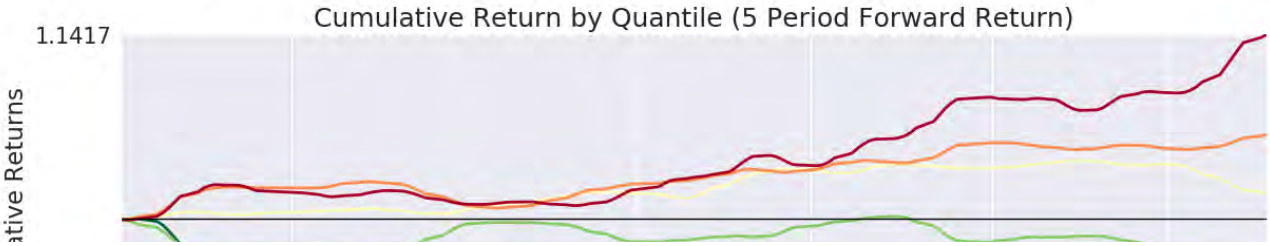


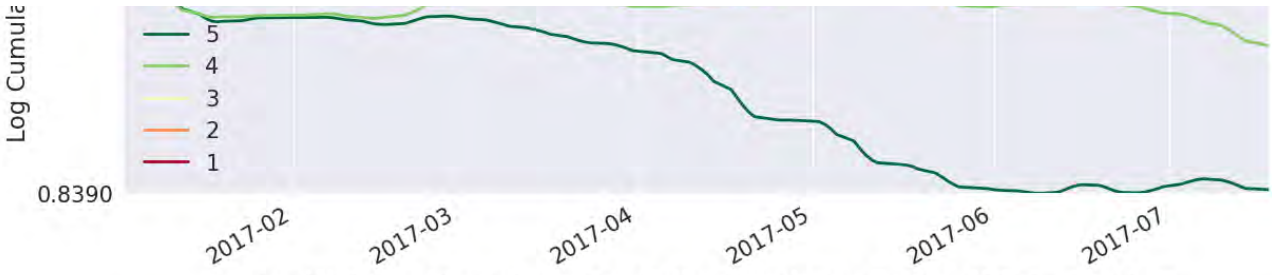

Return (b
0
-20
-40
-60
forward_periods
1
5
10
1
2
3
4
5


Factor Weighted Long/Short Portfolio Cumulative Return (1 Fwd Period)
Cumulative Returns
1.00
0.98
0.96
0.94
0.92
0.90
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07


Cumulative Return by Quantile (1 Period Forward Return)
Log Cumulative Returns
1.1431
0.8436
5
4
3
2
1
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07

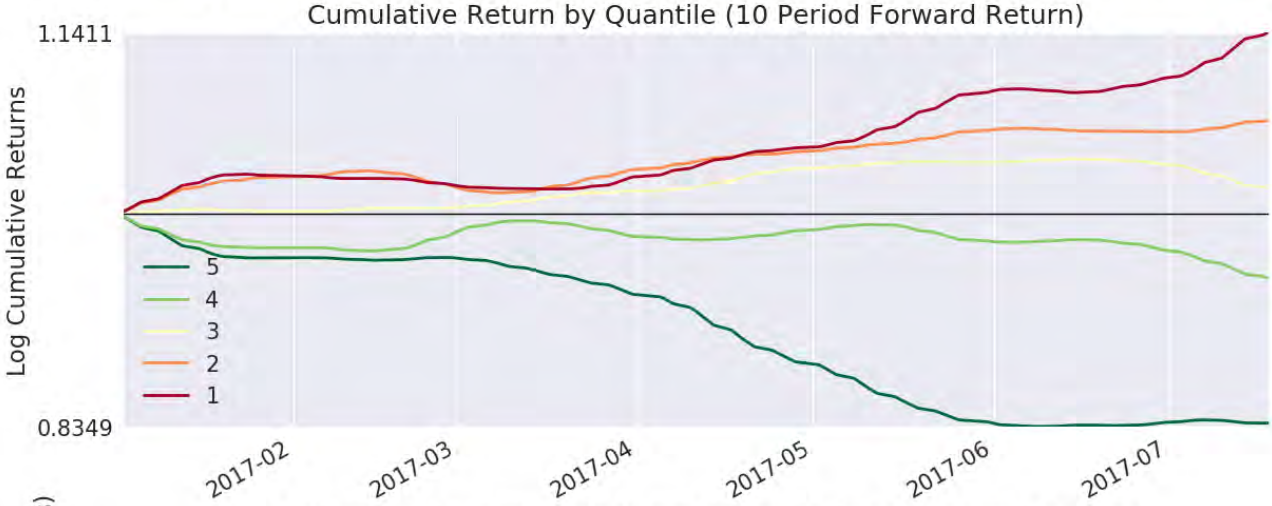

Factor Weighted Long/Short Portfolio Cumulative Return (5 Fwd Period)
Cumulative Returns
1.00
0.98
0.96
0.94
0.92
0.90
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07


Cumulative Return by Quantile (5 Period Forward Return)
1.1417

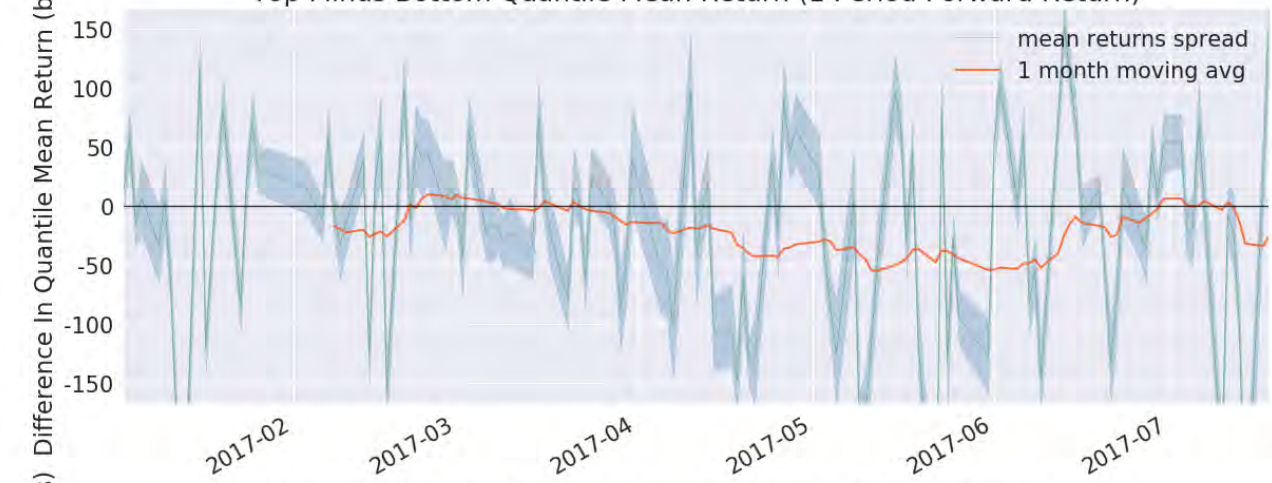


Factor Weighted Long/Short Portfolio Cumulative Return (10 Fwd Period)

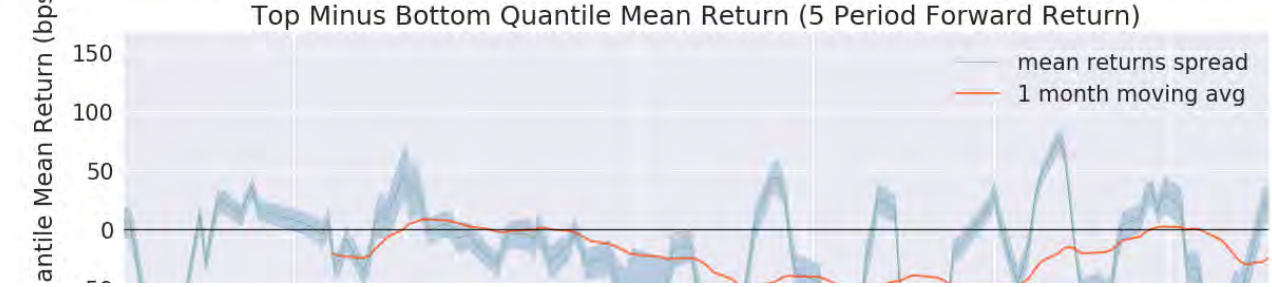
Cumulative Return by Quantile (10 Period Forward Return)

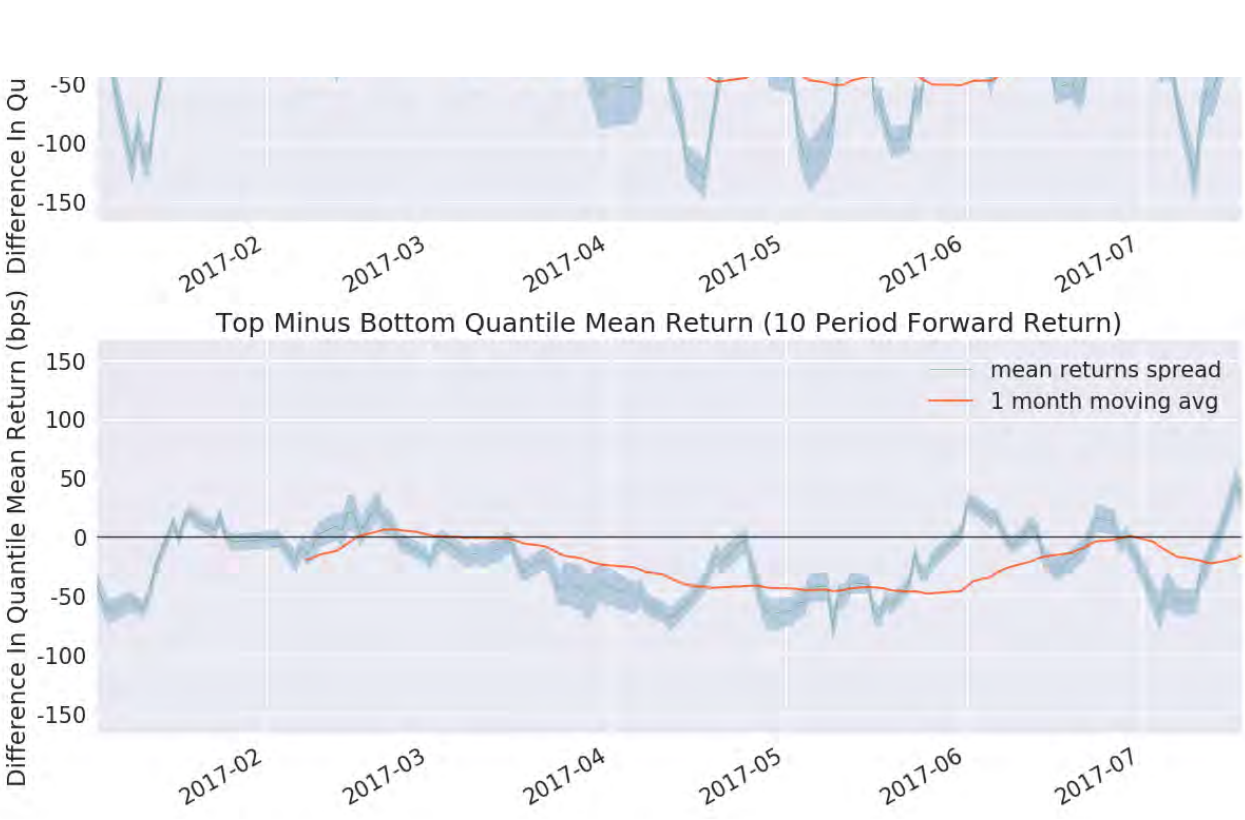


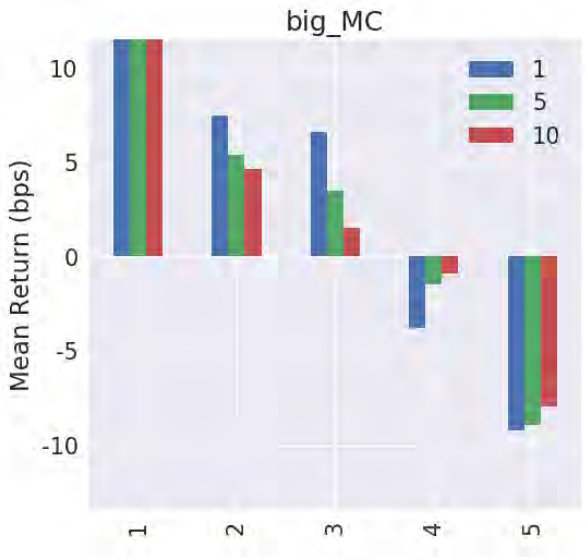
Top Minus Bottom Quantile Mean Return (1 Period Forward Return)

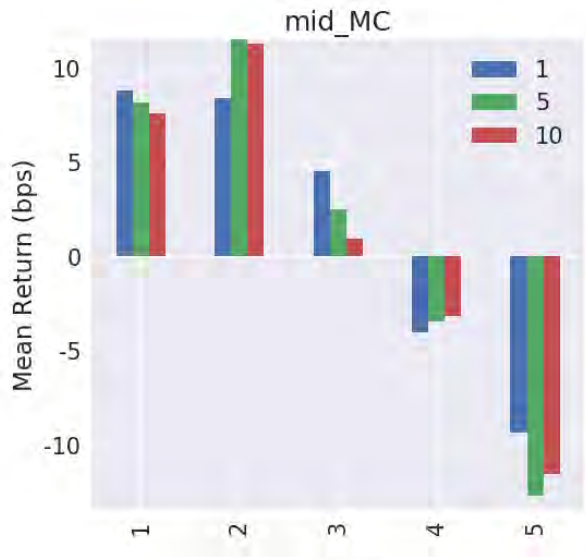


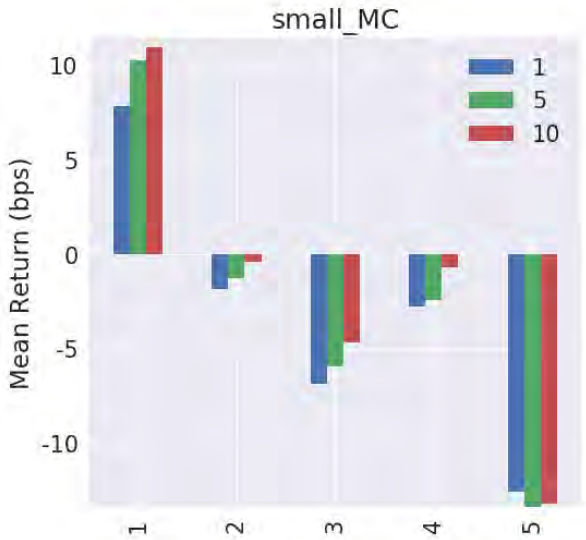
Top Minus Bottom Quantile Mean Return (5 Period Forward Return)

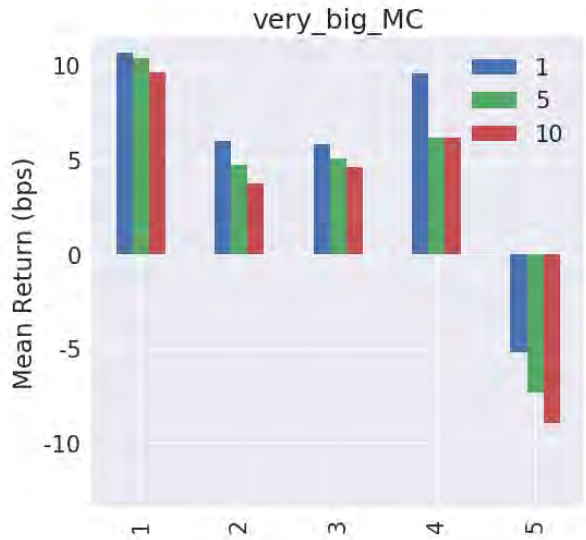


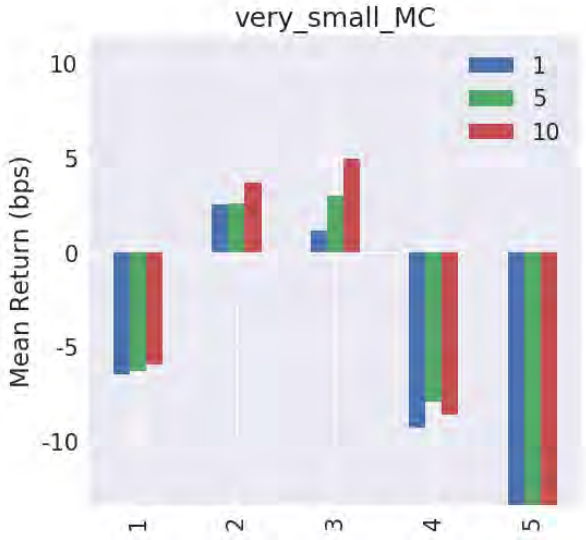

Difference In Qu
-50
-100
-150
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07
Top Minus Bottom Quantile Mean Return (10 Period Forward Return)
Difference In Quantile Mean Return (bps)
150
100
50
0
-50
-100
-150
mean returns spread
1 month moving avg
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07











3、 Turnover Analysis ¶

In [18]:

```
tears.create_turnover_tear_sheet?
```

参数	类型	注释
factor_data	pd.dataFrame	A MultiIndex DataFrame indexed by date (level 0) and asset (level 1), containing the values for a single alpha factor, forward returns for each period, The factor quantile/bin that factor value belongs too, and (optionally) the group the asset belongs to.

换手率分析 每个forward period的每组的平均换手率以及等级相关系数（秩相关）；

factor_data:里面放上面通过utils.get_clean_factor_and_forward_returns的dataframe

In [19]:

```
tears.create_turnover_tear_sheet(facs_data_analysis)
```

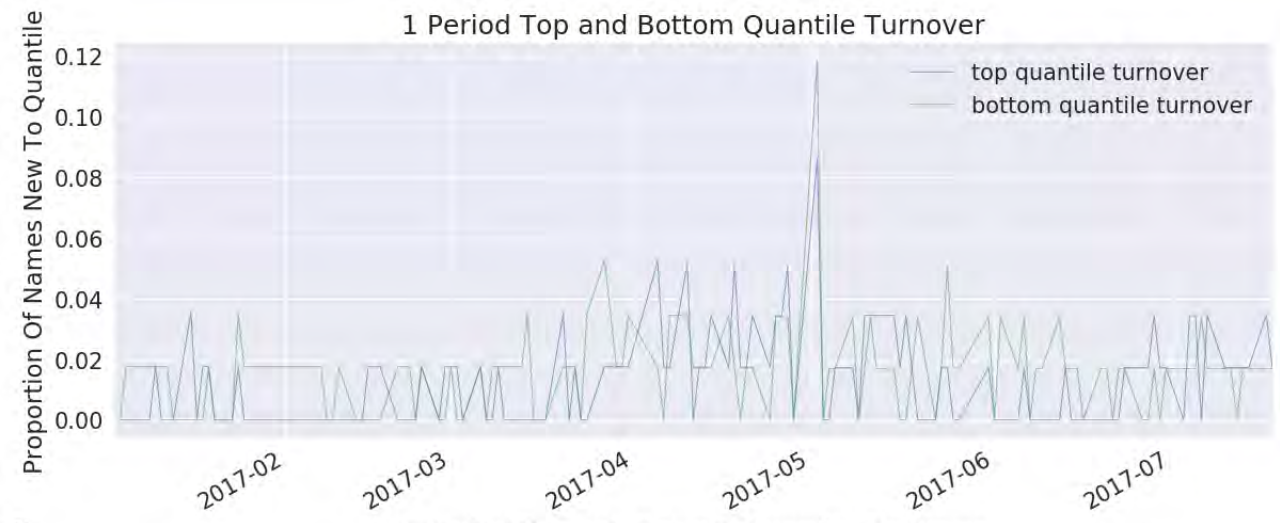
```
Turnover Analysis
```

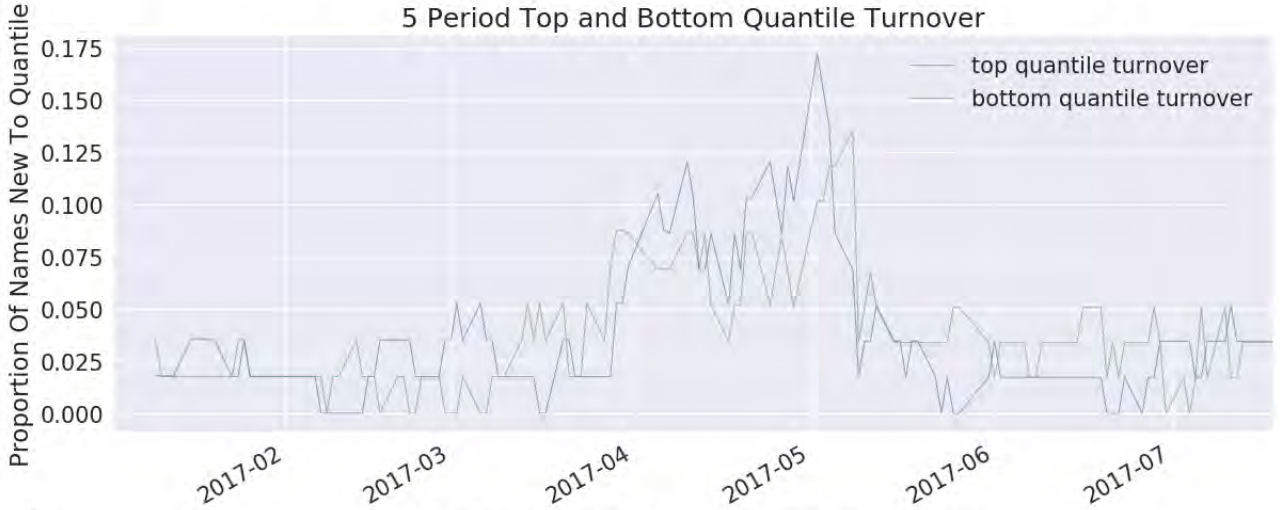
	1	5	10
Quantile 1 Mean T urnover	0.016	0.038	0.054
Quantile 2 Mean T urnover	0.036	0.085	0.119
Quantile 3 Mean T urnover	0.042	0.103	0.145
Quantile 4 Mean T urnover	0.034	0.086	0.124
Quantile 5 Mean T urnover	0.013	0.037	0.061

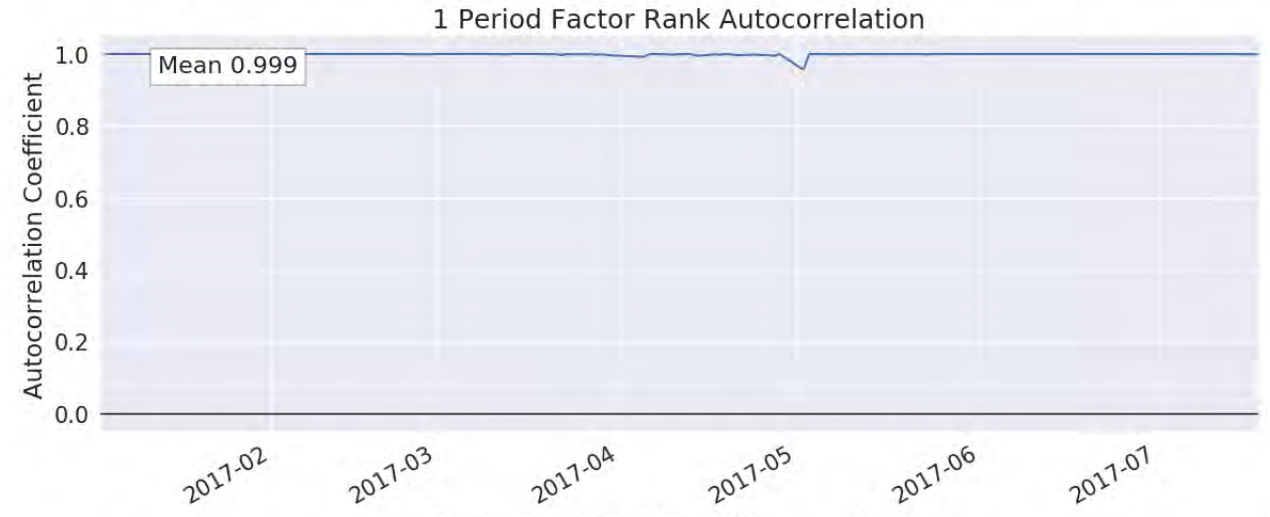
	1	5	10
Mean Factor Rank Autocorrelation	0.999	0.995	0.989

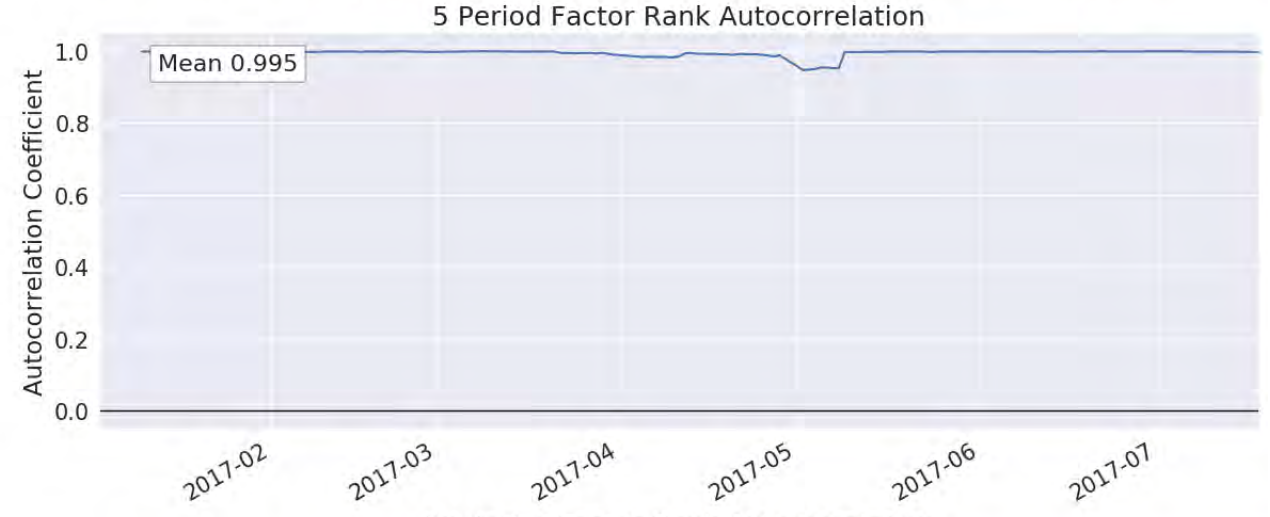
```
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```

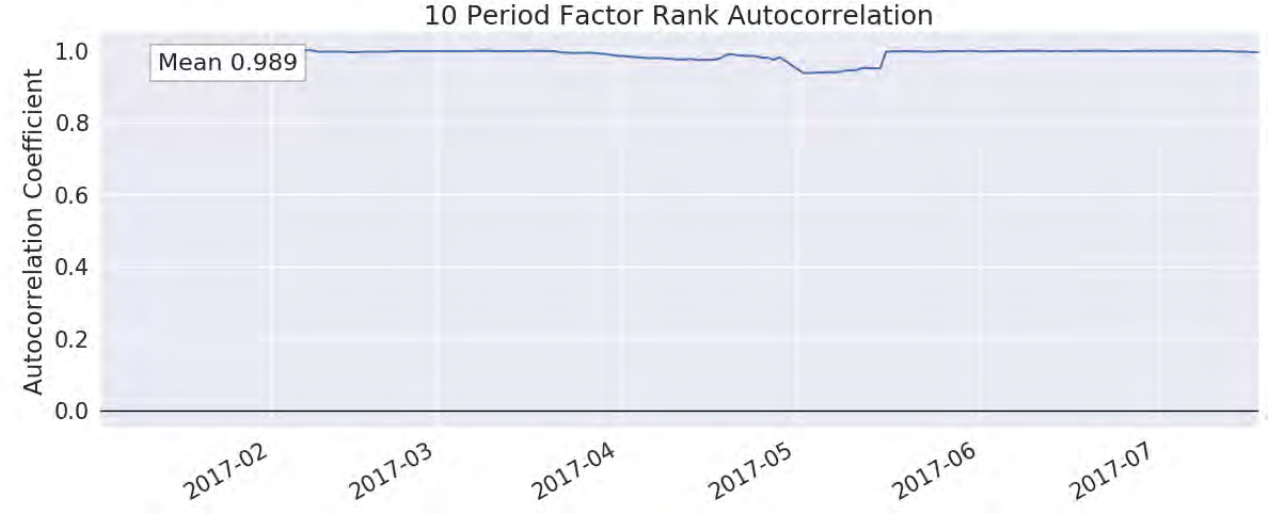
```
<matplotlib.figure.Figure at 0x7fb364673fd0>
```











4、 Information Analysis ¶

In [20]:

```
performance.factor_information_coefficient?
```

参数	类型	注释
factor_data	pd.DataFrame - MultiIndex	A MultiIndex DataFrame indexed by date (level 0) and asset (level 1), containing the values for a single alpha factor, forward returns for each period, The factor quantile/bin that factor value belongs too, and (optionally) the group the asset belongs to.
group_adjust	bool	Demean forward returns by group before computing IC.
by_group	bool	If True, compute period wise IC separately for each group.

1、计算因子值和预期收益之间的基于Spearman Rank Correlation （斯皮尔曼等级相关系数）的IC：¶

Computes the Spearman Rank Correlation based Information Coefficient (IC) between factor values and N period forward returns for each period in the factor index;

factor_data:里面放上面通过utils.get_clean_factor_and_forward_returns的dataframe

by_group:如果是True，会每个组都会计算IC;

group_adjust:在计算IC之前是否对预期收益进行处理

返回：Spearman Rank correlation between factor and provided forward returns.

In [21]:

```
IC = performance.factor_information_coefficient(facs_data_analysis,group_adjust=False,by_group=True)
IC.head()
```

Out[21]:

		1	5	10
date	group			
2017-01-03	big_MC	-0.054515	-0.093436	-0.163469
	mid_MC	0.018916	0.046430	-0.223072
	small_MC	0.111347	0.137631	-0.218596
	very_big_MC	0.199528	0.279660	-0.043889
	very_small_MC	-0.208929	-0.268201	-0.489790

2、Get the mean information coefficient of specified groups. ¶

获得某段时期或某种分组的平均IC¶

In [22]:

```
performance.mean_information_coefficient?
```

参数	类型	注释
factor_data	pd.DataFrame	A MultiIndex DataFrame indexed by date (level 0) and asset (level 1), containing the values for a single alpha factor, forward returns for each period, The factor quantile/bin that factor value belongs too, and (optionally) the group the asset belongs to.
group_adjust	bool	Demean forward returns by group before computing IC.
by_group	bool	If True, take the mean IC for each group.
by_time	str (pd time_rule), optional	Time window to use when taking mean IC. See http://pandas.pydata.org/pandas-docs/stable/timeseries.html for available options.

factor_data:里面放上面通过utils.get_clean_factor_and_forward_returns的dataframe

by_group:如果是True，计算每组的平均IC

group_adjust:在计算IC之前是否对预期收益进行处理

by_time:(按哪种时间规则计算，1q=1季度，1w=1周)

In [23]:

```
performance.mean_information_coefficient(facs_data_analysis,
                                group_adjust=False,
                                by_group=True,
                                by_time='1q')
```

Out[23]:

		1	5	10
date	group			
2017-03-31	big_MC	-0.051797	-0.103310	-0.132592
	mid_MC	-0.047242	-0.096206	-0.131520
	small_MC	-0.037899	-0.086773	-0.123012
	very_big_MC	-0.039192	-0.066714	-0.056065
	very_small_MC	-0.051387	-0.131445	-0.138329
2017-06-30	big_MC	-0.047627	-0.104323	-0.147700
	mid_MC	-0.039585	-0.065402	-0.105835
	small_MC	-0.040343	-0.069644	-0.112827
	very_big_MC	-0.006601	-0.046204	-0.103649
	very_small_MC	-0.044493	-0.069723	-0.115473
2017-09-30	big_MC	-0.010216	-0.022674	0.019017
	mid_MC	-0.034368	-0.136469	-0.159777
	small_MC	-0.060249	-0.204879	-0.248883
	very_big_MC	-0.143601	-0.343696	-0.408611
	very_small_MC	0.008471	0.017144	-0.002601

5、 其他¶

In [26]:

```
performance.factor_alpha_beta?
```

参数	类型	注释
factor_data	pd.DataFrame	A MultiIndex DataFrame indexed by date (level 0) and asset (level 1), containing the values for a single alpha factor, forward returns for each period, The factor quantile/bin that factor value belongs too, and (optionally) the group the asset belongs to.

计算alpha和beta¶

根据因子和预期收益计算alpha，beta等； actor_data:里面放上面通过utils.get_clean_factor_and_forward_returns的dataframe；

```
Compute the alpha (excess returns), alpha t-stat (alpha significance),
and beta (market exposure) of a factor. A regression is run with
the period wise factor universe mean return as the independent variable
and mean period wise return from a portfolio weighted by factor values
as the dependent variable.
```

In [27]:

```
performance.factor_alpha_beta(facs_data_analysis)
```

Out[27]:

	1	5	10
Ann. alpha	-0.211008	-0.227070	-0.230437
beta	0.185954	0.261177	0.370930

In [28]:

```
tears.create_event_returns_tear_sheet?
```

参数	类型	注释
factor_data	pd.DataFrame - MultiIndex	A MultiIndex Series indexed by date (level 0) and asset (level 1), containing the values for a single alpha factor, forward returns for each period, The factor quantile/bin that factor value belongs too, and (optionally) the group the asset belongs to.
prices	pd.DataFrame	A wide form Pandas DataFrame indexed by date with assets in the columns. Pricing data should span the factor analysis time period plus/minus an additional buffer window corresponding to periods_after/periods_before parameters.

avgretplot	tuple (int, int) - (before, after)	If not None, plot quantile average cumulative returns
long_short	bool	Should this computation happen on a long short portfolio?
by_group	bool	If True, view the average cumulative returns for each group.

这个函数是观察每组的累积收益的，可以在avgretplot调整要观察的period;

在by_group如果是True将会每组都展示平均累积收益

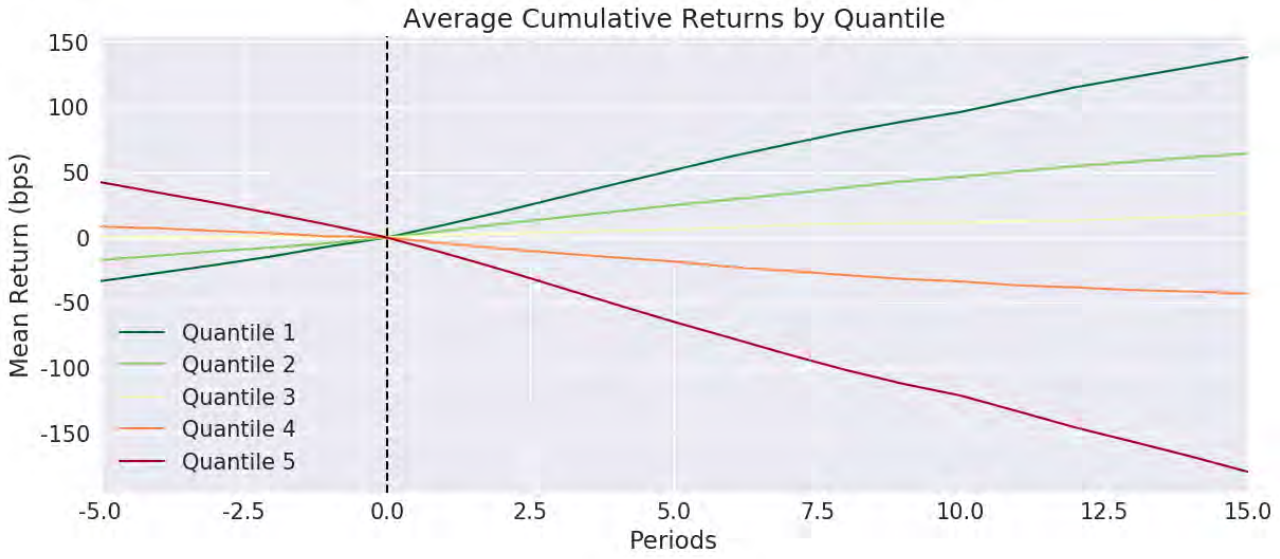
Creates a tear sheet to view the average cumulative returns for a factor within a window (pre and post event).

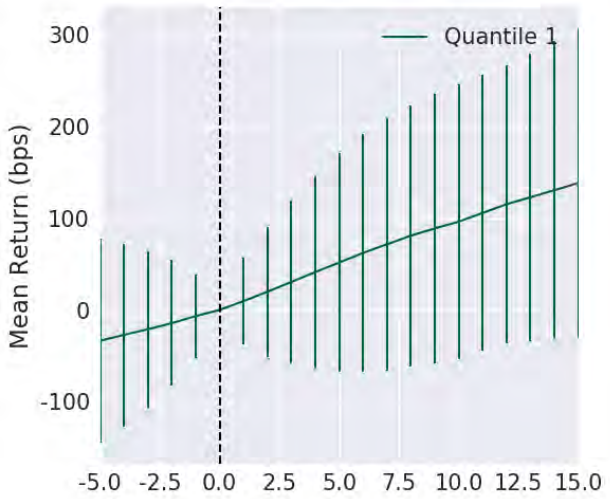
In [29]:

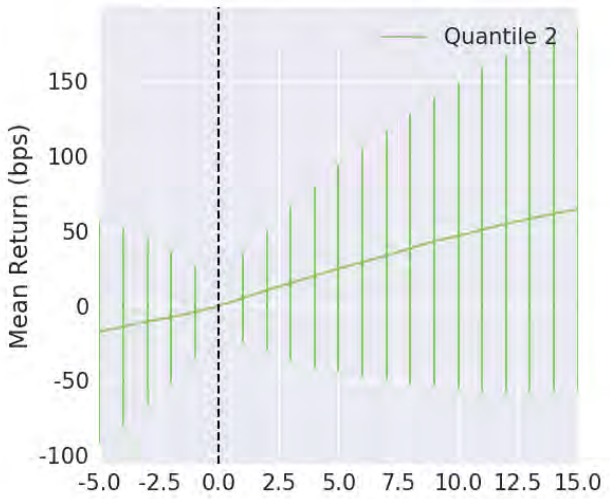
```
tears.create_event_returns_tear_sheet(facs_data_analysis,prices=price,avgretplot=(5, 15),
                                      long_short=True,
                                      by_group=True)
```

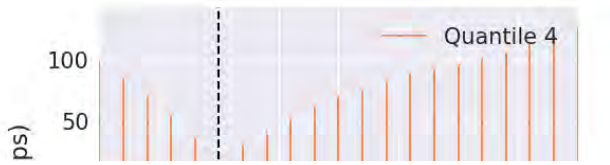
```
/srv/env/lib64/python3.4/site-packages/matplotlib/axes/_axes.py:2818: MatplotlibDeprecationWarning: Use of None object as fmt keyword
  warnings.warn(msg, mplDeprecation, stacklevel=1)
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```

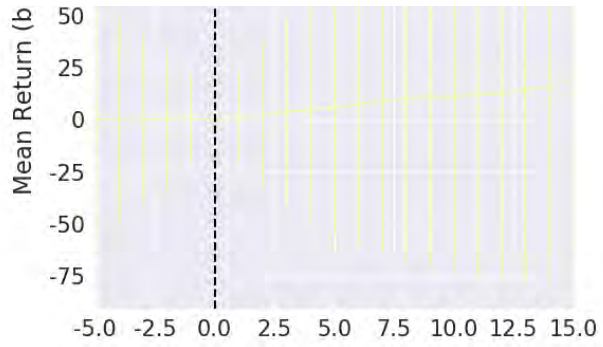
```
<matplotlib.figure.Figure at 0x7fb365ec3a58>
```

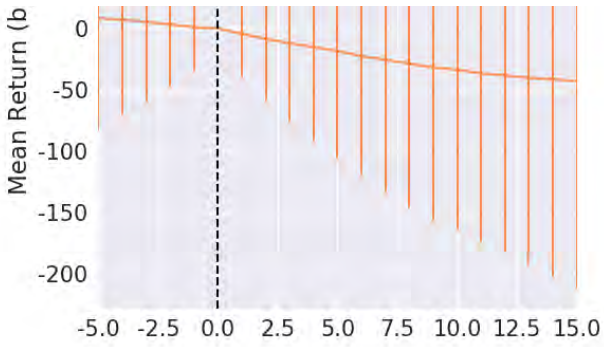


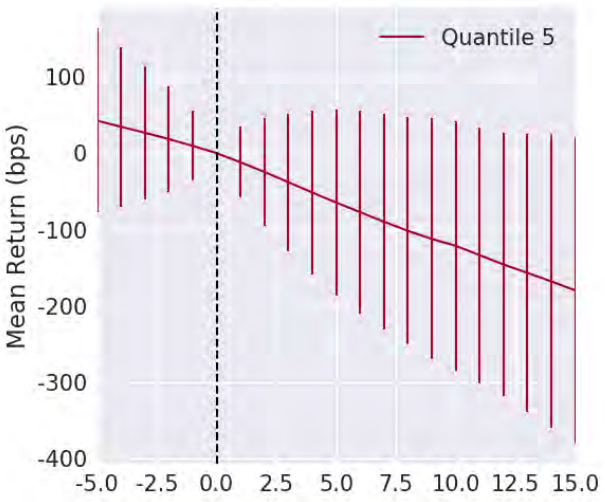


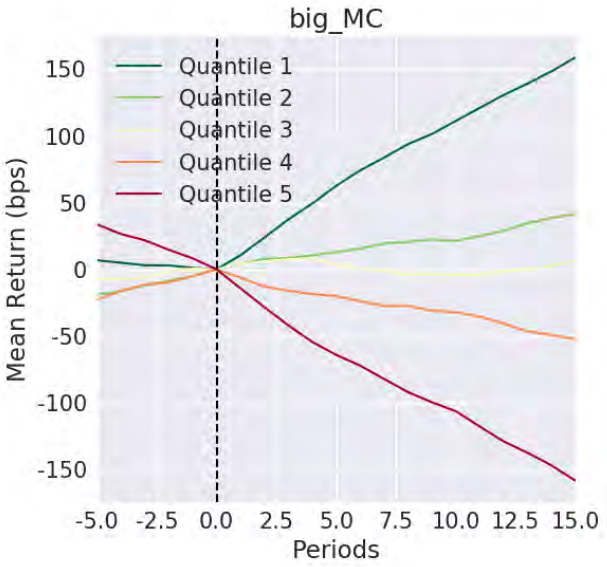


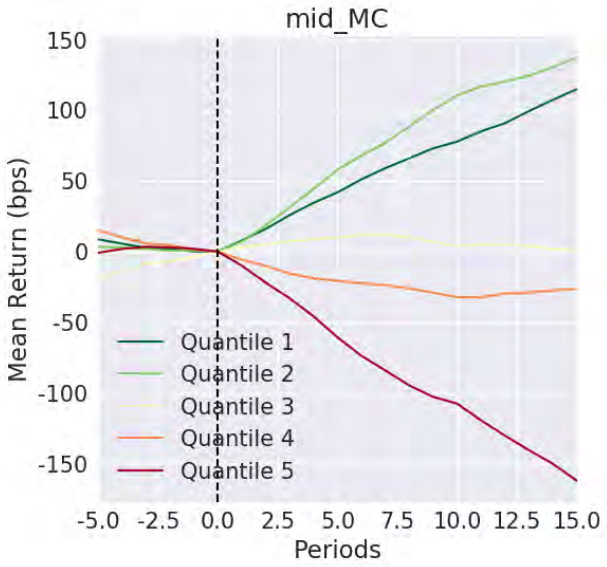


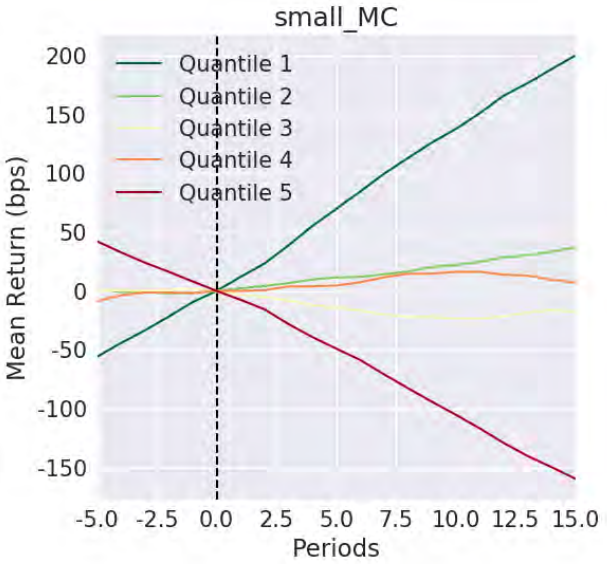

Mean Return (b
50
25
0
-25
-50
-75
-5.0 -2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0

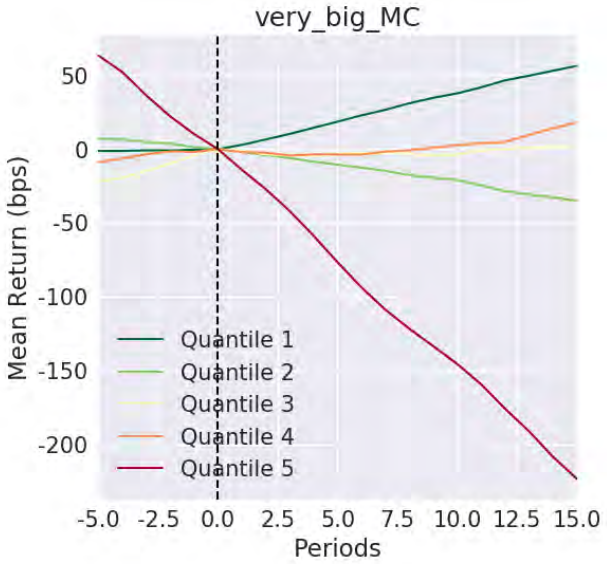

Mean Return (b
0
-50
-100
-150
-200
-5.0 -2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0

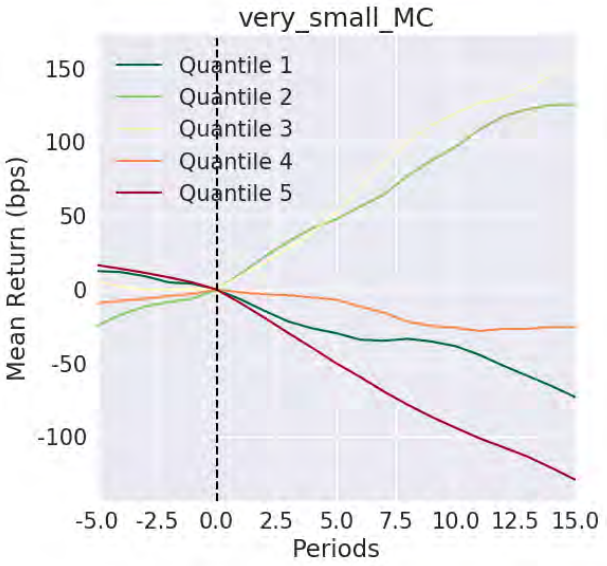

Quantile 5
Mean Return (bps)
100
0
-100
-200
-300
-400
-5.0 -2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0











绩效分析之Brinson 模型

在众多的绩效归因方法中，最出名的也许是Brinson分析。我们在这个帖子中将介绍Brinson模型，并使用实例让大家更清晰地明白我们RQBeta中的Brinson分析是如何算出来~

Brinson和Falcher(1985年)最早对业绩归因进行研究，由他们所创建Brinson模型将业绩归因为四个部分：资产配置、个股选择、交互作用和基准收益。

举一个很简单的例子，假如我们看到一个策略竟然有100%的收益，然而这100%是怎么来的呢？直观又简单的方法就是查看它的绩效分析，就可以看出它的收益的来源，是来自行业配置，还是选的个股表现超凡，更或者是来自大盘的涨跌呢？

一、单期Brinson 模型

首先，这里单期的意思是指在这段时期没有交易。我们先构建4个组合：

Q1 & Q4：业绩基准收益和实际投资组合；

Q2：主动资产配置组合。这表示此事基金经理能自主选择资产配置的比例，但是每个资产类别内部则完全按照其业绩基准配置，即每个资产i的收益等于在基准中资产i的收益；

Q3：主动股票选择组合。这表示此时基金经理完全按照业绩基准的比例进行资产类别的配置，但是每个资产内部能够自主选择个股，即组合中每个资产i的权重等于基准中资产i的权重。

它们的计算公式如下图，其中wp,i表示组合中资产i的权重；wb,i表示基准中资产i的权重；rp,i表示组合中资产i的收益率；rb,i表示基准中资产i的收益率。一般来说，资产i代表着不同行业。

	组合资产 i 收益（$r_{p,i}$）	基准资产 i 收益（$r_{b,i}$）
组合资产 i 权重（$w_{p,i}$）	$Q_4 = \sum_{i=1}^{n} w_{p,i} * r_{p,i}$	$Q_2 = \sum_{i=1}^{n} w_{p,i} * r_{b,i}$
基准资产 i 权重（$w_{b,i}$）	$Q_3 = \sum_{i=1}^{n} w_{b,i} * r_{p,i}$	$Q_1 = \sum_{i=1}^{n} w_{b,i} * r_{b,i}$

此时，我们的基准组合收益由上图可以表示为 rb = Q1。根据此表，我们可以表示出Brinson模型对这四类收益的定义

1、资产配置收益（即我们RQBeta的板块配置）——由资产配置带来的超额收益

假设我们能够自主选择决定组合中资产配置的比例，但是在每一个资产类别内部则完全按照该资产类别的基准配置，那么该组合的收益率超过基准收益率的部分称为资产配置收益（Asset Allocation）。

RAA = Q2 - Q1

2、个股选择收益（即我们RQBeta的版块内选股）——由个股选择而带来的超额收益

假设我们完全按照基准的比例进行资产类别配置，但是在每一个资产类别内部则能够自主进行个股选择，那么该组合的收益率超过基准收益率的部分称为个股选择收益（Stock Selection）。

RSS = Q3 - Q1

3、交互作用收益——由资产配置和个股选择的交互而带来的超额收益

资产组合的超额收益不仅来自资产配置收益和个股有选择收益，还有一部分是由于二者的交互作用所带来的收益，即所谓的交互作用（Interaction）。

RIN = Q4 - Q3 - Q2 + Q1

4、总超额收益

RTotal = Q4 - Q1

而后Damien Laker提出资产配置收益的定义可能会带来错误，故将之修改为：

$$R'_{AA} = R_{AA} - \sum_{i=1}^{n}(w_{p,i} - w_{b,i}) * r_b = R_{AA}$$

尽管新定义的加入了基准收益，但由于权重之和都为1，所以更改后的资产配置收益和原结果在总和上是相等的。

这两者的差异在于，原定义强调超额资产配置的收益来源于对收益更高的资产i的超配，或对收益低的资产j的低配；而新定义强调对表现优于基准总收益的类别i超配，或劣于基准总收益的类别j低配，这种更符合实际。

二、多期Brinson 模型

当我们需要以月度、季度、年度来考评时，此时的每期之间的个资产i的权重常有改变，所以单期的Brinson模型需要被推广至多期，其主要思路与单期一致

此时t期的各权重与收益计算如下：

t期：基准行业i权重： $w_{t,b,i}$，**组合资产i权重：** $w_{t,p,i}$；**基准行业i收益：** $r_{t,b,i}$，**组合资产i收益：** $r_{t,p,i}$

基准组合t期收益： $r_{t,b} = \sum_{i=1}^{n} w_{t,b,i} * r_{t,b,i}$，**资产组合t期收益：** $r_{t,p} = \sum_{i=1}^{n} w_{t,p,i} * r_{t,p,i}$

资产组合的多期累积收益，与基准组合的多期累积收益的计算公式如下：

资产组合累计收益： $R_p = (1 + r_{1,p})(1 + r_{2,p}) \dots (1 + r_{t,p}) - 1$

基准组合累计收益： $R_b = (1 + r_{1,b})(1 + r_{2,b}) \dots (1 + r_{t,b}) - 1$

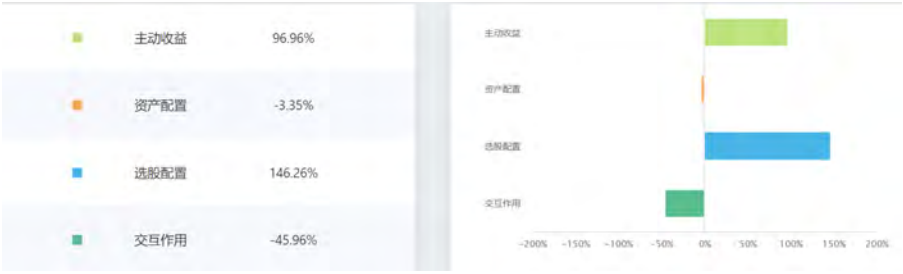
接下来的四类收益计算方法与定义与单期模型相同。

Brinson分析不仅可以适用于单独的策略收益分析，更可以拓广至更大的范围，例如基金经理，乃至公司。

我们以格雷厄姆策略为例，从 2015-01-01到 2017-10-01，初始股票资金 ￥100000 (每日交易数据)，

这是RQBeta的Brinson分析截图，接下来我们便来示范如何得到这几个结果~

	权重			收益			板块配置	板块内选股	交互作用	板块主动收益
	投资组合权重	基准组合权重	板块权重差值	投资组合收益	基准组合收益	板块收益差值				
现金	34.06%	0%	34.06%	0%	0%	0%	-1.82%	0%	0%	0%
能源	5.99%	2.79%	3.21%	68.24%	1.32%	66.92%	-0.13%	1.86%	2.15%	4.05%
原材料	1.73%	6.72%	-4.99%	542.89%	7.60%	535.29%	-0.11%	35.96%	-26.70%	8.88%
非必需消费品	8.37%	11.36%	-2.98%	66.65%	10.19%	56.47%	-0.14%	6.41%	-1.69%	4.42%
必需消费品	0%	6.78%	-6.78%	0%	21.25%	-21.25%	-1.08%	-1.44%	1.44%	-1.44%
医疗保健	4.44%	4.43%	0.00%	582.76%	13.64%	569.12%	0.00%	25.23%	0.03%	25.26%
金融	16.78%	41.50%	-24.73%	142.78%	2.56%	140.22%	0.69%	58.20%	-34.67%	22.89%
信息技术	0%	7.76%	-7.76%	0%	6.81%	-6.81%	-0.11%	-0.53%	0.53%	-0.53%
电信服务	0%	1.04%	-1.04%	0%	6.49%	-6.49%	-0.01%	-0.07%	0.07%	-0.07%
公共服务	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
工业	28.63%	17.62%	11.01%	116.77%	-0.33%	117.09%	-0.63%	20.63%	12.89%	33.49%
总计	100.00%	100.00%	0%	102.31%	5.35%	96.96%	-3.35%	146.26%	-45.96%	96.96%



Brinson.ipynb

克隆研究 +4

In [1]:

```
import pandas as pd
```

```
name = ['现金','能源','原材料','非必需消费品','必需消费品','医疗保健','金融','信息技术','电信服务','公共服务','工业']
```

```
w1 = [0.3406,0.0599,0.0173,0.0837,0.0000,0.0444,0.1678,0.0000,0.0000,0.0000,0.2863]
```

```
w2 = [0.0000,0.0279,0.0672,0.1136,0.0678,0.0443,0.4150,0.0776,0.0104,0.0000,0.1762]
```

```
r1 = [0.0000,0.6824,5.4289,0.6665,0.0000,5.8276,1.4278,0.0000,0.0000,0.0000,1.1677]
```

```
r2 = [0.0000,0.0132,0.0760,0.1019,0.2125,0.1364,0.0256,0.0681,0.0649,0.0000,-0.0033]
```

```
?
```

```
df = pd.DataFrame({'weight_portf_i': w1,
```

```
                   'weight_bench_i': w2,
```

```
                   'return_portf_i': r1,
```

```
                   'return_bench_i': r2},
```

```
                  index = name)
```

```
df
```

Out[1]:

	return_bench_i	return_portf_i	weight_bench_i	weight_portf_i
现金	0.0000	0.0000	0.0000	0.3406
能源	0.0132	0.6824	0.0279	0.0599
原材料	0.0760	5.4289	0.0672	0.0173
非必需消费品	0.1019	0.6665	0.1136	0.0837
必需消费品	0.2125	0.0000	0.0678	0.0000
医疗保健	0.1364	5.8276	0.0443	0.0444
金融	0.0256	1.4278	0.4150	0.1678
信息技术	0.0681	0.0000	0.0776	0.0000
电信服务	0.0649	0.0000	0.0104	0.0000
公共服务	0.0000	0.0000	0.0000	0.0000
工业	-0.0033	1.1677	0.1762	0.2863

In [2]:

```
df['Q1'] = df['return_bench_i']*df['weight_bench_i']
```

```
df['Q2'] = df['return_bench_i']*df['weight_portf_i']
```

```
df['Q3'] = df['return_portf_i']*df['weight_bench_i']
```

```
df['Q4'] = df['return_portf_i']*df['weight_portf_i']
```

```
df
```

Out[2]:

	return_bench_i	return_portf_i	weight_bench_i	weight_portf_i	Q1	Q2	Q3	Q4
现金	0.0000	0.0000	0.0000	0.3406	0.000000	0.000000	0.000000	0.000000
能源	0.0132	0.6824	0.0279	0.0599	0.000368	0.000791	0.019039	0.040876
原材料	0.0760	5.4289	0.0672	0.0173	0.005107	0.001315	0.364822	0.093920
非必需消费品	0.1019	0.6665	0.1136	0.0837	0.011576	0.008529	0.075714	0.055786
必需消费品	0.2125	0.0000	0.0678	0.0000	0.014408	0.000000	0.000000	0.000000
医疗保健	0.1364	5.8276	0.0443	0.0444	0.006043	0.006056	0.258163	0.258745
金融	0.0256	1.4278	0.4150	0.1678	0.010624	0.004296	0.592537	0.239585
信息技术	0.0681	0.0000	0.0776	0.0000	0.005285	0.000000	0.000000	0.000000

	return_bench_i	return_portf_i	weight_bench_i	weight_portf_i	Q1	Q2	Q3	Q4
电信服务	0.0649	0.0000	0.0104	0.0000	0.000675	0.000000	0.000000	0.000000
公共服务	0.0000	0.0000	0.0000	0.0000	0.000000	0.000000	0.000000	0.000000
工业	-0.0033	1.1677	0.1762	0.2863	-0.000581	-0.000945	0.205749	0.334313

In [3]:

```
result = pd.DataFrame()
```

```
result['板块配置'] = df['Q2'] - df['Q1']-(df['weight_portf_i'] - df['weight_bench_i'])*sum(df['Q1'])
```

```
result['板块内选股'] = df['Q3'] - df['Q1']
```

```
result['交互收益'] = df['Q4'] - df['Q3'] - df['Q2'] + df['Q1']
```

```
result['总超额收益'] = df['Q4'] - df['Q1']
```

```
result.ix['合计'] = result.apply(lambda x:x.sum())
```

```
result
```

Out[3]:

	板块配置	板块内选股	交互收益	总超额收益
现金	-0.018223	0.000000	0.000000	0.000000
能源	-0.001290	0.018671	0.021414	0.040507
原材料	-0.001123	0.359715	-0.267110	0.088813
非必需消费品	-0.001447	0.064139	-0.016882	0.044210
必需消费品	-0.010780	-0.014408	0.014408	-0.014408
医疗保健	0.000008	0.252120	0.000569	0.252703
金融	0.006898	0.581913	-0.346624	0.228961
信息技术	-0.001133	-0.005285	0.005285	-0.005285
电信服务	-0.000119	-0.000675	0.000675	-0.000675
公共服务	0.000000	0.000000	0.000000	0.000000
工业	-0.006254	0.206330	0.128927	0.334894
合计	-0.033462	1.462520	-0.459337	0.969721

In [1]:

```
import pandas as pd
name = ['现金','能源','原材料','非必需消费品','必需消费品','医疗保健','金融','信息技术','电信服务','公共服务','工业']
w1 = [0.3406,0.0599,0.0173,0.0837,0.0000,0.0444,0.1678,0.0000,0.0000,0.0000,0.2863]
w2 = [0.0000,0.0279,0.0672,0.1136,0.0678,0.0443,0.4150,0.0776,0.0104,0.0000,0.1762]
r1 = [0.0000,0.6824,5.4289,0.6665,0.0000,5.8276,1.4278,0.0000,0.0000,0.0000,1.1677]
r2 = [0.0000,0.0132,0.0760,0.1019,0.2125,0.1364,0.0256,0.0681,0.0649,0.0000,-0.0033]

df = pd.DataFrame({'weight_portf_i': w1,
                   'weight_bench_i': w2,
                   'return_portf_i': r1,
                   'return_bench_i': r2},
                  index = name)
df
```

Out[1]:

	return_bench_i	return_portf_i	weight_bench_i	weight_portf_i
现金	0.0000	0.0000	0.0000	0.3406
能源	0.0132	0.6824	0.0279	0.0599
原材料	0.0760	5.4289	0.0672	0.0173
非必需消费品	0.1019	0.6665	0.1136	0.0837
必需消费品	0.2125	0.0000	0.0678	0.0000
医疗保健	0.1364	5.8276	0.0443	0.0444
金融	0.0256	1.4278	0.4150	0.1678
信息技术	0.0681	0.0000	0.0776	0.0000
电信服务	0.0649	0.0000	0.0104	0.0000
公共服务	0.0000	0.0000	0.0000	0.0000
工业	-0.0033	1.1677	0.1762	0.2863

In [2]:

```
df['Q1'] = df['return_bench_i']*df['weight_bench_i']
df['Q2'] = df['return_bench_i']*df['weight_portf_i']
df['Q3'] = df['return_portf_i']*df['weight_bench_i']
df['Q4'] = df['return_portf_i']*df['weight_portf_i']
df
```

Out[2]:

	return_bench_i	return_portf_i	weight_bench_i	weight_portf_i	Q1	Q2	Q3	Q4
现金	0.0000	0.0000	0.0000	0.3406	0.000000	0.000000	0.000000	0.000000
能源	0.0132	0.6824	0.0279	0.0599	0.000368	0.000791	0.019039	0.040876
原材料	0.0760	5.4289	0.0672	0.0173	0.005107	0.001315	0.364822	0.093920
非必需消费品	0.1019	0.6665	0.1136	0.0837	0.011576	0.008529	0.075714	0.055786
必需消费品	0.2125	0.0000	0.0678	0.0000	0.014408	0.000000	0.000000	0.000000
医疗保健	0.1364	5.8276	0.0443	0.0444	0.006043	0.006056	0.258163	0.258745
金融	0.0256	1.4278	0.4150	0.1678	0.010624	0.004296	0.592537	0.239585
信息技术	0.0681	0.0000	0.0776	0.0000	0.005285	0.000000	0.000000	0.000000
电信服务	0.0649	0.0000	0.0104	0.0000	0.000675	0.000000	0.000000	0.000000

	return_bench_i	return_portf_i	weight_bench_i	weight_portf_i	Q1	Q2	Q3	Q4
公共服务	0.0000	0.0000	0.0000	0.0000	0.000000	0.000000	0.000000	0.000000
工业	-0.0033	1.1677	0.1762	0.2863	-0.000581	-0.000945	0.205749	0.334313

In [3]:

```
result = pd.DataFrame()
result['板块配置'] = df['Q2'] - df['Q1']-(df['weight_portf_i'] - df['weight_bench_i'])*sum(df['Q1'])
result['板块内选股'] = df['Q3'] - df['Q1']
result['交互收益'] = df['Q4'] - df['Q3'] - df['Q2'] + df['Q1']
result['总超额收益'] = df['Q4'] - df['Q1']
result.ix['合计'] = result.apply(lambda x:x.sum())
result
```

Out[3]:

	板块配置	板块内选股	交互收益	总超额收益
现金	-0.018223	0.000000	0.000000	0.000000
能源	-0.001290	0.018671	0.021414	0.040507
原材料	-0.001123	0.359715	-0.267110	0.088813
非必需消费品	-0.001447	0.064139	-0.016882	0.044210
必需消费品	-0.010780	-0.014408	0.014408	-0.014408
医疗保健	0.000008	0.252120	0.000569	0.252703
金融	0.006898	0.581913	-0.346624	0.228961
信息技术	-0.001133	-0.005285	0.005285	-0.005285
电信服务	-0.000119	-0.000675	0.000675	-0.000675
公共服务	0.000000	0.000000	0.000000	0.000000
工业	-0.006254	0.206330	0.128927	0.334894
合计	-0.033462	1.462520	-0.459337	0.969721

学术专区

布林强盗策略

本策略基于布林通道，以焦炭的自制主力连续合约为例，收益和回撤结果比较好。

策略要素：

1、基于收盘价计算而来的布林通道

2、基于收盘价计算而来的进场过滤器

3、自适应出场均线

入场条件：

多头：满足过滤条件，并且价格上破布林通道上轨；

空头：满足过滤条件，并且价格上破布林通道下轨；

出场条件：

多头：当自适应出场均线低于布林通道上轨，并且价格下破自适应出场均线；

空头：当自适应出场均线高于布林通道下轨，并且价格上破自适应出场均线。

之前写的期货策略~

《平移布林带》

《基于开收盘价格间的相对关系变化进行判断的策略》

《均线和K线形态的高低点突破》

《基于价格通道突破（简单使用SAR止损）》

《平移通道策略》

《价格和均线相关差策略》

《平移的高低点均值通道与K线中值》

《指数移动平均线策略》

《波动加权后的WOBV（成交量变动趋势）策略》

《价格区间突破策略》

https://www.ricequant.com/community/topic/4144/

配对交易策略的简述：¶

配对交易其基本思想在社区里有前辈已经说得很清楚了，但是其局限性在于配对的hedge ratio目前还是一个固定值，并且存在样本内表现很好，但样本外表现不尽如人意的现象。我们知道，随着时间的推移，最优的对冲比率应该是不断变化的，所以如何使hedge ratio动起来，这是我们这次研究的主要目的。 目前参照国外相关网站的资料知道，用Kalman Filter来动态调整hedge ratio是当下一个比较流行的做法，这次思路也是跟随于此

在构建配对交易策略之前，我们首先要做的就是找出具有协整关系的股票对，但这里有点需要强调的是相关系数高的与协整关系的强弱没有必然的联系，这是一个大家容易进的误区，底下会给出具体的事例加以说明

在找出协整关系股票标的基础上，接下来就是使用Kalman Filter方法来估计出state_means，此处其是一个二维数组，前者代表斜率后者代表截距项。关于Kalman Filter的具体原理，有兴趣的读者可以继续深挖，此处就不做详细介绍，因为此处主要是对方法的应用，重点在配对交易上 为了更好的说明问题，我在这里同时应用了静态对冲和动态对冲两种方法，以比较二者的差异，具体的实战表现如何则需要通过策略板块来进行回测模拟，这个后面会补充进来

下面进入Demo

In [1]:

```
import numpy as np
import pandas as pd
import statsmodels.tsa.stattools as sts
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import coint
from pykalman import KalmanFilter
```

配对交易之前，我们首先是要找到具有协整关系的股票对，然后在此基础上构建策略 但我们大部分读者以前可能误以为相关性很高的股票对就是协整性很强的配对交易标的，这是不对的，以下以两个实例来解释说明一下

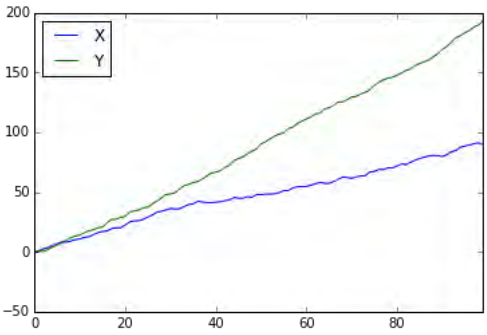
Instance 1 : Correlation without Cointegration ¶

In [2]:

```
np.random.seed(100)
```

In [3]:

```
X = np.random.normal(1,1,100)
Y = np.random.normal(2,1,100)

X_cum = pd.Series(np.cumsum(X), name='X')
Y_cum = pd.Series(np.cumsum(Y), name='Y')

pd.concat([X_cum, Y_cum], axis=1).plot();
```



In [4]:

```
_, pvalue, _ = coint(X_cum, Y_cum)
corr = np.corrcoef(X_cum, Y_cum)[0,1]
```

```
print ('cointegration pvalue = ',pvalue)
print ('correlation coefficient = ',corr)
```

```
cointegration pvalue =  0.16104204814
correlation coefficient =  0.990499430003
```

从以上结果可知，协整的P值显著不为0，说明不能拒绝原假设（null hypothesis） The Null hypothesis is that there is no cointegration, the alternative hypothesis is that there is cointegrating relationship. If the pvalue is small, below a critical size, then we can reject the hypothesis that there is no cointegrating relationship.

相关系数接近100%，为0.99，说明二者的相关性很高

Instance 2 ： Cointegration without Correlation ¶

In [5]:

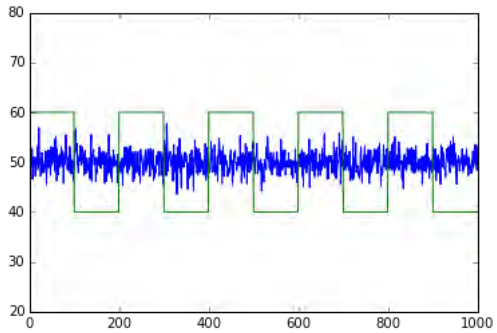
```
X = pd.Series(2*np.random.normal(0,1,1000)) + 50
X_wave = X.copy()
```

In [6]:

```
X_wave[:100] = 60
X_wave[100:200] = 40
X_wave[200:300] = 60
X_wave[300:400] = 40
X_wave[400:500] = 60
X_wave[500:600] = 40
X_wave[600:700] = 60
X_wave[700:800] = 40
X_wave[800:900] = 60
X_wave[900:1000] = 40

X.plot()
X_wave.plot()
plt.ylim([20,80])
```

Out[6]:

```
(20, 80)
```



In [7]:

```
_, pvalue, _ = coint(X, X_wave)
corr = np.corrcoef(X, X_wave)[0,1]
print ('cointegration pvalue = ',pvalue)
print ('correlation coefficient = ',corr)
```

```
cointegration pvalue =  0.0
correlation coefficient =  0.00875576208136
```

协整的P值为零，故拒绝没有协整关系的原假设，但是二者的相关系数接近于零，说明几乎无相关性

以上两个虚构的例子只是为了说明协整和相关性并不是一回事，大家要走出这个误区，不能简单的以相关性的高低来判断协整关系的强弱

下面我们以真实市场的股票价格来进行配对交易的策略设计

由于股灾期间，发生停牌股的股票标的特别多，所以此处选取的股票基数应比较大，此处选取50

In [8]:

```
# 从同一板块选取股票这里我们选择金融板块，主要是金融股大部分是蓝筹股，流动性高，适合作为配对交易的标的
selected_stocks = sector('Financials')
```

In [9]:

```
# 提取股票价格
start = '2014-01-01'
end = '2015-12-31'
stock_df = get_price(selected_stocks, start_date=start, end_date=end, fields='ClosingPx').dropna(axis=1)
stock_df.tail()
```

Out[9]:

	000001.XSHE	000002.XSHE	000006.XSHE	000011.XSHE	000014.XSHE	000029.XSHE	000031.XS
MDEntryDate							
2015-12-25	12.41	24.43	12.06	15.16	23.61	13.46	14.58
2015-12-28	11.98	24.43	11.51	14.32	22.25	12.92	14.15
2015-12-29	12.09	24.43	11.74	14.44	23.08	12.92	14.36
2015-12-30	12.10	24.43	11.76	14.49	23.25	12.83	14.33
2015-12-31	11.99	24.43	11.51	14.52	22.81	12.92	14.05

5 rows × 185 columns

因为股票作为配对交易标的时，主要是考察它们的收益率之间是否具有协整关系，所以我们这里也转为考察收益率

In [10]:

```
stock_ret_df = np.log(stock_df).diff().dropna()
stock_ret_df.head()
```

Out[10]:

	000001.XSHE	000002.XSHE	000006.XSHE	000011.XSHE	000014.XSHE	000029.XSHE	000031.XS
MDEntryDate							
2014-01-03	-0.025046	-0.018952	-0.029169	-0.023287	-0.016643	-0.021334	-0.016439
2014-01-06	-0.021979	-0.047006	-0.058776	-0.037338	-0.045439	-0.049734	-0.025176
2014-01-07	-0.002472	-0.006707	-0.002245	0.001358	0.005152	-0.008535	0.002829
2014-01-08	0.011077	-0.001347	-0.013575	-0.021949	-0.007220	0.000000	-0.005666
2014-01-09	0.004884	0.005376	-0.039494	-0.012561	-0.027284	-0.034887	-0.014306

5 rows × 185 columns

累计5天价格没有发生变化,等价于收益率为0，则认为所选时间段期间股票发生了停牌，由于停牌股在复牌后表现异常，故排除掉此类股票

In [11]:

```
# import copy

selected_stocks = list(stock_ret_df.columns.values)
#print(len(selected_stocks))
for stock in selected_stocks[:]:           # 注意一定要用selected_stocks[:]
    #print(stock)
    s=stock_ret_df[stock_ret_df[stock]!=0]
    if len(s)<len(stock_ret_df) - 5:
        selected_stocks.remove(stock)
print ('selected stocks :',selected_stocks)
```

```
selected stocks : ['000728.XSHE', '600773.XSHG', '601601.XSHG', '601628.XSHG']
```

In [12]:

```
# 找出以上具有协整关系的股票对的函数
def  find_cointegrated_pairs(stock_list, stock_ret_df):    # 两个参数分别代表股票池和收益的DataFrame
    n = len(stock_list)
    score_matrix = np.zeros((n,n))
    pvalue_matrix = np.ones((n,n))    # 注意初始矩阵构建的设定值
    pairs = []

    for i in range(n):
        for j in range(i+1, n):
            S1 = stock_ret_df[stock_list[i]]
            S2 = stock_ret_df[stock_list[j]]
            result = coint(S1,S2)
            score = result[0]         # 协整检验的t值（可以选择不予考虑）
```

```
            pvalue = result[1]        # 协整检验的P值
            score_matrix[i,j] = score
            pvalue_matrix[i,j] = pvalue
            if pvalue < 0.05:
                pairs.append((stock_list[i],stock_list[j]))

    return score_matrix, pvalue_matrix, pairs
```

In [13]:

```
# 找出股票收益之间符合配对条件的股票对
# 调用函数
scores, pvalues, pairs = find_cointegrated_pairs(selected_stocks, stock_ret_df)
print ('cointegrated pairs : ',pairs)
```

```
cointegrated pairs :  [('000728.XSHE', '600773.XSHG'), ('000728.XSHE', '601601.XSHG'), ('000728.XSHE', '601628.XSHG'), ('600773.XSHG'
```

不难发现同板块之间的股票具有协整关系的太多，我们画图更直观的看一下¶

我们随机选取其中一对股票对进行验证，观察一下他们的收益走势图

In [14]:

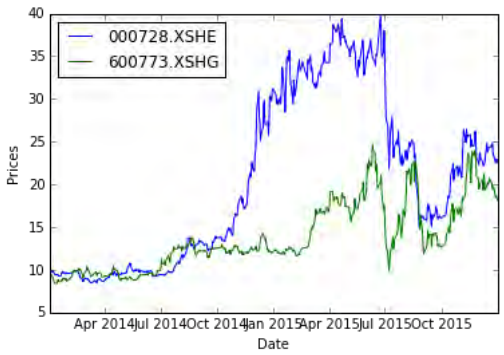
```
# 选取第一组 （五只股票之间，两两都具有协整关系，此处任意选择）
stock_1 = pairs[0][0]  # 第一组第一支股票
stock_2 = pairs[0][1]    # 第一组第二支股票
_, pvalue, _ = coint(stock_ret_df[stock_1], stock_ret_df[stock_2])
print (pvalue)
```

```
0.0
```

P值为零，说明具有强协整关系

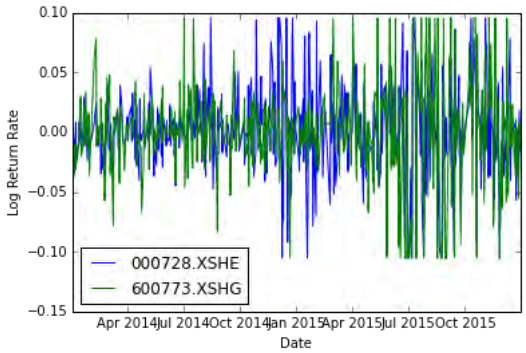
In [15]:

```
# 画出股价走势图
plt.plot(stock_df.index, stock_df[stock_1], label=pairs[0][0])
plt.plot(stock_df.index, stock_df[stock_2], label=pairs[0][1])
plt.legend(loc='best')
plt.xlabel('Date')
plt.ylabel('Prices');
```



In [16]:

```
# 画出二者的收益走势图
plt.plot(stock_ret_df.index, stock_ret_df[pairs[0][0]],label=pairs[0][0])
plt.plot(stock_ret_df.index, stock_ret_df[pairs[0][1]],label=pairs[0][1])
plt.legend(loc='best')
plt.xlabel('Date')
plt.ylabel('Log Return Rate');
```



由于收益变动图并不是很容易发现二者协整的关系，我们补充一个价格走势或者累计收益图

In [17]:

```
stock_ret_df.ix[0,:]=1
stock_ret_df.head()
```

Out[17]:

	000001.XSHE	000002.XSHE	000006.XSHE	000011.XSHE	000014.XSHE	000029.XSHE	000031.XS
MDEntryDate							
2014-01-03	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
2014-01-06	-0.021979	-0.047006	-0.058776	-0.037338	-0.045439	-0.049734	-0.025176
2014-01-07	-0.002472	-0.006707	-0.002245	0.001358	0.005152	-0.008535	0.002829
2014-01-08	0.011077	-0.001347	-0.013575	-0.021949	-0.007220	0.000000	-0.005666
2014-01-09	0.004884	0.005376	-0.039494	-0.012561	-0.027284	-0.034887	-0.014306

5 rows × 185 columns

In [18]:

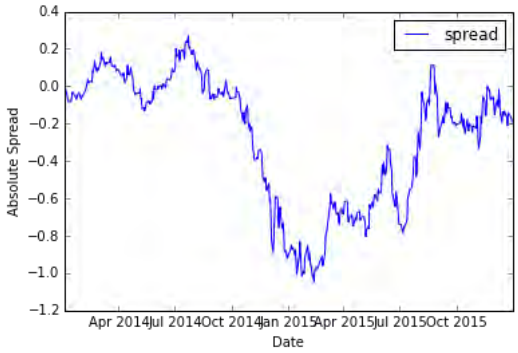
```
#画出二者的累计收益率走势图

cum_ret=stock_ret_df[list(pairs[0])].cumsum()
plt.plot(stock_ret_df.index,cum_ret.ix[:,0],'k-',label=pairs[0][0])
plt.plot(stock_ret_df.index,cum_ret.ix[:,1],'r-',label=pairs[0][1])
plt.legend(loc='best')
plt.xlabel('Date')
plt.ylabel('Cumulative Return Rate');
```

In [19]:

```
# 求出两支股票累计收益率的价差
abs_spread = cum_ret.diff(axis=1)
del abs_spread[stock_1]
abs_spread.rename(columns={str(stock_2):'spread'},inplace=True)
print (abs_spread.head())
plt.plot(abs_spread.index, abs_spread, label='spread')
plt.xlabel('Date')
plt.ylabel('Absolute Spread')
plt.legend(loc='best');
```

```
              spread
MDEntryDate
2014-01-03   0.000000
2014-01-06  -0.031024
```

```
2014-01-07  -0.032629
2014-01-08  -0.071846
2014-01-09  -0.073031
```



In [20]:

```
# 收益率之差的绝对值在统计学上的含义不明显，在这里转化为标准正太分布，以更直观的理解
def zscores(series):
    return (series - series.mean())/np.std(series)
```

In [21]:

```
print (zscores(abs_spread).head())
spread_series = zscores(abs_spread)

spread_series.plot(lw=2,color='k')
plt.axhline(np.mean(spread_series.values), color='k')
plt.axhline(1, color='r', linestyle='--')          # 此处选取的是一个标准差
plt.axhline(-1, color='r', linestyle='--');
```

```
              spread
MDEntryDate
2014-01-03   0.791311
2014-01-06   0.701482
2014-01-07   0.696833
2014-01-08   0.583283
2014-01-09   0.579851
```

Static Hedging ¶

现在我们先来看一下传统静态对冲的配对交易策略的效果，即取一个恒定的Beta值 进行静态对冲前，我们先来找出两个股票对之间的回归系数（alpha和beta值） 因为主要考察收益率之间的关系，所以我们依然继续回归到收益率层面

In [22]:

```
# 定义一个线性回归函数
from statsmodels import regression
import statsmodels.api as sm

stock_ret_df = stock_ret_df.ix[1:,:]

ret_1 = stock_ret_df[stock_1].values
ret_2 = stock_ret_df[stock_2].values

def linreg(x, y):
    x = sm.add_constant(x)
    model = regression.linear_model.OLS(y,x).fit()
    x = x[:,1]
    return model.params[0], model.params[1]
```

```
alpha, beta = linreg(ret_1, ret_2)
print ('alpha: ',alpha)
print ('beta: ',beta)
```

```
alpha:  0.000592403871854
beta:  0.425879354528
```

In [23]:

```
stock_ret_df.head()
```

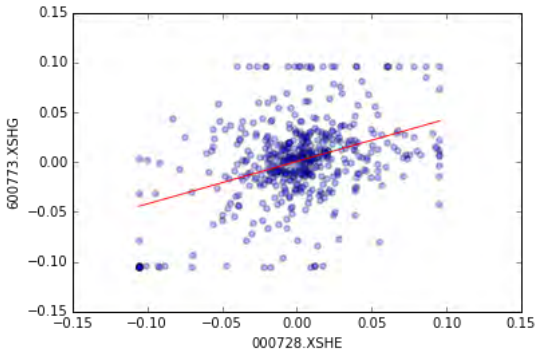
Out[23]:

	000001.XSHE	000002.XSHE	000006.XSHE	000011.XSHE	000014.XSHE	000029.XSHE	000031.XS
MDEntryDate							
2014-01-06	-0.021979	-0.047006	-0.058776	-0.037338	-0.045439	-0.049734	-0.025176
2014-01-07	-0.002472	-0.006707	-0.002245	0.001358	0.005152	-0.008535	0.002829
2014-01-08	0.011077	-0.001347	-0.013575	-0.021949	-0.007220	0.000000	-0.005666
2014-01-09	0.004884	0.005376	-0.039494	-0.012561	-0.027284	-0.034887	-0.014306
2014-01-10	0.000000	-0.010782	-0.021558	-0.035744	-0.041265	0.005900	-0.035194

5 rows × 185 columns

In [24]:

```
# 画出用估计出的alpha值和beta值拟合的图形
new_ret_1 = np.linspace(ret_1.min(), ret_1.max(), 100)
ret_2_hat = new_ret_1 * beta + alpha

plt.scatter(ret_1, ret_2, alpha=0.3)
plt.plot(new_ret_1, ret_2_hat, alpha=0.9,color='r')
plt.xlabel(pairs[0][0])
plt.ylabel(pairs[0][1]);
```

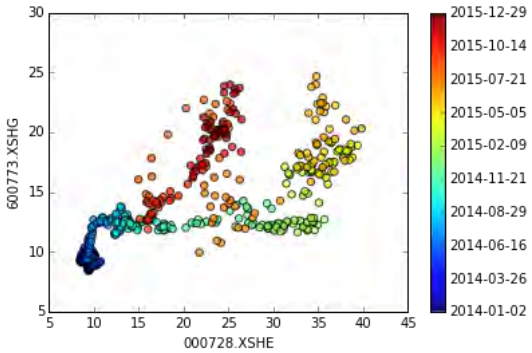


Dynamic Hedging ¶

接下来我们看看用KalmanFilter动态调整beta值会出现什么情况

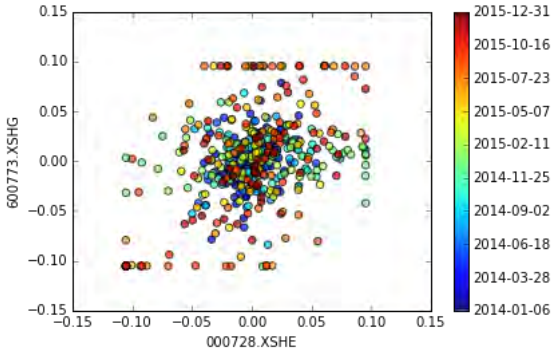
In [25]:

```
from pykalman import KalmanFilter

# 我们先来看看两种资产动态的价格变化散点图
cm = plt.cm.get_cmap('jet')
dates = [str(p.date()) for p in stock_df[::len(stock_df)//10].index]
colors = np.linspace(0.1, 1, len(stock_df.index))
sc = plt.scatter(stock_df[stock_1],stock_df[stock_2], s=30, c=colors, cmap=cm, edgecolor='k', alpha=0.7)
cb = plt.colorbar(sc)
cb.ax.set_yticklabels([str(p.date()) for p in stock_df[::len(stock_df)//9].index])
plt.xlabel(stock_1)
plt.ylabel(stock_2);
```



In [26]:

```
# 我们先来看看两种资产动态的收益率变化散点图
cm = plt.cm.get_cmap('jet')
dates = [str(p.date()) for p in stock_ret_df[::len(stock_df)//10].index]
colors = np.linspace(0.1, 1, len(stock_ret_df.index))
sc = plt.scatter(stock_ret_df[stock_1],stock_ret_df[stock_2], s=30, c=colors, cmap=cm, edgecolor='k', alpha=0.7)
cb = plt.colorbar(sc)
cb.ax.set_yticklabels([str(p.date()) for p in stock_ret_df[::len(stock_ret_df)//9].index])
plt.xlabel(stock_1)
plt.ylabel(stock_2);
```



In [27]:

```
# 现在开始用KalmanFilter的方法进行动态估计beta值
delta = 1e-3
trans_cov = delta/(1-delta)*np.eye(2)
obs_mat = np.vstack([stock_ret_df[stock_1], np.ones(stock_ret_df[stock_1].shape)]).T[:,np.newaxis]
```

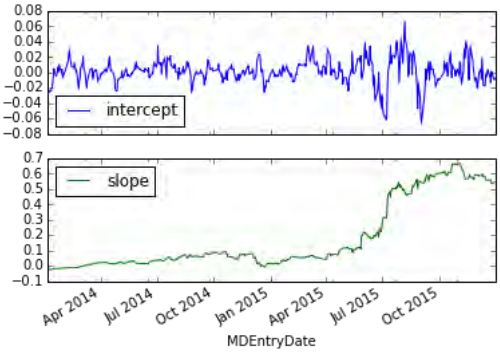
In [28]:

```
kf = KalmanFilter(n_dim_obs=1, n_dim_state=2,       # ret_2 is 1-dimensional, (alpha, beta) is 2-dimensional
                  initial_state_mean=np.zeros(2),
                  initial_state_covariance=np.ones((2,2)),
                  transition_matrices=np.eye(2),
                  observation_matrices=obs_mat,
                  observation_covariance=0.01,
                  transition_covariance=trans_cov)
```

In [29]:

```
state_means, state_covs = kf.filter(stock_ret_df[stock_2])
pd.DataFrame(dict(slope=state_means[:,0], intercept=state_means[:,1]), index=stock_ret_df.index).plot(subplots=True)
plt.tight_layout
print ('alpha :',np.mean(state_means[:,1]))
print ('beta :',np.mean(state_means[:,0]));
```

```
alpha : 0.00137472851765
beta : 0.178758327491
```



相比于静态对冲的方式，我们发现动态对冲的alpha（截距项）值也是在0附近，但是beta值相差较大，主要是因为beta值在不断更新的情形下，最近的值会占据较大的权重，并且初始值如果设定差异过大（此处初始值设为0），会显著影响其均值，而最后时刻更新的beta比较接近静态对冲的值

为了更好地理解Kalman Filter的方法和原理，下次会单独开始一个研究板块进行补充说明

最后，关于两种对冲方式下的表现，我会在策略板块下更新，敬请关注<吐舌>

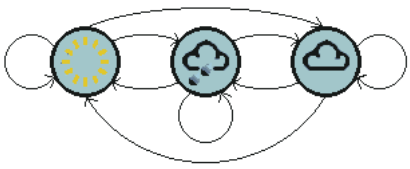
In [ ]:

HMM在股票上的简单应用

今天我们来介绍一下HMM（隐马尔科夫模型）在股票上的简单应用。
隐马尔科夫模型，乍一听起来好高端，完全不知道是什么鬼，那么就让我们退一步，先看看马尔科夫链。
马尔可夫链，因安德烈·马尔可夫（A.A.Markov，1856 - 1922）得名（就是下面这家伙），是指数学中具有马尔可夫性质的离散事件随机过程。在给定当前知识或信息的情况下，过去（即当前以前的历史状态）对于预测将来（即当前以后的未来状态）是无关的。


马尔可夫，A. A.

该过程中，每个状态的转移只依赖于之前的n个状态，这个过程被称为1个n阶的模型，其中n是影响转移状态的数目。最简单的马尔科夫过程就是一阶过程，每一个状态的转移只依赖于其之前的那一个状态。
用数学表达式表示就是下面的样子：
$P(X_{n+1} = x|X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = P(X_{n+1} = x|X_n = x_n)$. 举一个日常生活中的例子，我们希望根据当前天气的情况来预测未来天气情况。一种办法就是假设这个模型的每个状态都只依赖于前一个的状态，即马尔科夫假设，这个假设可以极大简化这个问题。当然，这个例子也是有些不合实际的。但是，这样一个简化的系统可以有利于我们的分析，所以我们通常接受这样的假设，因为我们知道这样的系统能让我们获得一些有用的信息，尽管不是十分



准确的。 上面的图显示了天气进行转移的模型。
注意一个含有N个状态的一阶过程有N^2个状态转移。每一个转移的概率叫做状态转移概率，就是从一个状态转移到另一个状态的概率。这所有的N^2个概率可以用一个状态转移矩阵来表示，上面天气例子的状态转移矩阵如下：

$$\begin{array}{cc} & \begin{array}{ccc} & Today & \\ sun & cloud & rain \end{array} \\ \begin{array}{c} Yesterday \\ \\ \end{array} & \begin{array}{c} sun \\ cloud \\ rain \end{array} \begin{bmatrix} 0.50 & 0.375 & 0.125 \\ 0.25 & 0.125 & 0.625 \\ 0.25 & 0.375 & 0.375 \end{bmatrix} \end{array}$$

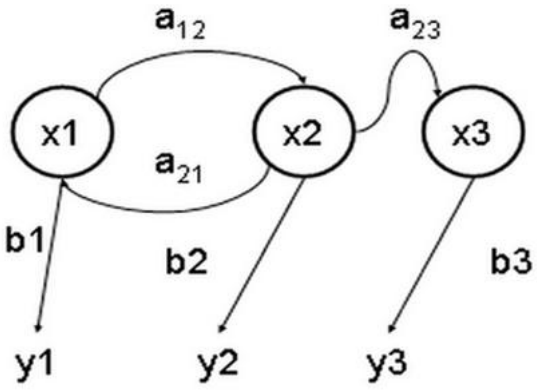
这个矩阵表示，如果昨天是阴天，那么今天有25%的可能是晴天，12.5%的概率是阴天，62.5%的概率会下雨，很明显，矩阵中每一行的和都是1。

Sun	Cloud	Rain
1.0	0.0	0.0

为了初始化这样一个系统，我们需要一个初始的概率向量：
这个向量表示第一天是晴天。到这里，我们就为上面的一阶马尔科夫过程定义了以下三个部分：

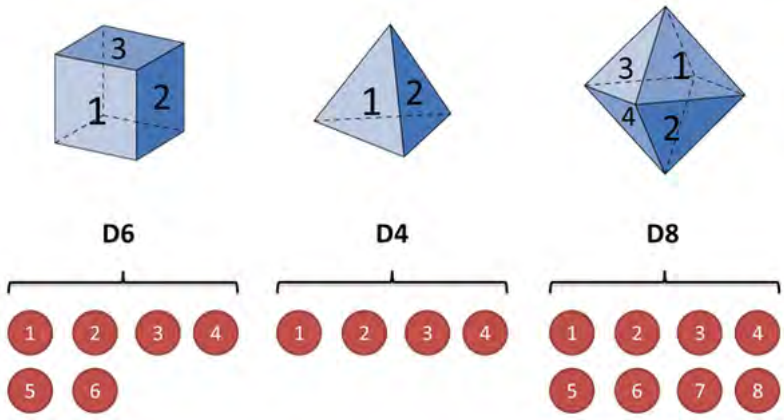
1. 状态：晴天、阴天和下雨。
2. 初始向量：定义系统在时间为0的时候的状态的概率。
3. 状态转移矩阵：每种天气转换的概率。所有的能被这样描述的系统都是一个马尔科夫过程。

然而，当马尔科夫过程不够强大的时候，我们又该怎么办呢？在某些情况下，马尔科夫过程不足以描述我们希望发现的模式。
比如我们的股市，如果只是观测市场，我们只能知道当天的价格、成交量等信息，但是并不知道当前股市处于什么样的状态（牛市、熊市、震荡、反弹等等），在这种情况下我们有两个状态集合，一个可以观察到的状态集合（股市价格成交量状态等）和一个隐藏的状态集合（股市状况）。我们希望能找到一个算法可以根据股市价格成交量状况和马尔科夫假设来预测股市的状况。
在上面的这些情况下，可以观察到的状态序列和隐藏的状态序列是概率相关的。于是我们可以将这种类型的过程建模为有一个隐藏的马尔科夫过程和一个与这个隐藏马尔科夫过程概率相关的并且可以观察到的状态集合，就是隐马尔可夫模型。
隐马尔可夫模型(Hidden Markov Model) 是一种统计模型，用来描述一个含有隐含未知参数的马尔可夫过程。其难点是从可观察的参数中确定该过程的隐含参数，然后利用这些参数来作进一步的分析。下图是一个三个状态的隐马尔可夫模型状态转移图，其中x表示隐含状态，y表示可观察的输出，a表示状态转换

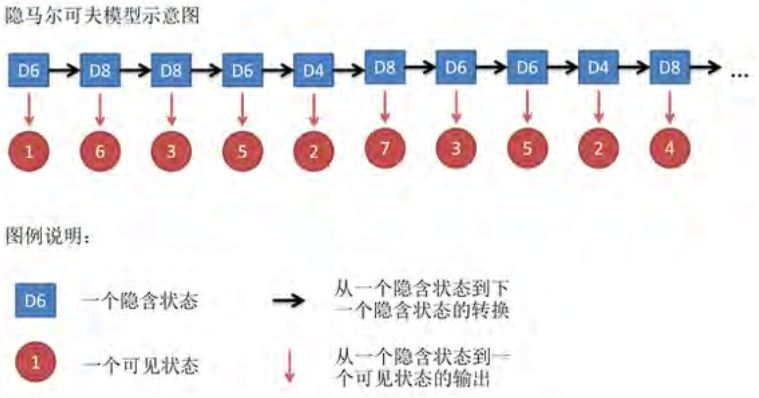


概率，b表示输出概率。

用一个掷筛子的例子阐述一下：假设我手里有三个不同的骰子。第一个骰子是我们平常见的骰子（称这个骰子为D6），6个面，每个面（1，2，3，4，5，6）出现的概率是1/6。第二个骰子是个四面体（称这个骰子为D4），每个面（1，2，3，4）出现的概率是1/4。第三个骰子有八个面（称这个骰子为D8），每个面（1，2，3，4，5，6，7，8）出现的概率是1/8。

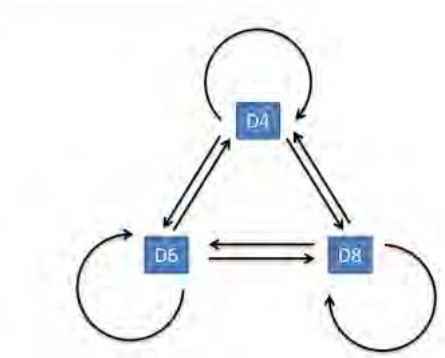


假设我们开始掷骰子，我们先从三个骰子里挑一个，挑到每一个骰子的概率都是1/3。然后我们掷骰子，得到一个数字，1，2，3，4，5，6，7，8中的一个。不停的重复上述过程，我们会得到一串数字，每个数字都是1，2，3，4，5，6，7，8中的一个。例如我们可能得到这么一串数字（掷骰子10次）：1 6 3 5 2 7 3 5 2 4
这串数字叫做可见状态链。但是在隐马尔可夫模型中，我们不仅仅有这么一串可见状态链，还有一串隐含状态链。在这个例子里，这串隐含状态链就是你用的骰子的序列。比如，隐含状态链有可能是：D4 D6 D8 D6 D4 D8 D6 D6 D6 D4。
一般来说，HMM中说到的马尔可夫链其实是指隐含状态链，因为隐含状态（骰子）之间存在转换概率。在我们这个例子里，D6的下一个状态是D4，D6，D8的概率都是1/3。D4，D8的下一个状态是D4，D6，D8的转换概率也都一样是1/3。这样设定是为了最开始容易说清楚，但是我们其实是可以随意设定转换概率的。比如，我们可以这样定义，D6后面不能接D4，D6后面是D6的概率是0.9，是D8的概率是0.1。这样就是一个新的HMM。
同样的，尽管可见状态之间没有转换概率，但是隐含状态和可见状态之间有一个概率叫做输出概率。就我们的例子来说，六面骰子（D6）产生1的输出概率是1/6。产生2，3，4，5，6的概率也都是1/6。我们同样可以对输出概率进行其他定义。比如，我有一个被赌场动过手脚的六面骰子，掷出来是1的概率更



大，是1/2，掷出来是2，3，4，5，6的概率是1/10。

隐含状态转换关系示意图



其实对于HMM来说，如果提前知道所有隐含状态之间的转换概率和所有隐含状态到所有可见状态之间的输出概率，做模拟是相当容易的。但是应用HMM模型时候呢，往往是缺失了一部分信息的，有时候你知道骰子有几种，每种骰子是什么，但是不知道掷出来的骰子序列；有时候你只是看到了很多次掷骰子的结果，剩下的什么都不知道。如果应用算法去估计这些缺失的信息，就成了一个很重要的问题。
和HMM模型相关的算法主要分为三类，分别解决三种问题：

1. 知道骰子有几种（隐含状态数量），每种骰子是什么（转换概率），根据掷骰子掷出的结果（可见状态链），我想知道每次掷出来的都是哪种骰子（隐含状态链）。
2. 还是知道骰子有几种（隐含状态数量），每种骰子是什么（转换概率），根据掷骰子掷出的结果（可见状态链），我想知道掷出这个结果的概率。
3. 知道骰子有几种（隐含状态数量），不知道每种骰子是什么（转换概率），观测到很多次掷骰子的结果（可见状态链），我想反推出每种骰子是什么（转换概率）。
   如果要解决上面股市中的问题，我们就需要解决问题1和问题3，下面我们就看看如何实现。

HMM在股票市场中的应用¶

我们假设隐藏状态数量是6，即假设股市的状态有6种，虽然我们并不知道每种状态到底是什么，但是通过后面的图我们可以看出那种状态下市场是上涨的，哪种是震荡的，哪种是下跌的。可观测的特征状态我们选择了3个指标进行标示，进行预测的时候假设假设所有的特征向量的状态服从高斯分布，这样就可以使用 hmmlearn 这个包中的 GaussianHMM 进行预测了。下面我会逐步解释。

首先导入必要的包：

In [1]:

```
from hmmlearn.hmm import GaussianHMM
import numpy as np
from matplotlib import cm, pyplot as plt
import matplotlib.dates as dates
import pandas as pd
import datetime
```

测试时间从2005年1月1日到2015年12月31日，拿到每日沪深300的各种交易数据。

In [2]:

```
beginDate = '2005-01-01'
endDate = '2015-12-31'
n = 6 #6个隐藏状态
data = get_price('CSI300.INDX',start_date=beginDate, end_date=endDate,frequency='1d')
data[0:9]
```

Out[2]:

	ClosingPx	HighPx	LowPx	NextTradingDate	OpeningPx	TotalTurnover	TotalVolumeTraded
MDEntryDate							
2005-01-04	982.79	994.77	980.66	20050105	994.77	4431976960	7412869
2005-01-05	992.56	997.32	979.88	20050106	981.58	4529207808	7119109
2005-01-06	983.17	993.79	980.33	20050107	993.33	3921015040	6288028
2005-01-07	983.96	995.71	979.81	20050110	983.05	4737468928	7298694
2005-01-10	993.88	993.96	979.79	20050111	983.76	3762931968	5791697
2005-01-11	997.14	999.55	991.09	20050112	994.19	3704076032	5849079
2005-01-12	996.75	996.98	989.26	20050113	996.65	3093298944	5014525
2005-01-13	996.88	999.47	992.70	20050114	996.08	3842172928	6044065
2005-01-14	988.31	1006.46	987.23	20050117	996.62	4162920960	7297842

拿到每日成交量和收盘价的数据。

In [3]:

```
volume = data['TotalVolumeTraded']
close = data['ClosingPx']
```

计算每日最高最低价格的对数差值，作为特征状态的一个指标。

In [4]:

```
logDel = np.log(np.array(data['HighPx'])) - np.log(np.array(data['LowPx']))
logDel
```

Out[4]:

```
array([ 0.01428574,  0.01764157,  0.01363667, ...,  0.01380317,
        0.01051247,  0.01207808])
```

计算每5日的指数对数收益差，作为特征状态的一个指标。

In [5]:

```
logRet_1 = np.array(np.diff(np.log(close)))#这个作为后面计算收益使用
logRet_5 = np.log(np.array(close[5:])) - np.log(np.array(close[:-5]))
logRet_5
```

Out[5]:

```
array([ 0.01449572,  0.00421252,  0.01384836, ..., -0.03007529,
       -0.02652464, -0.02603115])
```

计算每5日的指数成交量的对数差，作为特征状态的一个指标。

In [6]:

```
logVol_5 = np.log(np.array(volume[5:])) - np.log(np.array(volume[:-5]))
logVol_5
```

Out[6]:

```
array([-0.23693333, -0.35044388, -0.03957071, ..., -0.57079226,
       -0.67285963, -0.36793342])
```

由于计算中出现了以5天为单位的计算，所以要调整特征指标的长度。

In [7]:

```
logDel = logDel[5:]
logRet_1 = logRet_1[4:]
close = close[5:]
Date = pd.to_datetime(data.index[5:])
```

把我们的特征状态合并在一起。

In [8]:

```
A = np.column_stack([logDel,logRet_5,logVol_5])
A
```

Out[8]:

```
array([[ 0.00849983,  0.01449572, -0.23693333],
       [ 0.00777352,  0.00421252, -0.35044388],
       [ 0.00679663,  0.01384836, -0.03957071],
       ...,
       [ 0.01380317, -0.03007529, -0.57079226],
       [ 0.01051247, -0.02652464, -0.67285963],
       [ 0.01207808, -0.02603115, -0.36793342]])
```

下面运用 hmmlearn 这个包中的 GaussianHMM 进行预测。

In [9]:

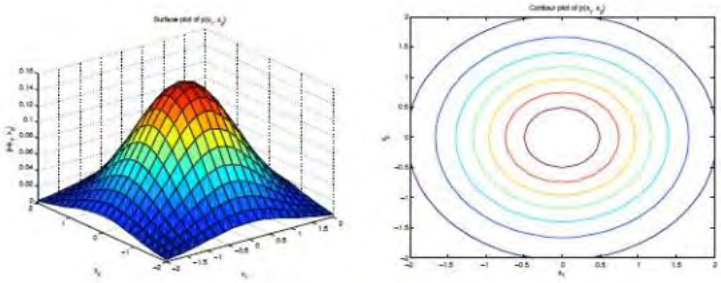
```
model = GaussianHMM(n_components= n, covariance_type="full", n_iter=2000).fit([A])
hidden_states = model.predict(A)
hidden_states
```

Out[9]:

```
array([0, 0, 0, ..., 3, 3, 3])
```

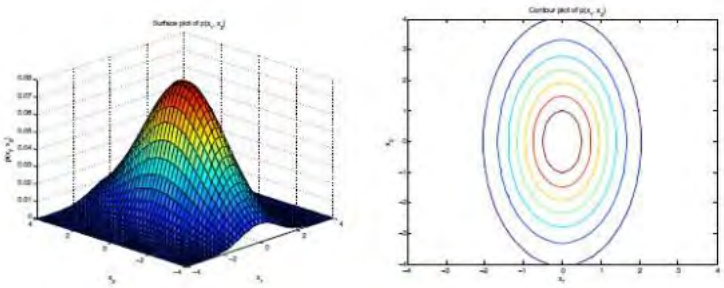
关于 covariance_type 的参数有下面四种：

spherical：是指在每个马尔可夫隐含状态下，可观察态向量的所有特性分量使用相同的方差值。对应协方差矩阵的非对角为0，对角值相等，即球面特性。这



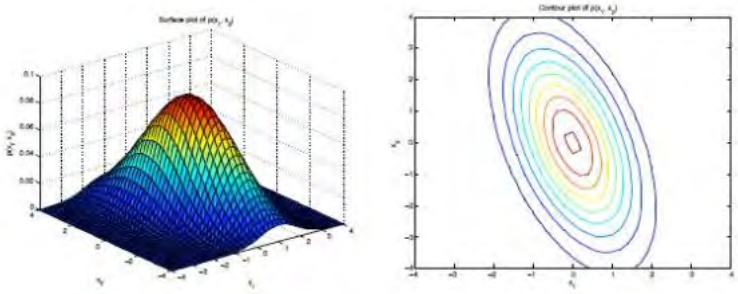
$$\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \rho_{12} = 0$$

是最简单的高斯分布PDF。 diag：是指在每个马尔可夫隐含状态下，可观察态向量使用对角协方差矩阵。对应协方差矩阵非对角为0，对角值不相等。diag是hmmlearn里面的默认类型。



$$\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \quad \rho_{12} = 0$$

full：是指在每个马尔可夫隐含状态下，可观察态向量使用完全协方差矩阵。对应的协方差矩阵里面的元素都是不为零。



$$\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \Sigma = \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix} \quad \rho_{12} = -0.5$$

tied：是指所有的马尔可夫隐含状态使用相同的完全协方差矩阵。

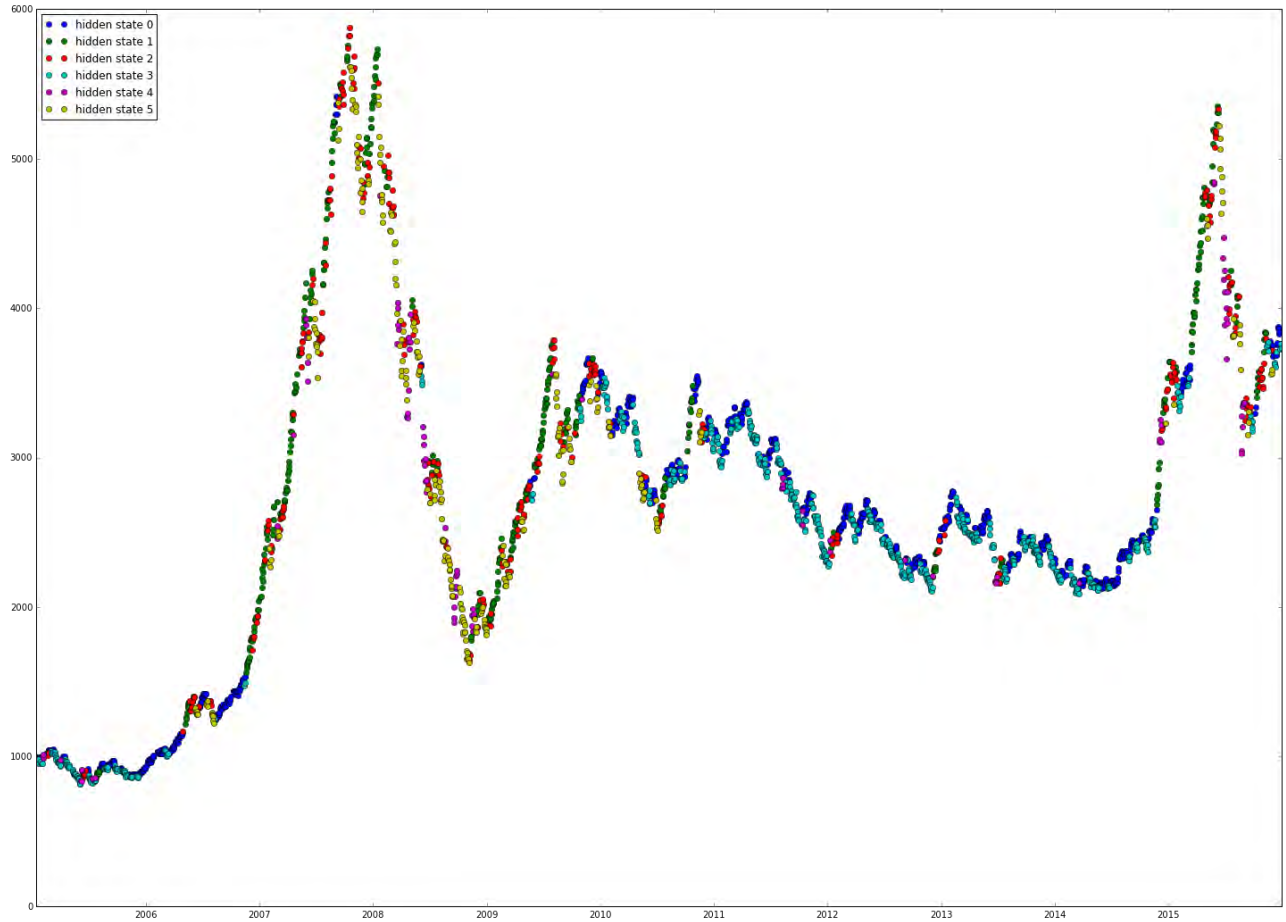
这四种PDF类型里面，spherical, diag和full代表三种不同的高斯分布概率密度函数，而tied则可以看作是GaussianHMM和GMMHMM的特有实现。其中，full是最强大的，但是需要足够多的数据来做合理的参数估计；spherical是最简单的，通常用在数据不足或者硬件平台性能有限的情况之下；而diag则是这两者一个折中。在使用的时候，需要根据可观察态向量不同特性的相关性来选择合适的类型。

转自知乎用户Aubrey Li

我们把每个预测的状态用不同颜色标注在指数曲线上看一下结果。

In [10]:

```
plt.figure(figsize=(25, 18))
for i in range(model.n_components):
    pos = (hidden_states==i)
    plt.plot_date(Date[pos],close[pos],'o',label='hidden state %d'%i,lw=2)
    plt.legend(loc="left")
```

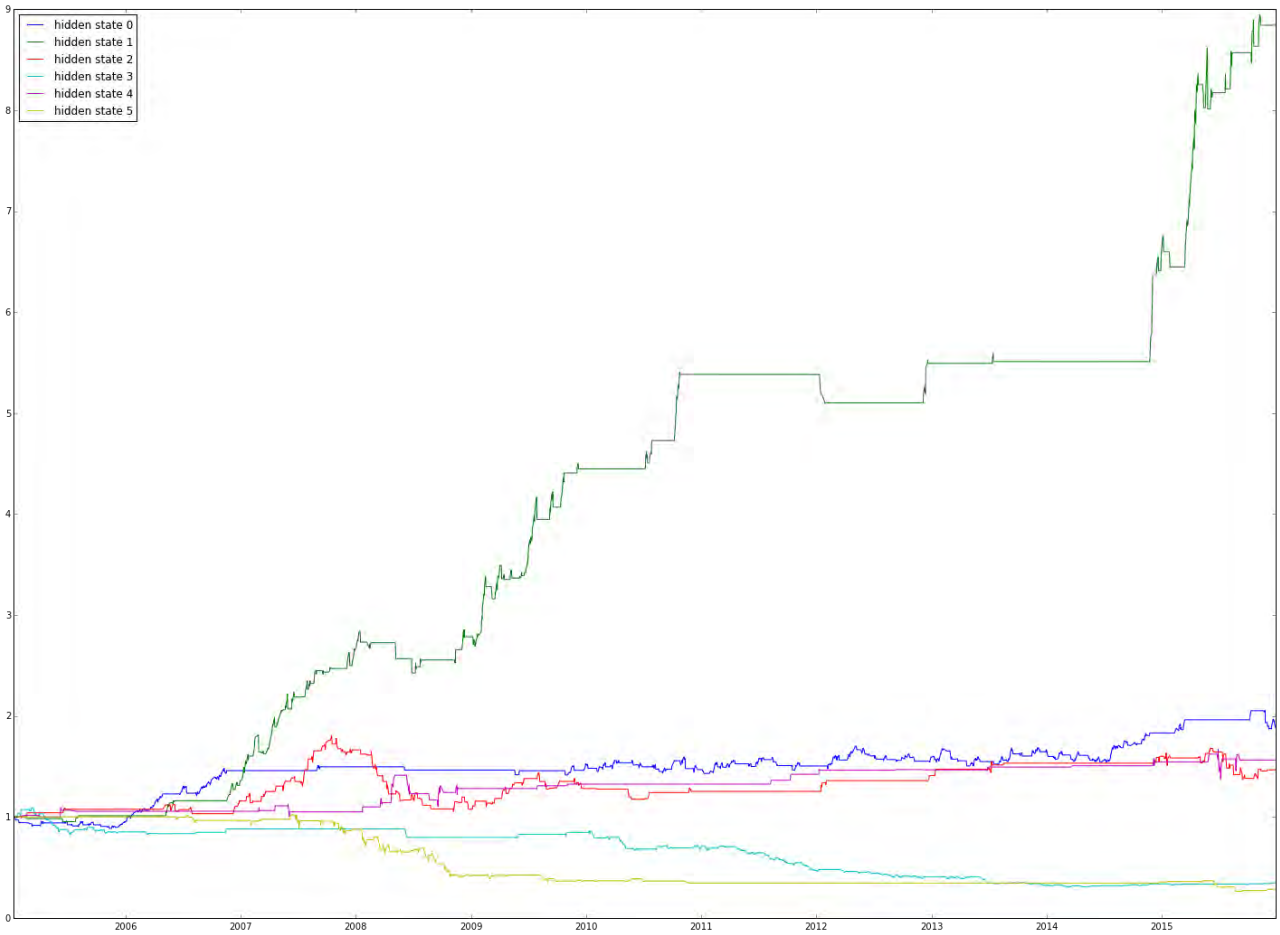


从图中可以比较明显的看出绿色的隐藏状态代表指数大幅上涨，浅蓝色和黄色的隐藏状态代表指数下跌。

为了更直观的表现不同的隐藏状态分别对应了什么，我们采取获得隐藏状态结果后第二天进行买入的操作，这样可以看出每种隐藏状态代表了什么。

In [11]:

```
res = pd.DataFrame({'Date':Date,'logRet_1':logRet_1,'state':hidden_states}).set_index('Date')
plt.figure(figsize=(25, 18))
for i in range(model.n_components):
    pos = (hidden_states==i)
    pos = np.append(0,pos[:-1])#第二天进行买入操作
    df = res.logRet_1
    res['state_ret%s'%i] = df.multiply(pos)
    plt.plot_date(Date,np.exp(res['state_ret%s'%i].cumsum()),'-',label='hidden state %d'%i)
    plt.legend(loc="left")
```



可以看到，隐藏状态1是一个明显的大牛市阶段，隐藏状态0是一个缓慢上涨的阶段(可能对应反弹)，隐藏状态3和5可以分别对应震荡下跌的大幅下跌。其他的两个隐藏状态并不是很明确。由于股指期货可以做空，我们可以进行如下操作：当处于状态0和1时第二天做多，当处于状态3和5第二天做空，其余状态则不持有。

In [12]:

```
long = (hidden_states==0) + (hidden_states == 1) #做多
short = (hidden_states==3) + (hidden_states == 5)  #做空
long = np.append(0,long[:-1]) #第二天才能操作
short = np.append(0,short[:-1]) #第二天才能操作
```

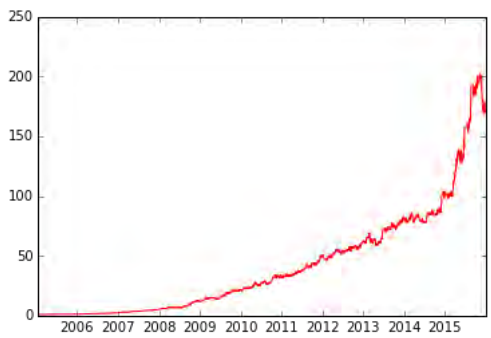
收益曲线图如下：

In [13]:

```
res['ret'] =  df.multiply(long) - df.multiply(short)
plt.plot_date(Date,np.exp(res['ret'].cumsum()),'r-')
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x7fb1b67bec50>]
```



可以看到效果还是很不错的。但事实上该结果是有些问题的。真实操作时，我们并没有未来的信息来训练模型。不过可以考虑用历史数据进行训练，再对之后的数据进行预测。

时间序列波动率建模及预测¶

本文是对https://www.ricequant.com/community/topic/992/garch模型-对波动率建模 的修改和完善，追踪quantopian的原文，有的地方说的不清楚（不完善），所以给大家一个完整的时间序列波动率建模过程（如有错误请指出）。 在时间序列中，a_t称为新信息或者扰动；在原文中的Garch(1,1)模型的新信息为x_t，我们的波动率模型 是对新信息的σ_t^2建模，我们用蒙特卡洛方法模拟出一个GARCH(1,1)过程，变量的动态变化过程如原公式，此外原公式省略了 $r_t=\mu+a_t$这个方程，并且假设μ等于零，也是为了标准化收益率做铺垫，从而直接用模拟生成的a_t也就是x_t来寻求异常值

In [55]:

```
import cvxopt
from functools import partial
import math
import numpy as np
import scipy
from scipy import stats
import statsmodels as sm
from statsmodels import regression
from statsmodels.stats.stattools import jarque_bera

import matplotlib.pyplot as plt
```

In [56]:

```
# 参数初始化
a0 = 1.0
a1 = 0.1
b1 = 0.8
sigma1 = math.sqrt(a0 / (1 - a1 - b1))
```

In [57]:

```
def simulate_GARCH(T, a0, a1, b1, sigma1):

    # 初始化
    X = np.ndarray(T)
    sigma = np.ndarray(T)
    sigma[0] = sigma1

    for t in range(1, T):
        # 产生下一个Xt
        X[t - 1] = sigma[t - 1] * np.random.normal(0, 1)
        # 产生下一个sigma_t
        sigma[t] = math.sqrt(a0 + b1 * sigma[t - 1]**2 + a1 * X[t - 1]**2)

    X[T - 1] = sigma[T - 1] * np.random.normal(0, 1)

    return X, sigma
```

In [58]:

```
X, _ = simulate_GARCH(10000, a0, a1, b1, sigma1) #生成模拟序列
X = X[1000:] # Drop burn in
X = X / np.std(X) # 标准化X

def compare_tails_to_normal(X):
    # Define matrix to store comparisons
    A = np.zeros((2,4))
    for k in range(4):
        A[0, k] = len(X[X > (k + 1)]) / float(len(X)) # 估计X的尾部，将X异常值得比率求出来
        A[1, k] = 1 - stats.norm.cdf(k + 1) # 正态分布的尾部的累计分布，与上面的值比较，上面的应该大于下面的表示异常值多
    return A

compare_tails_to_normal(X)#(而我们看到大部分确实是这样)
```

Out[58]:

```
array([[  1.54666667e-01,   2.53333333e-02,   1.44444444e-03,
          1.11111111e-04],
       [  1.58655254e-01,   2.27501319e-02,   1.34989803e-03,
          3.16712418e-05]])
```

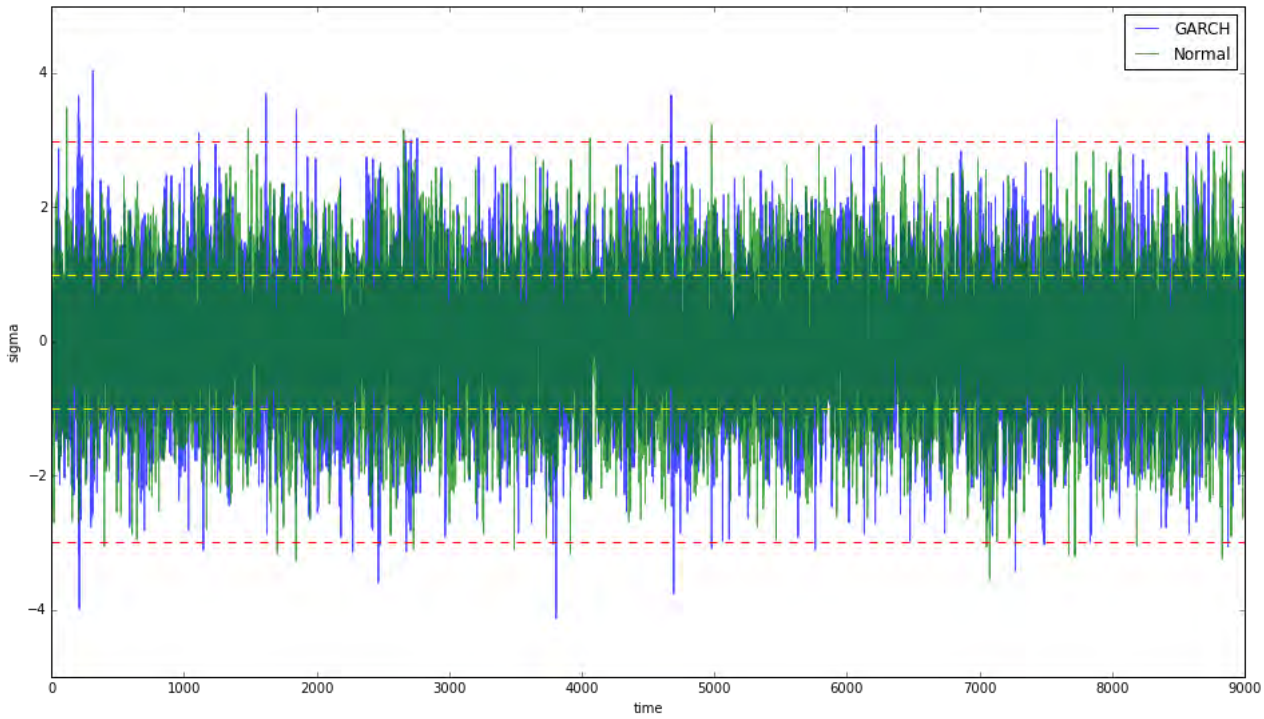
In [59]:

```
X2 = np.random.normal(0, 1, 9000)
# 将两者存到一个矩阵中
both = np.matrix([X, X2])
#同时做出我们通过Garch(1,1)模拟生成的序列X及正态序列的收益时序图
plt.figure(figsize=(16, 9))
plt.plot(both.T, alpha=.7);
plt.axhline(X2.std(), color='yellow', linestyle='--')
plt.axhline(-X2.std(), color='yellow', linestyle='--')
```

```
plt.axhline(3*X2.std(), color='red', linestyle='--')
plt.axhline(-3*X2.std(), color='red', linestyle='--')
plt.xlabel('time')
plt.ylabel('sigma')
plt.legend(["GARCH","Normal"])
```

Out[59]:

```
<matplotlib.legend.Legend at 0x7f34d6dea0b8>
```



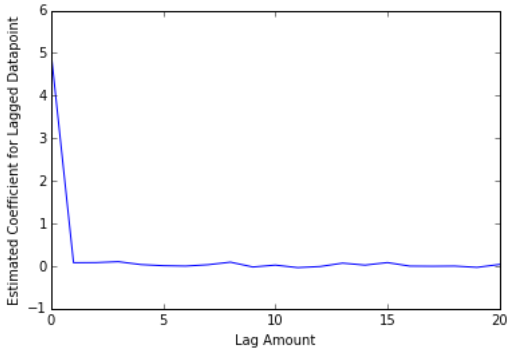
上面我们用蒙特卡洛模拟，模拟出了(收益率）序列a_t（r_t），下来我们就要假装我们不知道一个序列是不是满足Garch（1,1），来进行拟合，如原文所提到的，先进行Arch检验：Arch效应的检验有两种方法，第一种是Ljung-Box统计量用于序列a_t^2，即检验残差是否相关。第二种是Engel(1982)提出的拉格朗日乘子检验。该检验等价于线性回归中用F统计量检验$\alpha_i=0(i=1,2...m)$，检验方程为$a_t^2= \alpha_0+ \alpha_1a_{t-1}^2+...+\alpha_ma_{t-m}^2+e_t,t=m+1,...T$ Quantopian原文中使用的应该是Ljung-Box检验的变种(因为我没找到完全吻合的- - ！），P值越小越显著，Arch效应越显著，可以对波动率建模

In [60]:

```
X, _ = simulate_GARCH(1100, a0, a1, b1, sigma1)
X = X[100:] # Drop burn in

p = 20

# Drop the first 20 so we have a lag of p's
Y2 = (X**2)[p:]
X2 = np.ndarray((980, p))
for i in range(p, 1000):
    X2[i - p, :] = np.asarray((X**2)[i-p:i])[::-1]
con=np.empty((980,1))
for i in range(980):
    con[i,:]=1
X2=np.hstack((con,X2))
model = sm.regression.linear_model.OLS(Y2, X2)#原文中并没有拟合常数项，但是公式中显然用到了，到底拟不拟合计入F呢？
#大家还是按上面介绍的第二种方法做稳妥，这里我添加了，否则就跑偏了
model = model.fit()
theta = np.matrix(model.params)
omega = np.matrix(model.cov_HC0)
F = np.asscalar(theta * np.linalg.inv(omega) * theta.T)

#print np.asarray(theta.T).shape
print(theta)
plt.plot(range(21), np.asarray(theta.T))
plt.xlabel('Lag Amount')
plt.ylabel('Estimated Coefficient for Lagged Datapoint')

print ('F ='+ str(F))

chi2dist = scipy.stats.chi2(p)
pvalue = 1-chi2dist.cdf(F)
print ('p-value = ' + str(pvalue))#Pvalue is small,Arch condition is met.
```

```
# Finally let's look at the significance of each a_p as measured by the standard deviations away from 0
```

```
[[  5.00398970e+00   7.91909523e-02   8.16054764e-02   1.02422380e-01
    3.48122490e-02   8.52583813e-03  -3.48065747e-04   3.11362759e-02
    9.14522132e-02  -2.17775768e-02   2.22012631e-02  -3.66738314e-02
   -1.06271844e-02   6.83422118e-02   2.22335623e-02   8.14950773e-02
   -1.10637908e-03  -4.52856591e-03  -5.49474538e-04  -3.03768472e-02
    3.76003225e-02]]
F =572.2960640396009
p-value = 0.0
```



接下来就是用MLE和GMM方法来估计波动率方程$\sigma_t^2=\alpha_0+\alpha_1a_{t-1}^2+ b_1a_{t-2}^2$参数值，同理，我们先模拟出波动率

In [61]:

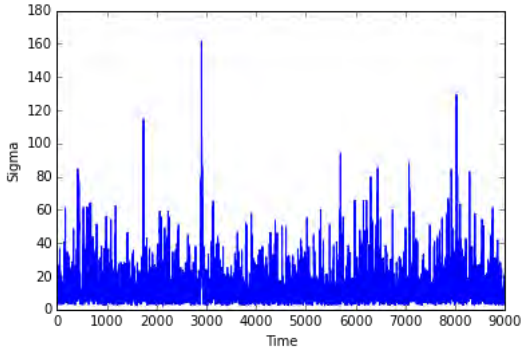
```
X, _ = simulate_GARCH(10000, a0, a1, b1, sigma1)
X = X[1000:] # Drop burn in
```

In [62]:

```
def compute_squared_sigmas(X, initial_sigma, theta):#生成sigma序列

    a0 = theta[0]
    a1 = theta[1]
    b1 = theta[2]

    T = len(X)
    sigma2 = np.ndarray(T)

    sigma2[0] = initial_sigma ** 2

    for t in range(1, T):
        # Here's where we apply the equation
        sigma2[t] = a0 + a1 * X[t-1]**2 + b1 * sigma2[t-1]

    return sigma2
```

In [63]:

```
plt.plot(range(len(X)), compute_squared_sigmas(X, np.sqrt(np.mean(X**2)), (1, 0.5, 0.5)))#这里的a0,a1,b1并不和我们初始一样，
#就是为了造成误差，否则就完全拟合了
plt.xlabel('Time')
plt.ylabel('Sigma');
```



In [64]:

```
def negative_log_likelihood(X, theta):

    T = len(X)
```

```
    # Estimate initial sigma squared
    initial_sigma = np.sqrt(np.mean(X ** 2))

    # Generate the squared sigma values
    sigma2 = compute_squared_sigmas(X, initial_sigma, theta)

    # Now actually compute
    return -sum(
        [-np.log(np.sqrt(2.0 * np.pi)) -
        (X[t] ** 2) / (2.0 * sigma2[t]) -
        0.5 * np.log(sigma2[t]) for
         t in range(T)]
    )
```

In [65]:

```
# 进行MLE估计，求出估计值
objective = partial(negative_log_likelihood, X)
# Define the constraints for our minimizer
def constraint1(theta):
    return np.array([1 - (theta[1] + theta[2])])

def constraint2(theta):
    return np.array([theta[1]])

def constraint3(theta):
    return np.array([theta[2]])

cons = ({'type': 'ineq', 'fun': constraint1},
        {'type': 'ineq', 'fun': constraint2},
        {'type': 'ineq', 'fun': constraint3})

# Actually do the minimization
result = scipy.optimize.minimize(objective, (1, 0.5, 0.5),
                         method='SLSQP',
                         constraints = cons)
theta_mle = result.x
print ('theta MLE: ' + str(theta_mle))
```

```
theta MLE: [ 1.08330365  0.09764981  0.79638653]
```

估计值已经出来，那估计的效果怎么样呢，原文用两个效果衡量，并且都是绝对比较： 1.How fat are the tails of the residuals 也就是收益率x_t，或者a_t. 2.How normal are the residuals under the Jarque-Bera normality test（从下面的P-value很大可以看出，无法拒绝正太性原假设） 但是绝对比较并不符合统计要求，完整的做法是对$\{a_t\}$做Ljung-Box,对$\{a_t^2\}$做Ljung-Box以及保证估计参数的Pvalue很小

In [66]:

```
def check_theta_estimate(X, theta_estimate):
    initial_sigma = np.sqrt(np.mean(X ** 2))
    sigma = np.sqrt(compute_squared_sigmas(X, initial_sigma, theta_estimate))
    epsilon = X / sigma
    print ('Tails table')
    print (compare_tails_to_normal(epsilon / np.std(epsilon)))
    print ( '')

    _, pvalue, _, _ = jarque_bera(epsilon)
    print ('Jarque-Bera probability normal: ' + str(pvalue))

check_theta_estimate(X, theta_mle)
```

```
Tails table
[[  1.54888889e-01   2.22222222e-02   2.00000000e-03   1.11111111e-04]
 [  1.58655254e-01   2.27501319e-02   1.34989803e-03   3.16712418e-05]]

Jarque-Bera probability normal: 0.218225625935
```

除了用MLE估计，还可用GMM估计，具体原理见原文（思想是迭代）

In [67]:

```
#定义求阶矩公式
def standardized_moment(x, mu, sigma, n):
    return ((x - mu) ** n) / (sigma ** n)
```

In [68]:

```
def gmm_objective(X, W, theta):
    # Compute the residuals for X and theta
```

```
    initial_sigma = np.sqrt(np.mean(X ** 2))
    sigma = np.sqrt(compute_squared_sigmas(X, initial_sigma, theta))
    e = X / sigma

    # Compute the mean moments
    m1 = np.mean(e)
    m2 = np.mean(e ** 2) - 1
    m3 = np.mean(standardized_moment(e, np.mean(e), np.std(e), 3))
    m4 = np.mean(standardized_moment(e, np.mean(e), np.std(e), 4) - 3)

    G = np.matrix([m1, m2, m3, m4]).T

    return np.asscalar(G.T * W * G)

def gmm_variance(X, theta):
    # Compute the residuals for X and theta
    initial_sigma = np.sqrt(np.mean(X ** 2))
    sigma = np.sqrt(compute_squared_sigmas(X, initial_sigma, theta))
    e = X / sigma

    # Compute the squared moments
    m1 = e ** 2
    m2 = (e ** 2 - 1) ** 2
    m3 = standardized_moment(e, np.mean(e), np.std(e), 3) ** 2
    m4 = (standardized_moment(e, np.mean(e), np.std(e), 4) - 3) ** 2

    # Compute the covariance matrix g * g'
    T = len(X)
    s = np.ndarray((4, 1))
    for t in range(T):
        G = np.matrix([m1[t], m2[t], m3[t], m4[t]]).T
        s = s + G * G.T

    return s / T
```

In [69]:

```
# Initialize GMM parameters
W = np.identity(4)
gmm_iterations = 10

# First guess
theta_gmm_estimate = theta_mle

# Perform iterated GMM
for i in range(gmm_iterations):
    # Estimate new theta
    objective = partial(gmm_objective, X, W)
    result = scipy.optimize.minimize(objective, theta_gmm_estimate, constraints=cons)
    theta_gmm_estimate = result.x
    print ('Iteration ' + str(i) + ' theta: ' + str(theta_gmm_estimate))

    # Recompute W
    W = np.linalg.inv(gmm_variance(X, theta_gmm_estimate))


check_theta_estimate(X, theta_gmm_estimate)
```

```
Iteration 0 theta: [ 0.89961561  0.08336233  0.82815649]
Iteration 1 theta: [ 0.89962901  0.08348014  0.828286  ]
Iteration 2 theta: [ 0.89962901  0.08348014  0.828286  ]
Iteration 3 theta: [ 0.89962901  0.08348014  0.828286  ]
Iteration 4 theta: [ 0.89962901  0.08348014  0.828286  ]
Iteration 5 theta: [ 0.89962901  0.08348014  0.828286  ]
Iteration 6 theta: [ 0.89962901  0.08348014  0.828286  ]
Iteration 7 theta: [ 0.89962901  0.08348014  0.828286  ]
Iteration 8 theta: [ 0.89962901  0.08348014  0.828286  ]
Iteration 9 theta: [ 0.89962901  0.08348014  0.828286  ]
Tails table
[[  1.55111111e-01   2.25555556e-02   2.22222222e-03   1.11111111e-04]
 [  1.58655254e-01   2.27501319e-02   1.34989803e-03   3.16712418e-05]]

Jarque-Bera probability normal: 0.231900700185
```

既然我们已经估计出了模型参数，那么开始预测（装逼）吧!

In [70]:

```
sigma_hats = np.sqrt(compute_squared_sigmas(X, np.sqrt(np.mean(X**2)), theta_mle))
initial_sigma = sigma_hats[-1]
initial_sigma#拿出最后一个值，要从此值开始我们的预测之旅
```

Out[70]:

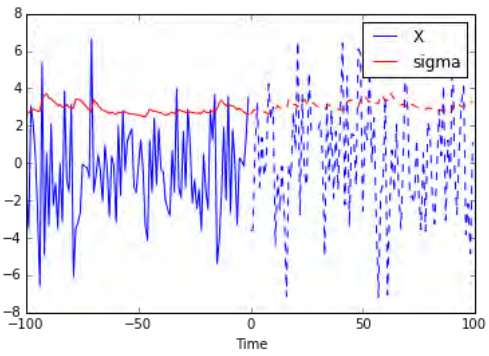
```
2.6194141066428118
```

In [71]:

```
a0_estimate = theta_gmm_estimate[0]
a1_estimate = theta_gmm_estimate[1]
b1_estimate = theta_gmm_estimate[2]
#这里用了GMM的估计值，差不了多少
X_forecast, sigma_forecast = simulate_GARCH(100, a0_estimate, a1_estimate, b1_estimate, initial_sigma)#这样就求出了预计收益率和预计波动率
```

In [72]:

```
plt.plot(range(-100, 0), X[-100:], 'b-')
plt.plot(range(-100, 0), sigma_hats[-100:], 'r-')
plt.plot(range(0, 100), X_forecast, 'b--')
plt.plot(range(0, 100), sigma_forecast, 'r--')
plt.xlabel('Time')
plt.legend(['X', 'sigma']);
```

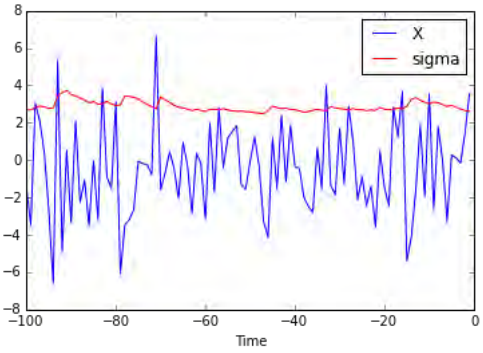


In [73]:

```
plt.plot(range(-100, 0), X[-100:], 'b-')
plt.plot(range(-100, 0), sigma_hats[-100:], 'r-')
plt.xlabel('Time')
plt.legend(['X', 'sigma'])
```

Out[73]:

```
<matplotlib.legend.Legend at 0x7f34de445240>
```

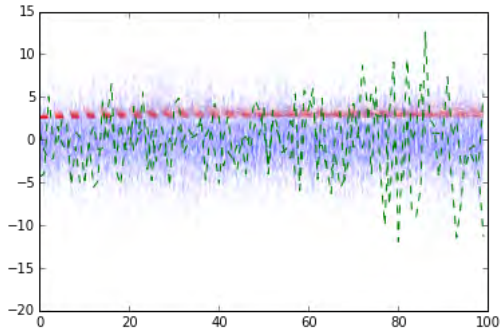


因为只是模拟了一次，显然不具备代表性，所以文章模拟了100次，并作出每次的拟合值及极值序列

In [74]:

```
max_X = [-np.inf]
min_X = [np.inf]
for i in range(100):
    X_forecast, sigma_forecast = simulate_GARCH(100, a0_estimate, a1_estimate, b1_estimate, initial_sigma)
    if max(X_forecast) > max(max_X):
        max_X = X_forecast
    elif min(X_forecast) < min(max_X):
        min_X = X_forecast
    plt.plot(range(0, 100), X_forecast, 'b--', alpha=0.05)
    plt.plot(range(0, 100), sigma_forecast, 'r--', alpha=0.05)

# Draw the most extreme X values specially
```

```
plt.plot(range(0, 100), max_X, 'g--', alpha=1.0)
plt.plot(range(0, 100), min_X, 'g--', alpha=1.0);
```



总结:时间序列波动率建模的顺序就是： 1.建立原文省略的均值方程（甚至可以用AR,MA,ARMA建立） 2.检验ARCH效应，如果显著，则对均值方程（收益率方程）和波动率方程建模 3.利用MLE（GMM)拟合 4.检查拟合效果

ARMA+GARCH 交易策略在沪深300指数上的应用¶

策略的思路很简单，这里简要介绍一下：

1. 对于每一天，利用前k天的收益率收据拟合一个最优的ARMA和GARCH模型
2. 利用联合构建的模型来预测下一天的收益
3. 如果预测的收益是负值，则在第二天开盘时卖空或者以当天的收盘价卖空，若为正，则买入 4.若预测值与收益率的前值方向相同，则不进行任何操作

逻辑叙述完后，继续按套路出牌，这里先只贴出研究板块的内容，如果有感兴趣的小伙伴，欢迎自己写回测，并补充出来，我自己后期也会将回测内容贴出来

In [1]:

```
import numpy as np
import pandas as pd
from scipy import optimize
import matplotlib.pyplot as plt
import datetime as dt
```

In [2]:

```
start = dt.datetime(2014,1,1)
end = dt.datetime(2015,12,31)
benchmark = '000300.XSHG'
prices = get_price(benchmark, start_date=start, end_date=end, fields='ClosingPx')
returns = pd.DataFrame(np.log(prices).diff().dropna())
returns.rename(columns={'ClosingPx':'Log Return Rate'}, inplace=True)
```

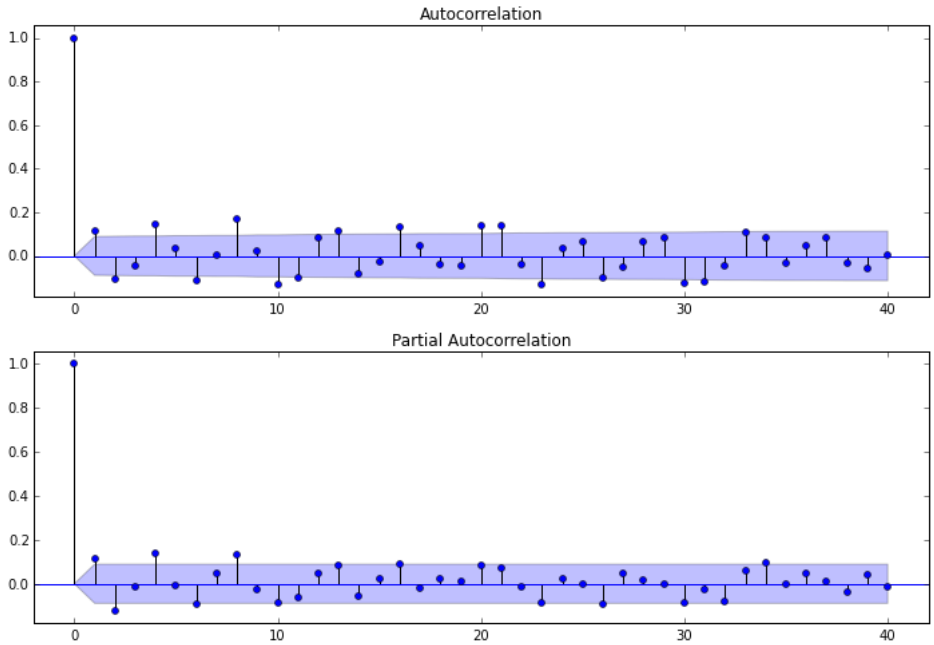
In [3]:

```
# 检验收益率序列的平稳性
from statsmodels.tsa.stattools import adfuller
returns_array_like = [x for l in returns.values for x in l]
_, pvalue, *_ = adfuller(returns_array_like)
print ('pvalue :',pvalue)
```

```
pvalue : 1.81283251617e-06
```

In [4]:

```
import statsmodels.api as sm

fig = plt.figure(figsize=(12, 8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(returns.values, lags=40, alpha=0.05, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(returns.values, lags=40, alpha=0.05, ax=ax2);
```



由于ARMA模型不能简单的根据ACF和PACF的截尾简单地判断，所以我们这里不依据图像确定阶数，而且即使依赖图形也无法很好地确定

In [5]:

```
import  statsmodels.tsa.stattools as sts
resid = sts.arma_order_select_ic(returns, max_ar=4, max_ma=4,
                                 ic=['aic','bic','hqic'], trend='nc', fit_kw=dict(method='css'))
print ('AIC-order :{}'.format(resid.aic_min_order))
print ('BIC-order :{}'.format(resid.bic_min_order))
print ('HQIC-order :{}'.format(resid.hqic_min_order))
```

```
/srv/env/lib64/python3.4/site-packages/statsmodels/base/model.py:466: ConvergenceWarning: Maximum Likelihood optimization failed to c
  "Check mle_retvals", ConvergenceWarning)
/srv/env/lib64/python3.4/site-packages/statsmodels/base/model.py:466: ConvergenceWarning: Maximum Likelihood optimization failed to c
  "Check mle_retvals", ConvergenceWarning)
```

```
AIC-order :(2, 4)
BIC-order :(0, 1)
HQIC-order :(2, 2)
```

```
/srv/env/lib64/python3.4/site-packages/statsmodels/base/model.py:466: ConvergenceWarning: Maximum Likelihood optimization failed to c
  "Check mle_retvals", ConvergenceWarning)
```

本着降低模型复杂度的考虑，我们依据BIC准则进行建模 通过BIC（贝叶斯信息准则），给出的最优阶数是(0, 1),为此我们知道沪深300指数收益率序列可以直接由MA(1)模型刻画

In [6]:

```
# 在确定好ARMA模型的阶数后，我们来估计其参数

arma_mod01 = sm.tsa.ARMA(returns, (0, 1)).fit()
print (arma_mod01.summary())
print ('-------------------------------------')
print (arma_mod01.params)
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:        Log Return Rate   No. Observations:                  488
Model:                     ARMA(0, 1)   Log Likelihood                1228.890
Method:                       css-mle   S.D. of innovations              0.020
Date:                Thu, 23 Jun 2016   AIC                          -2451.781
Time:                        07:33:55   BIC                          -2439.210
Sample:                    01-03-2014   HQIC                         -2446.843
                         - 12-31-2015
===============================================================================================
                              coef    std err          z      P>|z|      [95.0% Conf. Int.]
-----------------------------------------------------------------------------------------------
const                       0.0010      0.001      0.958      0.339        -0.001     0.003
ma.L1.Log Return Rate       0.1421      0.048      2.932      0.004         0.047     0.237
                                    Roots
```

```
=============================================================================
                 Real           Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
MA.1           -7.0369           +0.0000j            7.0369            0.5000
-----------------------------------------------------------------------------
-------------------------------------
const                    0.000966
ma.L1.Log Return Rate    0.142107
dtype: float64
```

In [7]:

```
#为了进一步检验所构建模型的合理性，我们这里对收益率的残差进行自相关性检验

from statsmodels.stats import diagnostic
resid = arma_mod01.resid
_, pvalue, _, bppvalue = diagnostic.acorr_ljungbox(resid, lags=None, boxpierce=True)
print (pvalue,'\n',bppvalue)
```

```
[  7.86913837e-01   9.40923337e-02   1.15160562e-01   2.74076362e-03
   5.15661433e-03   6.05217068e-04   1.31039535e-03   1.01137128e-05
   2.17039446e-05   2.36870399e-06   8.85104582e-07   5.15723912e-07
   7.76641755e-08   2.90609637e-08   5.01442272e-08   3.28355809e-09
   5.60514043e-09   9.27749759e-09   1.00660493e-08   8.41667489e-10
   8.02950108e-11   1.30687399e-10   9.12301415e-12   1.30035911e-11
   9.43602777e-12   2.13448298e-12   2.99055566e-12   3.00697071e-12
   1.15556805e-12   9.66836251e-14   3.26197179e-14   4.23771590e-14
   9.50122055e-15   6.75580902e-15   8.82649126e-15   1.29717752e-14
   6.62204909e-15   1.04197625e-14   1.25329849e-14   2.47097856e-14]
 [  7.87551456e-01   9.59181970e-02   1.17728833e-01   2.96487314e-03
   5.56887031e-03   6.86076813e-04   1.47706817e-03   1.29066655e-05
   2.75445899e-05   3.24383692e-06   1.26891780e-06   7.65637445e-07
   1.25439265e-07   4.96639691e-08   8.56709593e-08   6.42575957e-09
   1.09708777e-08   1.81755407e-08   2.01383478e-08   1.96251511e-09
   2.18268145e-10   3.55952596e-10   2.98957929e-11   4.30498559e-11
   3.29250766e-11   8.48508319e-12   1.20299351e-11   1.25263181e-11
   5.34913541e-12   5.59268806e-13   2.13339887e-13   2.82601088e-13
   7.45989048e-14   5.69470138e-14   7.59115904e-14   1.12644021e-13
   6.39320101e-14   1.00899389e-13   1.24846386e-13   2.41201630e-13]
```

我们发现残差基本是不相关的，尤其是在三阶以后，显著不相关，所以收益率模型的设定是比较合理的

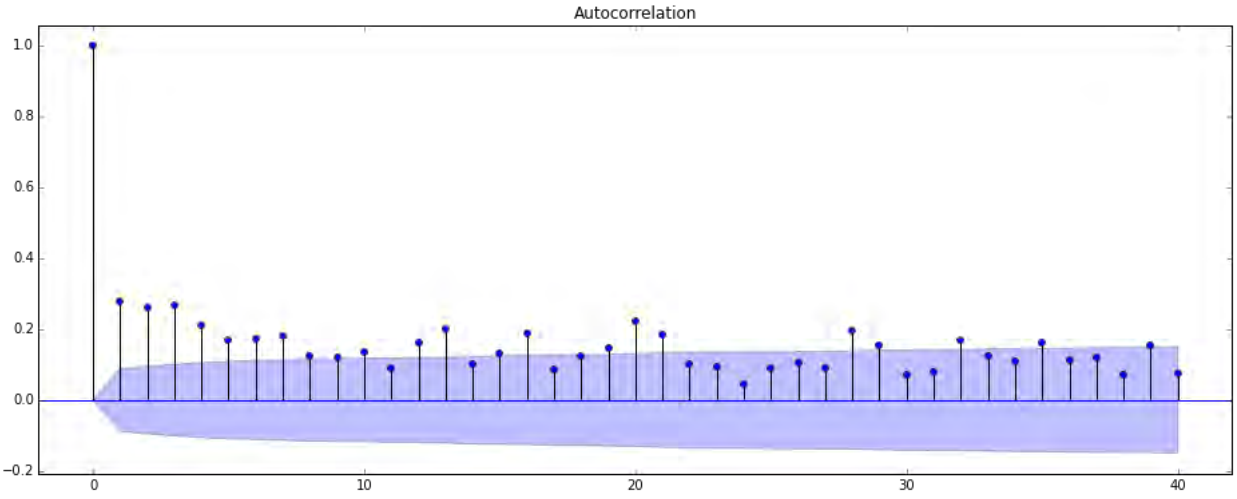
In [8]:

```
# 在检验完残差的自相关性后，我们现在来考虑残差是否具有ARCH效应，从而更好地对收益率的波动率建模

*_, fpvalue = diagnostic.het_arch(resid)
print (fpvalue)
```

```
6.48155789518e-12
```

In [9]:

```
fig = plt.figure(figsize=(16, 6))
ax1 = fig.add_subplot(111)
fig = sm.graphics.tsa.plot_acf(resid.values ** 2, lags=40, ax=ax1)
```



检验ARCH效应的原假设是不存在异方差性，根据检验结果我们可以发现收益率的残差项存在显著的异方差性，为了更直观地感受我们画出残差平方的自相关图

现在我们可以对收益率的波动率进行建模了，为了方便我们这里借助第三方模块来帮助我们实现想法

但是我们知道在金融时间序列分析中，资产收益率通常不是服从正态分布的，这里我们可以简单地检验一下

在此基础上，待会建模对分布进行设定时我们使用更加合理的学生t分布来取代默认的正太分布，当然学生t分布也并不能完全刻画资产收益率的尖峰厚尾现象

In [10]:

```
from statsmodels.stats.stattools import jarque_bera
_, jbpv, *_ = jarque_bera(returns.values)
print ('pvalue-->',jbpv)
```

```
pvalue--> [  5.30287710e-92]
```

In [11]:

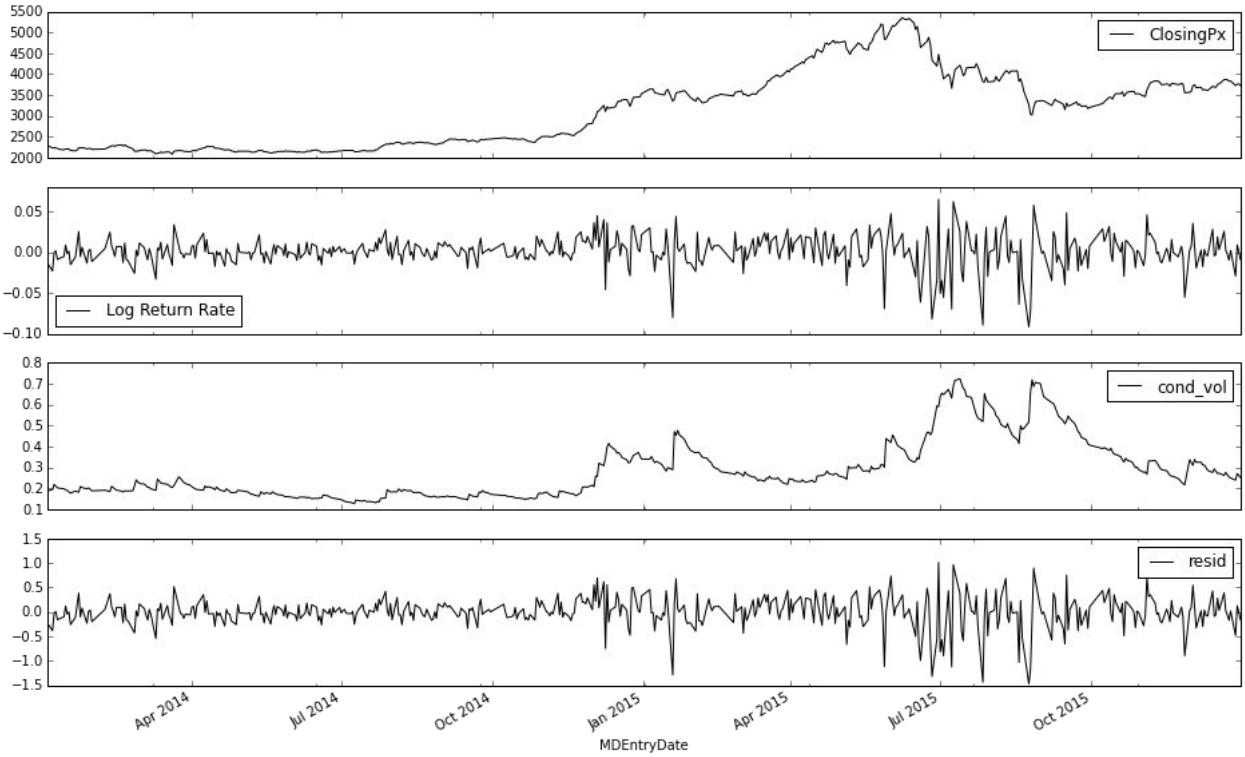
```
from arch import arch_model
from arch.univariate import ZeroMean, GARCH, StudentsT, ConstantMean

arch_mod = ConstantMean(returns)
arch_mod.volatility = GARCH(1,0,1)
arch_mod.distribution = StudentsT()

res = arch_mod.fit(update_freq=5, disp='off')
print (res.summary())
print (' ')
print ('The estimated parameters: ')
print ('----------------------------------------')
print (res.params)
```

```
                      Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:              Log Return Rate   R-squared:                      -0.000
Mean Model:                   Constant Mean   Adj. R-squared:                 -0.000
Vol Model:                            GARCH   Log-Likelihood:                1333.08
Distribution:      Standardized Student's t   AIC:                           -2656.15
Method:                  Maximum Likelihood   BIC:                           -2635.20
                                              No. Observations:                  488
Date:                      Thu, Jun 23 2016   Df Residuals:                      483
Time:                              07:33:56   Df Model:                            5
                                Mean Model
============================================================================
                 coef    std err          t      P>|t|        95.0% Conf. Int.
----------------------------------------------------------------------------
mu         1.1923e-03  3.094e-07   3853.459      0.000   [1.192e-03,1.193e-03]
                            Volatility Model
============================================================================
                 coef    std err          t      P>|t|        95.0% Conf. Int.
----------------------------------------------------------------------------
omega      3.1550e-06  1.444e-15  2.185e+09      0.000   [3.155e-06,3.155e-06]
alpha[1]       0.0874  3.691e-04    236.685      0.000   [8.663e-02,8.807e-02]
beta[1]        0.9088  2.986e-04   3043.502      0.000       [  0.908,  0.909]
                                Distribution
============================================================================
                 coef    std err          t      P>|t|        95.0% Conf. Int.
----------------------------------------------------------------------------
nu             5.3838      0.267     20.168  1.866e-90       [  4.861,  5.907]
============================================================================

Covariance estimator: robust

The estimated parameters:
----------------------------------------
mu          0.001192
omega       0.000003
alpha[1]    0.087350
beta[1]     0.908814
nu          5.383755
Name: params, dtype: float64
```

建模到这里就告一段落了，我们将相应的图形显示出来以方便我们直觉上的认知

In [12]:

```
pd.DataFrame(prices[1:]).join(pd.DataFrame(returns)).join(np.sqrt(252) * pd.DataFrame(res.conditional_volatility).
                                                          join(pd.DataFrame(res.resid))).plot(figsize=(16, 10), subplots=True ,color=
```



在参数估计出来后，我们开始利用所估计出来的参数进行一步提前估计，来预测明天的收益率，从而提前对明天的投资决策做判断

我们在这里写出均值方程和波动率方程的形式，为了更直观地理解接下来的预测过程

$$
\begin{aligned}
r_t &= \mu + \epsilon_t + \theta\epsilon_{t-1} \\
\epsilon_t &= \sigma_t e_t \\
\sigma_t^2 &= \omega + \alpha\epsilon_{t-1}^2 + \beta\sigma_{t-1}^2
\end{aligned}
$$

首先我们需要估计出下一期的波动率，即sigma^2{t+1}, *在此基础上估计出*{t+1}

In [13]:

```
# 参数值
mu = arma_mod01.params[0]
theta = arma_mod01.params[1]

omega = res.params[1]
alpha = res.params[2]
beta = res.params[3]

#print (mu, theta, omega, alpha, beta)
```

In [14]:

```
# sigma_forecastorecast是sigma_t的下一期预测值,sigma_t是当前的波动率

sigma_t = res.conditional_volatility.ix[-1]
sigma_forecast = np.sqrt(omega + alpha * res.resid.ix[-1] ** 2 + beta * res.conditional_volatility.ix[-1] ** 2)

epsilon_t = sigma_t * np.random.standard_normal()
epsilon_forecast = sigma_forecast * np.random.standard_normal()

returns_forecast = mu + epsilon_forecast + theta * epsilon_t
returns_forecast
```

Out[14]:

```
-0.023200367701441398
```

In [15]:

```
# 假设依据现在的参数，预测出未来的10期收益率的值
def returns_predict(period):
    returns_pool = []
    for i in range(period):
        sigma_t = res.conditional_volatility.ix[-1]
        epsilon_t = sigma_t * np.random.standard_normal()
```

```
        sigma_forecast = np.sqrt(omega + alpha * epsilon_t ** 2 + beta * sigma_t ** 2)
        epsilon_forecast = sigma_forecast * np.random.standard_normal()

        returns_forecast = mu + epsilon_forecast + theta * epsilon_t
        returns_pool.append(returns_forecast)
        sigma_t = sigma_forecast

    return returns_pool

returns_predict(5)
```

Out[15]:

```
[-0.0021855133152268677,
 0.01847087322402725,
 -0.028159608570175364,
 -0.023493623876478284,
 0.020373308427950199]
```

当预测的收益率大于0时，则明天开盘买入，否则，开盘卖出 当然我们可以进一步细化交易策略，比如说如果高开大于预测的收益率，则卖出；若开盘跳空低开使得低于预测的收益率，则买入 下面我们进入到逻辑验证的回测阶段

资产组合不同动态调整方法的简单对比--- 基于 CPPI vs. TIPP vs. Constant Mix

社区曾有小伙伴分享过CPPI策略的回测，并加入了一些技术分析判断，链接地址

鉴于前人已经做过相关策略的研究，此处另辟蹊径，想做一个资产组合不同动态调整方法的比较，包括固定组合保险策略（CPPI，Constant Proportion Portfolio Insurance），时间不变性投资组合保险策略（TIPP, Time Invariant Portfolio Protection）以及 恒定混合策略（Constant Mix Strategy），当然加入买入持有策略（Buy and Hold）进行四者比较将会是一个不错的idea

我们先来看看固定组合保险策略的表现情况，此处先进行一个简单的科普：

CPPI概况：
CPPI 策略(Constant Proportion Portfolio Insurance)，又称固定比例组合保险策略，是投资组合保险策略中的一种。CPPI策略通过对投资组合进行动态调整，使之既能提供下行风险的保护，又能最大化潜在收益。CPPI策略将资产分配在低风险资产（如固定收益类资产）和风险资产（如权益类资产）上，通过定量化的资产配置达到本金安全，用投资于低风险资产上的现金净流入来冲抵风险资产组合潜在的可能亏损，并通过投资于风险资产来获得超额收益
CPPI原理：

1. 计算缓冲额度Cushion，C = V - F , V为投资组合的价值，F为最低要保额度，即floor insurance
2. 计算投资于风险性资产(exposure)的部分e : e = m * C , m 为风险参数，一般设置为5左右效果较好
3. 计算投资于无风险资产(bond)的部分b : b = V - e

CPPI特点：

1. 高买低卖，助涨杀跌
2. 连续不断的调整，会使得成本放大，如何兼顾成本与收益也是值得考虑的一点

https://www.ricequant.com/community/topic/1121/

沪深300指数的特征工程和聚类分析-以WorldQuant Formulaic 101 Alphas　为例

作者：江嘉键

The algorithms we used are very standard for Kagglers. […] We spent most of our efforts in feature engineering. [...] We were also very careful to discard features likely to expose us to the risk of over-fitting our model.

—?Xavier Conort, "Q&A with Xavier Conort"

“Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.”

—Andrew Ng, Machine Learning and AI via Brain simulations

研究概述

在机器学习应用中，选择数据的特征维度对于分析结果的优劣至关重要。本文利用米筐科技（RiceQuant）策略研究平台提供的数据，基于《WorldQuant Formulaic 101 Alphas》研究报告中给出的基础变量和变量变换规则，构建了五个新的特征维度对沪深300 指数的收益率进行建模分析。我们提出第 t-1 个交易日的特征和第 t 个交易日的收益率的相关系数，可以作为新特征维度的筛选依据。对K-平均聚类分析的结果进行收益率的均值和历史波动率的估计，证明得到的聚类明显对应不同的收益率分布。依据聚类的结果，我们提出了一个基准择时投资策略，并对其累积收益率曲线进行了分析。研究的详细的代码实现请参看 RiceQuant社区

1 背景介绍

1.1 特征工程 (feature engineering)

在日常的数据分析中，我们会把数据整理成（观测值，特征维度）的两维数据列表的形式。例如，对于一个人数为50的班级，我们可以从 1 到 50 对学生进行编号，并选定姓名，性别，身高，体重，成绩 5 个维度作为分析的特征维度。此时我们的得到的就是一个 50 乘 5 的数据列表（dataframe）。

在一些问题的分析上，对原特征维度进行变换所产生的新特征维度，会增强模型的解释力（explanatory power）。例如，如果我们感兴趣的问题是以上班级学生的健康状况，我们可以通过身高和体重计算学生的的BMI（body mass index）值：

$$BMI = \frac{w}{h^2}$$

其中 w 为体重（千克），h 为身高（米）。通常认为 BMI 在18.5~25之间为合理体重。在这里，BMI 值就是对身高和体重两个维度的组合变换。这种对原特征维度进行变换，来寻找新特征维度的做法，在机器学习中称为特征工程（feature engineering）。知名的国际数据挖掘比赛 Kaggle 的首席数据科学家 Xavier Conort 在采访中谈及他的数据分析经验是提到：“Kaggle上参加数据挖掘的人使用的都是一些标准化的模型，…，我们大部分的时间都花在问题的特征工程上。我们对于舍去特征尤为谨慎，以避免我们的模型出现过拟合的问题。”而斯坦福大学教授，Coursera 上著名的机器学习课程主讲老师 Andrew Ng 的表述则更为言简意赅：“基本上，所谓应用机器学习应用，就是进行特征工程。”

当我们在模型中加入新的特征维度以后，如何判断新模型的解释力？在一些问题中，我们可以对模型的解释力建立量化的判断标准。例如，在网页检索和文本分类问题中，我们可以通过TD-IDF（term frequency-inverse document frequency）值来判断特定关键词对于区分网页或文章主题的能力；在简单线性回归中，依据残差服从正态分布的先验假设，我们通常使用赤池信息准则（Akaike information criterion, AIC）或贝叶斯信息准则（Bayesian information criterion, BIC）来判断自变量对于回归模型的重要性；而如果所需解决的是分类问题，我们可以通过分类结果的准确率（或误差率）的改善程度来判断新加入的特征维度的重要性。另一类通用的方法是，依据数据和问题的特征，我们可以选择合适的数据降维技术，例如主成分分析法（principal component analysis, PCA），线性判别分析（ linear discriminant analysis, LDA）或稀疏表示（sparse representation）来进行数据特征维度筛选。

在这里，基于我们希望能够区分下一个交易日不同的收益率情况的目标，我们提出一个筛选新特征维度的依据：如果第 t-1 个交易日的特征和第 t 个交易日的收益率的相关系数的绝对值越大，则该特征在对于区分不同收益率的重要性越高。在下来的聚类分析中，我们将会验证这个依据的合理性。

2.2 WorldQuant LLC 101 Formulaic Alphas

在量化交易中，一类常见的主动投资策略称为阿尔法策略（alpha trading strategies）。这类策略的目标是获得市场基准组合 (benchmark portfolio) 收益以外的超额收益。在回归分析中，如果我们根据特定的阿尔法策略选定一个投资组合的收益率为R_s，当对市场基准组合收益率R_b 进行回归得到以下的回归方程：

$$R_s = \beta_s \cdot R_b + \theta_s$$

其中系数 β_b 衡量的是 R_s 和市场基准组合的收益率R_s 相关的部分，而 θ_s 则衡量 R_s 中与市场基准组合无关的部分。此时，我们选定的投资组合的阿尔法值（α_s）可定义为 θ_s 的期望：

$$\alpha_s = E[\theta_S]$$

在一篇研究报告中，以数据挖掘能力而闻名业界的对冲基金 WorldQuant LLC 给出了他们的101个阿尔法的数学表达式。这些阿尔法是基于一系列的基本变量、衍生变量和变换方法构建的。以下是一些本文中将会使用的变量和变换方法：

基本变量：*开盘价（*open*）*，*收盘价（*close*）*，*日内最高价（*high*）*，*日内最低价（*low*）*，*交易量（*volume*）*

衍生变量：adv(d)：average daily dollar volume for the past d days.

变换方法：

abs(x): absolute of x.

sum(x, d): time-serial sum over the past d days.

correlation(x, y , d): time-serial correlation of x and y for the past d days.

scale(x, a): rescaled x such that sum(abs(x)) = a (the default is a = 1).

Delta(x, a): today’s value of x minus the values of x d days ago.

signedpower(x, a): x^a

x ? y : z : if x is true, executes y; otherwise executes z.

WorldQuant LLC 给出的的这101个阿尔法的表达式在实盘交易中曾被证明行之有效。因此，我们假设通过这些表达式构建新的特征维度，将有较大的可能反映一定的市场特征。在下面的特征工程和聚类分析中，我们采用报告中给出的五个阿尔法表达式（alpha#6, alpha#23, alpha#28, alpha#54 和 alpha#101）来构建新的特征维度，对2005年1月1日至2015年12月31日这11年间共2671个交易日的沪深300指数进行建模。

应该注意的是，我们采用这些阿尔法表达式生成沪深300指数的特征维度，它们不再具有阿尔法原来的涵义。我们的分析对象始终是沪深300指数，因此不存在根据阿尔法的取值挑选股票投资组合，或决定资产头寸的问题。

2.3 K-平均聚类分析（K-means cluster analysis ）

聚类分析是机器学习中一类无监督学习（unsupervised learning）方法，能够对没有标注的数据进行建模和分析。其核心思想是在特征维度张成的向量空间中，计算观测值之间的距离。距离较近的观测值认为属于同一个聚类（cluster），距离较远的观测值则认为属于不同的聚类。K-平均聚类分析（K-means cluster analysis）是一种常用的聚类分析方法，对于给定的距离度量（例如欧几里得距离，或曼哈顿距离）和聚类个数（K值），K-平均聚类分析通过迭代把观测值划分为不同的聚类。

我们同样可以使用对上述班级学生的例子理解聚类分析。从生活经验来说，和女生相比，男生的身高和体重一般较大。如果以身高和体重作为特征维度，计算观测值之间的距离，把50个学生划分为两个聚类，那么如果其中一个聚类对应较大的升高和体重，其中包含的学生为男生的比例较高；反之为女生的比例较高。

在模型训练中，K-平均聚类分析的迭代算法属于最大期望算法（Expectation Maximization, EM）中的一种。它以聚类内部距离的平方和最小为目标函数（objective function），通过迭代计算目标函数和聚类中心（centroid）来寻找目标函数的极小值。K-平均聚类分析的主要缺点在于，对于非凸的聚类，它无法保证迭代计算收敛到全局最小值，此时计算结果依赖于聚类中心的初始选择。下图为维基百科提供的两个聚类分析示意图：

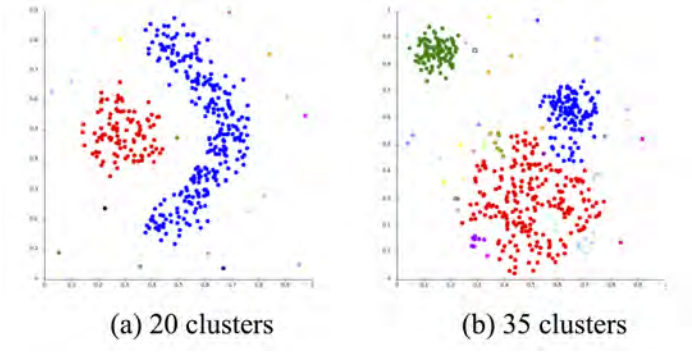


图1：维基百科提供的聚类分析示意图：(a) 20个聚类的聚类分析结果；(b) 35个聚类的聚类分析结果

2 建模流程，结果和分析

根据以上概述部分中提及的变量和变换规则，接下来我们对采用的5个阿尔法表达式所构建的特征维度进行简单讨论，并计算这些特征维度与收益率之间的相关系数。依据构建的特征维度，我们进一步对收益率进行K-平均聚类分析。这里使用的收盘-收盘收益率 R_c (close-to-close return rate)) 为对数收益率：

$$R_c = ln\frac{S_t}{S_{t-1}}$$

S_t 为第t个交易日的收盘价，S_{t-1} 则为第t-1个交易日的收盘价。

2.1 alpha#6

alpha#6 的数学表达式如下：

$$-1 \cdot correlation(open, volume, 10))$$

在《101 Formulaic Alphas》研究报告中，他们认为 alphas 可以分为三类：趋势跟随型（trend-following），均值回归型（mean-reverting）和混合型（trend-following and mean-reverting mixed）。按照这个分类标准，alpha#6 是一个混合型的阿尔法。例如，如果过去十天量价齐升(相关系数为正)，此时alpha为负，属于均值回归型 alpha；如果过去十天量价齐跌（相关系数仍为正），此时 alpha 为负，属于趋势跟随型。

2.2 alpha#23

alpha#23 的数学表达式如下：

$$(((sum(high, 20)/20 < high)?(-1 \cdot delta(high, 2)) : 0)$$

alpha#23的意义可表述如下：如果今天的日内最高价高于过去20天的平均日内最高价，则alpha#23为2日前最高价和今日最高价的差值；否则alpha#23为0。

alpha#23为混合型alpha。今天的日内最高价高于过去20天的平均日内最高价是一种上涨趋势，而alpha#23是趋势跟随型还是均值回归型，则取决于2日前最高价和今日最高价的差值。如果差值为正，则为趋势跟随型，否则为均值回归型。

2.3 alpha#28

alpha#28的数学表达式如下：

$$scale((correlation(adv20, low, 5) + ((high + low)/2) - close))$$

alpha#28是一个趋势跟随型的alpha。由于相关系数的取值范围为[-1, 1]，因此alpha#28的取值主要取决于 $((high + low)/2) - close))$ 部分，当日内最高价和最低价的均值高于收盘价时，alpha#28为正；否则为负。

2.4 alpha#54

alpha#54的数学表达式如下：

$$((-1 \cdot ((low - close) \cdot (open^5)))/((low - high) \cdot (close^5)))$$

alpha#54是一个趋势跟随型的alpha。$open^5$ 和 $close^5$ 恒为正值，而$(low - close)$和$(low - high)$ 恒为负值，因此alpha#54恒为负值。

alpha#54的取值大小则主要决定于$open^5$ 和 $close^5$ ，当开盘价远高于于收盘价时（下跌趋势），alpha#54为一个很大的负数；否则alpha#54为一个较小的负数。

2.5 alpha#101

alpha#101的数学表达式如下：

$$((close - open)/((high - low) + 0.001))$$

alpha#101是一个趋势跟随型的alpha。$(high - low)$ 恒为正值，当收盘价高于开盘价时（上涨趋势），alpha#101为正；否则alpha#101为负。

2.6 相关系数计算

至此，我们共有6个基本特征维度（open, close, high, low, volume, turnover），一个衍生特征维度（adv（20））和5个基于阿尔法的特征维度（alpha#6, alpha#23, alpha#28, alpha#54 alpha#101）。在聚类分析的例子里面，当我们希望两个聚类对应不同的性别时，我们根据生活经验，知道身高和体重两个特征维度和性别有明显的相关性，因此对特征维度进行了人工筛选。但是在预测下一个交易日的收益率的问题上，哪一个特征维度与之存在相关性并不明显。因此，这里我们通过计算第 t-1 交易日的这些特征和第 t 个交易日的收益率的相关系数来实现特征筛选：

$$correlation[features(t-1), R_c(t)]$$

得到表1：

表 1：R_c和特征维度的相关系数计算

open	*close*	*high*	*low*	*volume*	*turnover*
-0.0444	-0.0421	-0.0419	-0.0440	0.0177	0.0005
adv(20)	*alpha#6*	*alpha#23*	*alpha#28*	*alpha#54*	*alpha#101*
-0.0163	-0.0211	-0.0345	-0.0248	-0.0173	0.0557

以相关系数的绝对值大于 0.03 作为标准，可以看出，开盘价，收盘价，最高价，最低价 alpha#23 和 alpha#101 和下一个交易日的 R_c 呈现出较强的相关性。

2.7 K-平均聚类分析

我们对 2647 个交易日（为了保证时间戳对齐，前23个交易日的收益率舍去）进行K-平均聚类分析，我们选择聚类个数为 2（K=2），距离度量为欧几里得距离，经测试 500 次迭代可保证结果收敛。在这里，我们通过聚类对应的R_c 分布来判断聚类分析的效果。具体的标准是：同一个聚类对应的R_c 分布标准差越小越好，而不同聚类对应的 R_c 分布之间的均值差值越大越好。这意味着两个聚类对应明显不同的 R_c 分布。另外需要注意的是，这里 R_c 的均值对应日收益率的均值，而 R_c 的标准差则可作为收益率的历史波动率估计。

首先使用所有的特征维度对下一个交易日的 R_c进行聚类分析，得到表2：

表 2：聚类对应的R_c分布

	数量	对应R_c均值	对应R_c标准差
R_c	2647	0.0005	0.0188
Cluster 1	2013	0.0006	0.0169
Cluster 2	634	0.0002	0.0238

接着根据以上相关系数计算结果，选择和收益率相关系数较大的特征维度（开盘价，收盘价，最高价，最低价，alpha#23 和 alpha#101）进行聚类分析，得到表3：

表 3：筛选特征维度后聚类对应的R_c分布

	交易日数量	对应R_c均值	对应R_c标准差
R_c	2647	0.0005	0.0188
Cluster 1	1614	0.0010	0.0164
Cluster 2	1033	-0.0003	0.0221

对比表2和表3，依据相关系数进行特征维度筛选后，聚类1和聚类2对应的 R_c 之间的均值差值增加，而聚类内的 R_c 标准差减少，说明通过相关系数进行特征维度的筛选能有效提高了不同聚类的区分度。

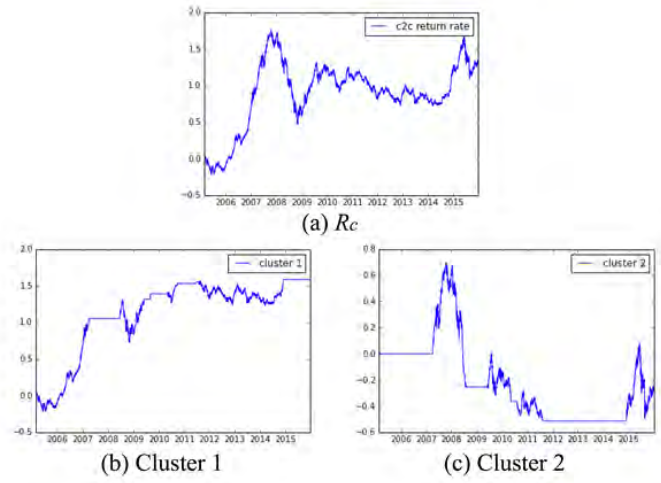
按照表 3 所示，在不考虑交易费用的前提下，聚类1的平均日收益率比这 11 年间的沪深300日交易率均值较高，而且波动率较低。据此，我们可以提出一个基准择时交易策略（benchmark timing strategy）。基准择时交易策略是阿尔法策略以外，另一类常用的主动投资策略。它是指逐个时期选择合适的主动贝塔（即概述中收益率回归方程的 β_b 系数）获得超额收益的投资策略。

在《101 Formulaic Alphas》报告中，阿尔法策略分为“delay-0”和“delay-d”：“delay-0”是在alpha信号生成的交易日内进行交易；而“delay-d”是在alpha信号生成后的 d 个交易日后进行交易。依据这种策略分类方法，和上面的聚类结果，我们可以构建一个“delay-0”的基准择时策略：当我们使用今日的交易日数据（计算中所需的收盘价用足够接近收市时的价格近似）分别计算和聚类 1 和聚类 2 的距离，如果和聚类 1 的距离较近，就在当日收盘时买入；如果和聚类 2 距离较近，则卖出，或不进行交易。

为了进一步观察两个聚类的特征，假设不存在交易费用的前提下，我们按照样本内预测（in-sample forecast）的方式执行上述择时策略并计算其累积收益率曲线（图2）。累积收益率的计算公式如下：

$$R_{t+n} = ln\frac{S_{t+n}}{S_1} = ln\frac{S_2}{S_1} + ln\frac{S_3}{S_2} + \ldots + ln\frac{S_{t+n}}{S_{t+n-1}}$$

$S_1, S_2, \ldots, S_{t+n}$ 为按照指定的策略进行交易的交易日收盘价。按照下图所示的收益率曲线，聚类 1 包含的交易日大致上对应大幅上涨和小幅震荡的行情；而聚类 2 包含的交易日则大致上对应大幅涨跌的行情，这也和上述两个聚类对应的日收益率均值和历史波动率估计相吻合。



(a) Rc (b) Cluster 1 (c) Cluster 2

图2：累积收益率曲线计算：(a)沪深300指数；(b) 在聚类1包含的交易日进行交易；(c) 在聚类2包含的交易日进行交易

3 总结和讨论

在对复杂多变的证券市场进行特征工程时，特征维度的构建有赖于对数据和问题的深刻理解，以及经验，直觉和持之以恒的尝试。在这里我们依据 WorldQuant LLC 提供的基本变量和变化规则，选取其中五个阿尔法表达式，对特征维度的构建和分析进行了简单的尝试。通过相关系数的计算对特征维度进行筛选，和K-均值聚类分析结果的讨论，证明了通过对原特征维度进行变换，并依据第 t-1 个交易日的特征和第 t 个交易日的收益率的相关系数进行特征筛选，能够获得更有解释力的模型。

在K-均值聚类分析中，我们需要自行决定的参数只有距离度量和 K 的个数（迭代次数可通过收敛性判断）。因此，在数据量有限的约束下，相比支持向量机或神经网络等需要进行大量参数估计的复杂模型，K-均值聚类分析能够有效降低过拟合的风险。其次，K-均值聚类分析是一种无监督学习算法，因此特别适用于金融量化分析中没有，或无法对数据进行标注的建模问题。在这里我们简单地选定了欧几里得距离作为距离度量，而K值设置为 2 而对聚类分析的结果进行了初步的探讨，而对这两个参数进行详细的测试和分析，有可能进一步优化聚类分析的结果。

特征选择方法探析—沪深300指数的集成特征选择和聚类分析

作者：江嘉键

泳者，行者，飞者，整个夏季颂扬

诞生，成长，而死去的众生

惑于感官的音乐，全都无视

纪念永生的智慧而立的碑石。

一个老人不过是一件废物，

一件破衣挂在木杖上，

除非灵魂拍掌而歌，愈歌愈激楚，

为了尘衣的每一片破碎；

没有人能教歌，除了去研读

为灵魂的宏伟而竖的石碑；

所以我一直在海上航行，

来到这拜占庭的圣城。

——《驶向拜占庭》节选

叶芝 /诗 余光中 /译

1 特征选择（Feature Selection ）

如果我们要设计一份调查问卷，我们可以用不同问题来了解人群各方面的信息：性别，年龄，爱好，星座，生日，收入水平，职业，政治面貌，是否喜欢叶芝的诗，是否支持川普竞选美国总统，等等。

那么，调查问卷应该包含什么信息？这取决于我们需要回答的问题。例如，如果我们想要了解一个人的政治倾向，根据我们的生活经验（先验假设），“收入水平”，“职业”，“政治面貌”，“是否支持川普竞选美国总统”可能是更为关键的信息。

世事纷繁复杂，针对特定的问题，对描述一个事物所需的信息进行筛选几乎是必不可少的过程。从机器学习算法的角度来叙述，在针对特定问题进行建模的时候，我们需要进行特征选择（feature selection）。在上面提到的例子中，我们认为这群人“是否支持川普竞选美国总统”是重要信息，一般称为相关特征（relevant feature），“是否喜欢叶芝的诗”则可能和他们的政治倾向没有太大的关联，称为不相关特征（irrelevant feature）。然而实际情况可能并非如此，一个人是否喜欢叶芝的诗也许和他的政治倾向存在显著关联。而发现这种有悖生活常识和直觉的关联，恰恰正是数据挖掘和机器学习优越之处。在上一篇研究报告中，我们探讨了如何从已有的数据特征维度出发，进行特征工程（feature engineering）来获得新的特征维度，从而加强模型的解释力。在这里，我们开始探讨通过特征工程获得大量的特征维度以后，如何系统地进行特征筛选。基于 scikit-learn 模块提供的方法，和一篇系统讲述特征选择和代码实现的文章，我们尝试在米筐科技公司策略研究平台中实现并测试了一个集成特征选择打分器（Ensemble Feature Grader, EFG）。具体代码实现请参看米筐科技论坛。

2 特征选择的搜索策略

假定我们通过特征工程获得了 30 个特征变量，当我们进行建模时，每个特征变量有两种可能的状态：“保留”和“被剔除”。那么，这组特征维度的状态集合中的元素个数就是 2^{30} 。更一般地，如果我们有 N 个特征变量，则特征变量的状态集合中的元素个数就是 2^N 。从算法角度讲，特征选择是一个组合优化问题（combinatorial optimization），通过穷举的方式进行求解的时间复杂度是指数级的（$O(2^N)$）。当 N 足够大时，特征筛选将会耗费大量的时间和计算资源（图1）。

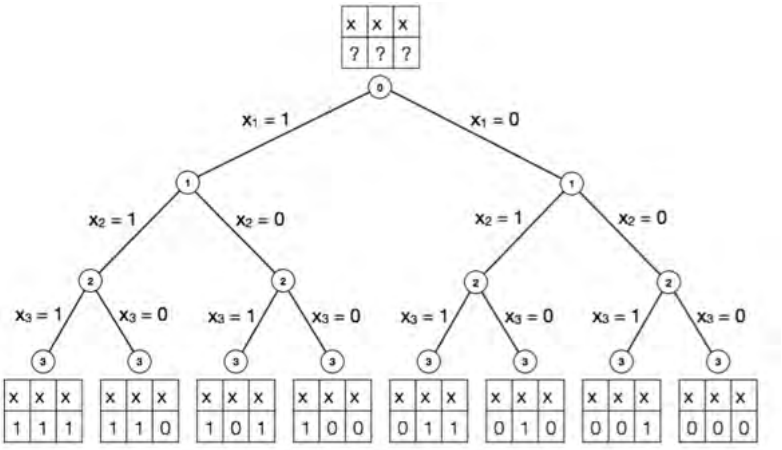


图1：通过穷举法求解特征选择问题的二叉树表示。状态集合中元素的个数随特征变量数目增加而呈现指数增长。

为了减少运算量，目前特征子集的搜索策略大都采用贪心算法（greedy algorithm），其核心思想是在每一步选择中，都采纳当前条件下最好的选择，从而获得组合优化问题的近似最优解。根据其实现的细节，又可将贪心算法分为三种：前向搜索 (forward search)，后向搜索(backward search)和双向搜索 (bidirectional search)。

前向搜索的思想是：假定我们有一个特征集合{a_1, a_2, … , a_{30}}，第一步先从特征变量的集合中选择一个特征变量，构成只有一个元素的特征变量子集。例如依据给定的评价标准，特征a1的效果最优，则子集为{a_1}；第二步，则是往已有的子集中加入下一个效果最优的特征变量，例如对于子集{a_1, a_i}，当i=2 时效果最优，则新的子集确定为{a_1, a_2}。如此重复进行搜索，直到新一轮获得的子集效果不如前一轮，则搜索停止。后向搜索的做法，是以包含全部特征的集合开始，逐步剔除特征，直到找到效果最优的子集。双向搜索则把前向搜索和后向搜索结合起来，不断在选定的子集中加入新特征，并同时剔除旧特征。

这三类算法在 R 语言的 stats 包中的线性回归建模中都实现了。Python 的 scikit-learn 模块中提供了一种循环特征剔除 (recursive feature elimination, RFE) 的实现，遵循的也是后向搜索的思路。值得注意的是，在很多中文资料中，都把RFE翻译为"递归特征消除"，然而根据 scikit-learn 模块官方文档的解释，RFE 的计算并非一个递归过程，因此应属于误译。

3 特征选择的子集的评价

一个完整的特征选择包含子集搜索（subset search）和子集评价（subset evaluation）两个步骤。在下面的讨论中，参照 scikit-Learn 模块中 feature_selection 类官方文档中给出的分类方法，我们把将会用到的子集评价标准分为：单变量特征剔除，依据模型的特征剔除和循环特征剔除。

3.1单变量特征剔除（Univariate Feature Elimination ）

3.1.1 Pearson 相关系数

Pearson 相关系数是最常用的判断特征和响应变量（response variable）之间的线性关联的标准。在上一篇关于沪深300指数的特征工程和聚类分析中，我们通过计算前一个交易日的特征和这个交易日的对数收益率的 Pearson 相关系数实现了特征选择。Pearson 相关系数的优点在于其计算简单，结果易于理解且易于比较；而其缺点在于不能反映变量之间的非线性关系，而且和线性回归一样，可能出现安斯库姆四重奏（Anscombe's quartet）的问题。

3.1.2 距离相关系数（Distance Correlation Coefficient ）

距离相关系数是针对 Pearson 相关系数只能表征线性关系的缺点而提出的。其思想是分别构建特征变量和响应变量的欧氏距离矩阵，并由此计算特征变量和响应变量的距离相关系数（图2）。详细的定义和计算过程可参考维基百科。

距离相关系数能够同时捕捉变量之间的线性和非线性相关（图2）。当距离相关系数为 0，则可断言两个变量相互独立（Pearson 相关系数为 0 不代表变量相互独立）。其缺点是与 Pearson 相关系数相比，其所需的运算量较大，而且取值范围为 [0, 1]，无法表征变量之间关联是正相关还是负相关。

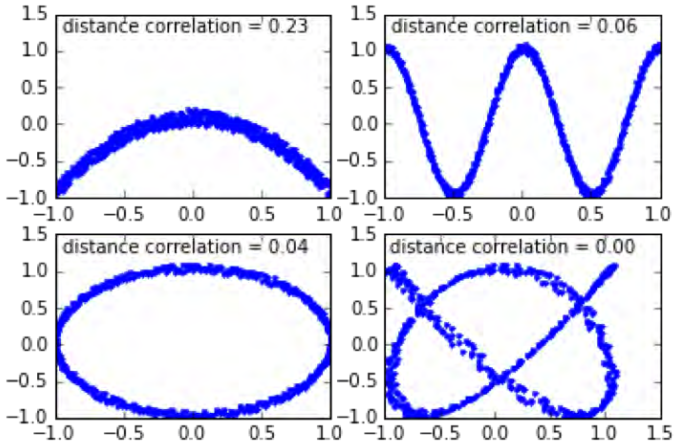


图2：不同的非线性相关下，两个变量的距离相关系数

3.1.3 赤池信息准则（Akaike information criterion, AIC ）和贝叶斯信息准则（Bayesian Information Criterions, BIC ）

AIC和BIC有相近的数学表达式：

$$AIC = 2k - 2ln(L)$$
$$BIC = ln(n) \cdot k - 2ln(L)$$

其中k为参数数目，L是似然函数（likelihood function），n是数据中观测值的数量。

AIC 和 BIC 的表达式中均包含了模型复杂度惩罚项（$2k$和$ln(n) \cdot k$）和和最大似然函数项（$ln(L)$）。不同的地方在于，在 BIC 的表达式中，惩罚项的权重随观测值的增加而增加。因此当观测值数量较大时，只有显著关联的特征变量才会被保留，从而降低模型的复杂性。在 AIC 的维基百科词条中，提到了 AIC 在实践中的效果一般优于 BIC。

在建模时，我们可以通过最小化 AIC 或 BIC 来选择模型的最优参数。由表达式可以看出，AIC 和 BIC 倾向于复杂度低（k越小越好）和符合先验假设（L越大越好）的模型。在简单线性回归中，似然函数L是依据残差服从正态分布的先验假设构建的，即如果特征变量的加入能够使残差更接近正态分布，则认为

这个特征能够显著改善线性回归模型。

3.2 依据模型的特征选择（Feature Selection using SelectFromModel ）

单变量特征剔除的步骤是先进行特征选择，再利用特征进行建模。而依据模型的特征剔除的思路则是把特征选择和模型结合起来。

在周志华老师的《机器学习》书中的第11章 “特征选择和稀疏学习” 中，把先进行特征选择，再进行建模的方法称为过滤式（filter），此时特征选择的标准和模型优化标准并不一定相同，例如上述的Pearson相关系数和距离相关系数的计算，和下一步将要使用的机器学习算法没有必然联系。而把特征选择和模型优化的标准统一起来的方法则有包裹式（wrapper）和嵌入式（embedding）。包裹式的方法以模型的优化标准作为特征选择的标准，但仍然把特征选择和模型训练分为两个步骤，例如上面提到的 AIC 和 BIC；嵌入式则是把特征选择和模型训练融为一体，不再分为两个步骤，例如以下将要介绍的随机森林算法和基于正则化的线性回归模型。

3.2.1 随机森林（Random Forest ）

随机森林算法是把决策树（decision tree）和自助重抽样法（bootstrapping resampling）结合起来的分类或者回归算法。其思想是通过对训练集进行重抽样的方法生成大量的决策树，再把决策树组合起来，从而减少噪音的干扰和过拟合的可能。在 scikit-learn 模块的 ensemble 类中，用随机森林算法进行回归的目标函数是均方差误差函数（Mean Square Error）。

3.2.2 基于正则化的线性回归（Regularization-Based Linear Regression ）

在优化理论（optimization theory）中，正则化（regularization）是一类通过对解施加先验约束把不适定问题（ill-posed problem）转化为适定问题的常用技巧。例如，在线性回归模型中，当用最小二乘法估计线性回归的系数β时，如果自变量存在共线性，系数的估计值$\tilde{\beta}$将具有较大的方差，因而会影响后续参数的统计检验。如果在最小二乘法的参数估计表达式中添加L1正则项 $\lambda|\beta|$ ，则称为Lasso线性回归模型：

$$\tilde{\beta} = arg\min_{\beta}[(Y-\beta\cdot X)^T(Y-\beta\cdot X)+\lambda|\beta|]$$

如果添加L2正则项$\lambda\beta^T\beta$ ，则称为岭回归模型（Ridge Regression）：

$$\tilde{\beta} = arg\min_{\beta}[(Y-\beta\cdot X)^T(Y-\beta\cdot X)+\lambda\beta^T\beta]$$

在线性回归的系数估计中，正则化处理在改善问题的适定性的同时，也会使得系数的估计有偏（biased estimation），因此在选择正则化项的权重λ时，一般的原则是在问题足够适定的前提下，λ应该尽可能小。在接下来的实现中，我们使用上面提到的AIC和BIC来确定正则化项的权重。另外，L2正则化和L1正则化对解施加的是不同的先验约束：L2正则化会令解出现集中分布的特性，而L1正则化则会令解出现稀疏的特性。在Lasso回归中，因为L1正则化会令部分的系数趋近于0，因此也是一种常用的特征选择方法。

尽管L1正则化能够减少系数估计的方差，但利用L1正则化进行特征选择时，其稳定性仍然会受到自变量共线性的影响。例如，在上一篇关于沪深300指数的特征工程的研究报告中，我们提到利用身高，体重两个维度对学生性别进行分类的例子。基于日常生活经验，我们可以预见身高和体重这两个特征变量本身也存在显著的正相关（共线性）。如果先用身高这个维度对学生的性别进行分类，其分类的准确率可能已经达到80%，再利用体重进行下一轮分类，其准确率只提升到85%。在这种情况下，我们可能会得到一个错误的结论：身高这个维度对于区分性别的效果远好于体重这个维度。但实际情况可能并非如此，如果我们改变特征使用的顺序，即先以体重对性别进行分类，得到的分类准确率也可能达到80%甚至更高。

在统计学中，我们通常把这类和自变量和响应变量均存在显著相关性的变量称为混淆变量（confounding variable）。在Lasso回归中，如果存在自变量共线性的问题，则哪一个特征维度被剔除将取决于特征选择子集构建的顺序，从而造成特征选择结果的不确定性。

为了减少 Lasso 回归中特征选择顺序的影响，Python的 scikit-learn 模块 linear_model类 中提供了一个 RandomizedLasso 类。其思路是对训练集的数据进行多次重抽样，从而得到一系列特征选择的子集，再对子集中各个特征变量出现的频率进行统计，剔除掉出现频率低的特征变量。然而在实际测试中，在调用该类的成员函数fit( )时会显示 “This function is deprecated” 的错误信息，其官方文档中亦未对其作出说明。

3.3 循环特征剔除（Recursive Feature Elimination, RFE ）

如上所述，循环特征剔除遵循的是贪婪算法中的后向搜索的策略，在每一步选择最优结果，从而逐步剔除不相关特征。在基于 scikit-learn 模块的实现中，我们需要给出最优结果的评估方法（estimator），在下面的实现中，我们将用简单线性回归作为评估方法，来测试其效果。

3.4 集成特征打分器（Ensemble Feature Grader, EFG）

回顾我们以上的讨论，特征选择的本质上是求解一个计算量随特征变量个数呈指数增长的组合优化问题。基于不同的子集搜索和评价标准，不同的方法给出的都只是一个近似最优解，而解的合理性也将受方法本身的局限性所影响。在真实的数据分析中，如果我们通过特征工程产生大量的特征变量，数据集中包含噪音，共线性，非线性相关等情况的问题将难于避免。因此，为了系统化地进行特征选择，获得更为合理的相关特征变量子集，在这里我们借鉴机器学习里面的集成学习（ensemble learning）的思想，提出一个集成特征打分器（以下称EFG）。其主要思路是，根据以上特征选择的方法对特征进行打分（分数的取值范围为0到1），观察特征变量在不同方法下的得分，进而计算其总得分，以尽量减少数据和单一特征选择方法引起的问题，并改善特征选择的效果。

4 基于 scikit-learn 模块的 EFG 实现和测试

在这里，我们选取八个方法来对特征进行打分：Pearson 相关系数，距离相关系数，简单线性回归，基于 AIC 的 Lasso 回归，基于 BIC 的 Lasso 回归，随机森林（以下称 RF），循环特征剔除（以下称 RFE）和岭回归。其中，scikit-learn 模块并未提供距离相关系数计算。我们通过 Github 上的一份源代码对其进行了实现。

为了保证特征在各个特征选择方法中得分在 0 到 1 之间，我们需要对各个方法给出的特征的得分作归一化处理。对于循环特征剔除，这里我们选用简单线性回归作为它的估计方法，由于其给出的是特征重要性的排序，在这里我们把前五个最重要的特征定为 1 分，而其他特征的得分在 0 到 1 之间均匀分布。

在这里，我们选取 Frideman 训练集对 EFG 进行测试。这个训练集中包含非线性相关项，噪音项，权重不同的线性相关项和引起多重共线性的项，基本上囊括了实际数据处理中可能出现的主要问题，因此适合于 EFG 的测试。测试的结果和分析可参看附录1。

5 基于 EFG 的沪深300指数特征选择

我们因循前一篇沪深300指数的特征工程和聚类分析报告的思路，继续尝试对收盘-收盘对数收益率R_c（close-to-close logarithmic return rate）进行聚类分析。除了上一份研究报告中使用的 11 个特征变量，我们通过特征工程进一步获得 49 个新的特征变量（附录2），因此共有 60 个特征变量。接下来我们对这 60 个特征变量进行特征选择。

在 EFG 中我们使用了数个线性回归或基于线性回归的特征选择方法，由于线性回归要求因变量 Y 近似服从正态分布，我们先通过分位数-分位数图（quantile-quantile plot, QQ Plot）图观察R_c的分布，确认其大致符合正态分布（R=0.9538），且存在一定的集中分布特征（图3）。

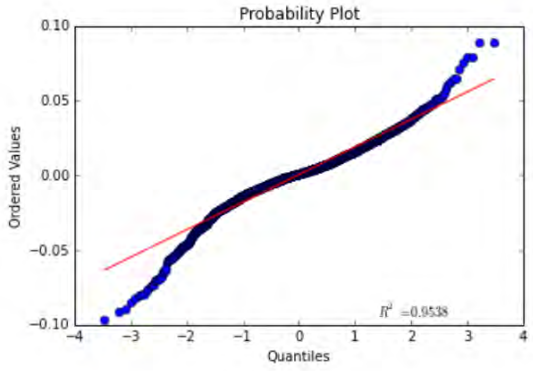


图3：R_c 的分位数-分位数图

在EFG测试中，我们以得分高于0.35为标准，选取了14个特征变量进行聚类分析（附录3）。依照上一篇研究报告的分析方法，我们计算了两个聚类对应的均值和标准差（表1），并和上一篇报告的结果进行了对比（表2）。结果表明两个聚类的均值差值增加，聚类区分度提高，且聚类1的标准差增加的同时聚类2的标准差减少，说明一部分波动率较大的交易日被归为聚类1。

表 1：使用 EFSG 特征选择后聚类对应R_c的统计量

	交易日数量	R_c均值	R_c标准差
R_c	2647	0.0005	0.0188
Cluster 1	1189	0.0012	0.0203
Cluster 2	1458	-0.0004	0.0174

表 2：使用 Pearson 相关系数进行特征选择后聚类对应R_c的统计量

	交易日数量	R_c均值	R_c标准差
R_c	2647	0.0005	0.0188
Cluster 1	1614	0.0010	0.0164
Cluster 2	1033	-0.0003	0.0221

接下来我们用新的聚类结果计算上一篇报告提出的基准择时策略的累计收益率（图4）。和由上一次的聚类结果得到的累计收益率相比（图5），2007年和2015年大幅上涨的交易日由聚类2进入了聚类1，使得聚类1对应的累计收益率上升。综上，通过进一步的特征工程和特征选择，我们有效地提升了聚类分析的建模效果。

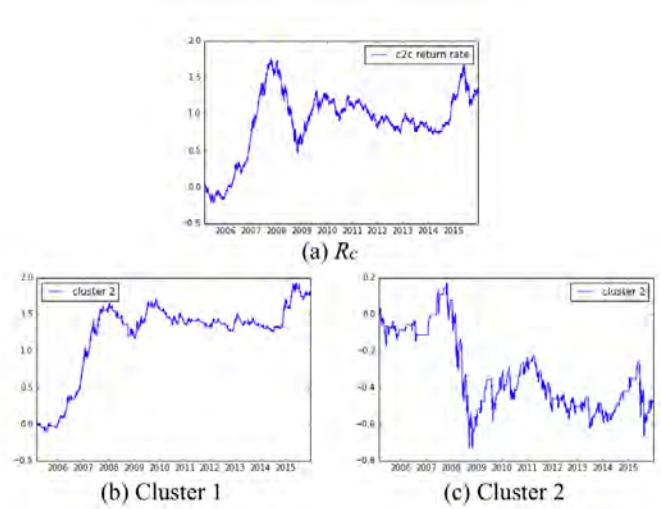


图4：使用EFG进行特征选择后累积收益率曲线计算：(a)沪深300指数；(b) 在聚类1包含的交易日进行交易；(c) 在聚类2包含的交易日进行交易

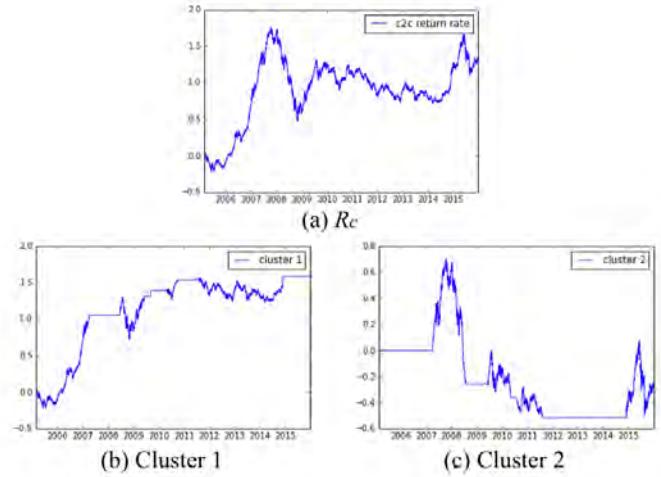


图5：使用 Pearson 相关系数进行特征选择后累积收益率曲线计算：(a)沪深300指数；(b) 在聚类1包含的交易日进行交易；(c) 在聚类2包含的交易日进行交易

6 总结和讨论

特征工程和特征选择是大部分机器学习算法建模前重要的预处理步骤。特征工程和特征选择对于分析结果的影响，往往比之后的机器学习模型的选择更为重要。在这里我们强调了特征选择本质上是一个复杂的组合优化为问题，讨论了各种特征选择方法的特点和局限性，并提出了一个集成特征选择打分器，尝试减少数据的缺陷和方法的局限性给分析结果带来的负面影响。

在这两份报告中我们大致描述了一个实现特征工程和特征选择的框架，可以一定程度上实现这两个步骤的自动化处理。但它仅仅是我们进行数据处理和建模的辅助，而不能代替我们对问题进行分析和思考。首先，如果在进行分析时没有准确，全面地理解数据和方法，我们将有可能错误地理解分析结果。例如，在附录1我们分析 EFG 在 Friedman 训练集测试的结果时，发现简单线性回归方法对非线性相关项给出比线性相关项更高的打分，但这并非是因为简单线性回归能够有效识别非线性相关，而是因为自变量的多重共线性引起了系数估计的不稳定性。其次，如果我们不加思考地通过特征工程获得大量的特征，再分析其和相应变量的关联，将有可能出现多重比较谬误（multiple comparisons fallacy），这种因为大量两两比较而出现的统计学上的伪关系（spurious relationship）将会导致我们对问题作出错误的分析和预测。要减少伪关系的干扰，除了一些通用的统计学处理方法外，一是分析时使用尽量多的样本（回测足够长的时间），同时我们要应当仔细推敲建模所采用的特征变量所对应的的经济学涵义。

7 致谢

本篇研究报告的主要思路和代码框架均来自于：

http://www.tuicool.com/articles/ieUvaq

相关系数矩阵源代码来源于：

https://gist.github.com/josef-pkt/2938402

特征选择方法的分类部分借鉴了周志华老师的《机器学习》的观点，在此一并致谢。

附录1：EFG 在 Friedman 训练集上的测试

我们按以下方式生成 14 组的随机特征变量，每组有750个观测值的 Friedman 训练集：

(1) 生成 10 组服从 [0, 1] 之间均匀分布的随机变量$x_1 \sim x_{10}$：

(2) 按如下的Friedman函数生成响应变量Y:

$$Y = 10 \cdot sin(\pi x_1 x_1) + 20 \cdot (x_3 - 0.5)^2 + 10x_4 + 5x_5 + \varepsilon$$

(3) 再定义变换:

$$F(X) = x + N(0,1)$$

把$x_1 \sim x_4$作为输入，生成和$x_1 \sim x_4$存在显著相关，又包含高斯白噪音的另外四个随机向量。在这14组随机变量中，$x_1 \sim x_3$为非线性项，x_4和x_5为权重不同的线性项，$x_6 \sim x_{10}$为噪音项，$x_{11} \sim x_{14}$为引起多重共线性项。使用EFG对这个训练集进行计算得到表3:

表 3：Friedman 训练的集特征变量在 EFSG 中的得分

	Corr.	Dis. Corr.	LR	Lasso_AIC	Lasso_BIC	RF	RFE	Ridge	Mean
x_1	0.28	0.46	1.0	1.0	0.81	0.63	1.0	0.74	0.74
x_2	0.41	0.59	0.64	0.58	0.84	0.56	1.0	0.74	0.67
x_3	0.0	0.02	0.53	0.43	0.0	0.01	1.0	0.05	0.26
x_4	1.0	1.0	0.64	0.73	1.0	0.92	1.0	1.0	0.91
x_5	0.1	0.15	0.33	0.39	0.56	0.36	0.78	0.89	0.44
x_6	0.0	0.0	0.01	0.01	0.0	0.02	0.22	0.03	0.04
x_7	0.01	0.02	0.0	0.0	0.0	0.02	0.11	0.0	0.02
x_8	0.02	0.03	0.03	0.03	0.03	0.02	0.44	0.08	0.08
x_9	0.01	0.01	0.0	0.0	0.0	0.01	0.0	0.0	0.0
x_{10}	0.0	0.0	0.01	0.01	0.0	0.0	0.33	0.01	0.05
x_{11}	0.27	0.45	0.52	0.44	0.0	0.51	1.0	0.6	0.47
x_{12}	0.41	0.58	0.14	0.0	0.0	0.66	0.67	0.67	0.39
x_{13}	0.0	0.02	0.51	0.41	0.0	0.03	0.89	0.01	0.23
x_{14}	0.99	0.99	0.04	0.06	0.17	1.0	0.56	0.96	0.6

备注：Corr. 表示Pearson 相关系数，Dis. Corr. 表示距离相关系数，LR 表示线性回归，Lasso_AIC 表示基于AIC的Lasso 回归，Lasso_BIC 表示基于BIC的Lasso 回归，RF 表示随机森林，RFE 表示循环特征剔除，Ridge 代表岭回归，最后一列为变量在所有方法中得分的均值。

首先对噪音项$x_6 \sim x_{10}$得分进行分析。在EFG中，这5个随机变量的得分均值明显低于其它特征变量，说明EFG能够有效识别出噪音项。另外，如果观察它们在每一个方法中的打分，其主要得分来源是RFE方法，而且和$x_{12} \sim x_{14}$的得分接近，说明基于简单线性回归的RFE虽然能够有效识别出最重要的几个特征变量，但对于次要的特征变量和噪音项的分辨能力不强。

然后对生成因变量随机变量$x_1 \sim x_5$和$x_{11} \sim x_{14}$进行分析。x_1，x_2，x_{11}和x_{12}在距离相关系数的计算中得分较高，证明了距离相关系数对于某些变量之间非线性相关的识别能力。对于非线性项x_3和x_{13}，它们在相关系数和计算中得分为0，符合相关系数无法表征变量之间非线性相关的特性；而在简单线性回归和基于简单线性回归的RFE中得分较高。然而，这并不说明简单线性回归能够识别非线性相关。其得分较高的原因是因为自变量存在多重共线性（$x_1 \sim x_4$和$x_{11} \sim x_{14}$）。，因而造成参数估计的不稳定。另外两个正则化处理的线性回归方法（基于BIC的Lasso和岭回归）均对x_3和x_{13}给出了较低的分数。

总结来说，在这八个方法里面，线性回归和基于简单线性回归的RFE由于受到多重共线性的干扰，其表现较差；基于AIC的Lasso回归同样表现一般：它既未能对权重最高的线性相关项 x_4 给出最高得分，同时也对等价的特征变量 x_1 和 x_2 也给出了差别较大的打分。而基于BIC的Lasso回归对于特征关联强度较为严苛，它对大部分的特征变量都给出了零分。

最后，各个方法得分均值较为符合我们的期望：相应变量Y中权重最高的线性项（x_4和x_{14}）得分最高（0.84），权重较高的非线性项和权重较低的线性项次之（$x_1 \sim x_3$，x_5和$x_{11} \sim x_{13}$），等价的非线性相关项得分接近（$x_1 \sim x_2$和$x_{11} \sim x_{12}$），噪音项低分最低（$x_5 \sim x_{10}$）。因此，可以说EFG在 Frideman 训练集中的表现令人满意。

附录2：沪深300指数的特征工程

在上一篇报告中，我们使用了以下 12 个特征变量：

（1）6个基本变量：开盘价（open），收盘价（close），日内最高价（high），日内最低价（low），成交量（volume），转手率（turnover）。

（2）1个衍生变量：adv(20)

（3）5个阿尔法变量：alpha#6，alpha#23，alpha#28，alpha#54，alpha#101

除这12个特征变量外，在下面的分析中，我们增加以下的特征变量：

除这12个特征变量外，在下面的分析中，我们增加以下的特征变量：

(1) 我们定义 变量 在前 个交易日的变化率：

$$R_x(d) = ln[\frac{x(t)}{x(t-d)}]$$

备注：对一个随机序列的前后两项的比取对数，来近似其变化率是常用的数学技巧。对上面的对数函数表达式作泰勒展开，可看出其为变化率的一阶近似。

x为以上的6个基本变量。我们计算前1个，5个和10个交易日的变化率，共得到18个特征变量。在下面的分析中，我们将用这些变量的英文首字母作为下标，例如，$R_h(1)$即代表日内最高价（high）在前1个交易日的变化率。

(2) 前一个交易日内最高价和日内最低价比的对数：$$R_{h/l} = ln[\frac{high}{low}]$$

(3) 前一个交易日内收盘价和开盘价比的对数：$$R_{c/o} = ln[\frac{close}{open}]$$

由以上(1)，(2)，(3) 我们得到20个新的特征变量，我们对这20个新特征变量和基本变量中除交易量外的5个变量继续进行特征工程：以交易量为权重，对这些变量的前5个交易日计算其加权平均值，再得到25个新的特征变量。我们对标量添加下标w来表示其为加权平均变量，例如，$R_{h,w}(1)$即 代表日内最高价（high）在前1个交易日的加权平均变化率。

按照《101 Formulaic Alphas》报告中 WorldQuant LLC 给出的阿尔法表达式 alpha #12，alpha #32 和 alpha #41 生成 3 个特征变量。

由此我们共得到 48 个新的特征变量，再加上原来的 12 个特征变量，共得到 60 个特征变量。

附录3：沪深300指数的特征变量在 EFG 中的得分

表4 为沪深300指数的 60 个特征变量的在 EFG 得分。值得注意的是，基于 BIC 的 Lasso 回归对所有的特征变量均给出了 0 分。其原因是是在 2647 个观测值的情况下，BIC 表达式中的惩罚项权重较高，我们选用的特征变量均未被认为能显著改善回归结果。红色部分为得分均值高于 0.35，用于聚类分析的特征变量。

表 4：EFG 下沪深 300 指数的 60 个特征变量的得分

	Corr.	Dis. Corr.	LR	Lasso_AIC	Lasso_BIC	RF	RFE	Ridge	Mean
$R_o(1)$	0.11	0.34	0.07	0.59	0.0	0.48	0.84	0.55	0.37
$R_c(1)$	0.5	0.38	0.04	0.58	0.0	0.47	0.8	0.5	0.41
$R_H(1)$	0.12	0.31	0.02	0.25	0.0	0.47	0.67	0.27	0.26
$R_L(1)$	0.26	0.28	0.06	1.0	0.0	0.48	0.87	0.8	0.47
$R_V(1)$	0.1	0.09	0.01	0.0	0.0	0.22	0.15	0.04	0.08
$R_T(1)$	0.09	0.08	0.01	0.16	0.0	0.28	0.38	0.15	0.14
$R_{c/o}$	0.05	0.29	0.06	0.23	0.0	0.43	0.82	0.34	0.28
$R_{h/l}$	0.02	0.46	0.02	0.21	0.0	0.63	0.45	0.31	0.26
$R_o(5)$	0.37	0.62	0.0	0.13	0.0	0.33	0.09	0.01	0.19
$R_c(5)$	0.74	0.69	0.01	0.08	0.0	0.37	0.29	0.26	0.3
$R_H(5)$	0.92	0.79	0.0	0.0	0.0	0.28	0.07	0.09	0.27
$R_L(5)$	0.5	0.63	0.06	0.69	0.0	0.42	0.78	0.75	0.48
$R_V(5)$	0.77	0.2	0.01	0.0	0.0	0.36	0.22	0.16	0.22
$R_T(5)$	0.93	0.25	0.01	0.01	0.0	0.39	0.24	0.18	0.25
$R_o(10)$	0.15	0.69	0.03	0.0	0.0	0.54	0.55	0.11	0.26
$R_c(10)$	0.3	0.65	0.01	0.27	0.0	0.57	0.2	0.31	0.29
$R_H(10)$	0.43	0.8	0.03	0.0	0.0	0.42	0.42	0.12	0.28
$R_L(10)$	0.1	0.63	0.03	0.28	0.0	0.67	0.73	0.24	0.34
$R_V(10)$	0.52	0.19	0.01	0.14	0.0	0.48	0.69	0.19	0.28
$R_T(10)$	0.68	0.25	0.02	0.33	0.0	0.37	0.71	0.34	0.34
O_w	0.64	0.76	0.58	0.0	0.0	0.06	0.62	0.05	0.34
C_w	0.63	0.75	0.21	0.0	0.0	0.05	0.53	0.05	0.28
H_w	0.62	0.78	1.0	0.0	0.0	0.1	0.96	0.26	0.46
L_w	0.63	0.72	0.19	0.0	0.0	0.05	0.85	0.17	0.33
T_w	0.0	0.47	0.03	0.34	0.0	0.16	0.51	0.46	0.25
$R_{o,w}(1)$	0.06	0.38	0.01	0.32	0.0	0.34	0.27	0.31	0.21
$R_{c,w}(1)$	0.27	0.66	0.01	0.0	0.0	0.3	0.6	0.05	0.24
$R_{H,w}(1)$	0.29	0.58	0.0	0.0	0.0	0.21	0.11	0.07	0.16
$R_{L,w}(1)$	0.35	0.57	0.04	0.36	0.0	0.62	0.47	0.43	0.36
$R_{V,w}(1)$	0.38	0.1	0.0	0.0	0.0	0.26	0.13	0.05	0.12
$R_{T,w}(1)$	0.51	0.13	0.0	0.14	0.0	0.23	0.05	0.06	0.14
$R_{c/o,w}$	0.1	0.47	0.04	0.72	0.0	0.67	0.76	0.46	0.4
$R_{H/L,w}$	0.16	1.0	0.02	0.0	0.0	0.52	0.44	0.12	0.28
$R_{o,w}(5)$	0.09	0.45	0.05	0.0	0.0	0.36	0.56	0.27	0.22
$R_{c,w}(5)$	0.16	0.54	0.01	0.05	0.0	0.34	0.33	0.46	0.24
$R_{H,w}(5)$	0.18	0.55	0.03	0.0	0.0	0.31	0.58	0.06	0.21
$R_{L,w}(5)$	0.14	0.51	0.0	0.36	0.0	0.27	0.02	0.33	0.2
$R_{V,w}(5)$	0.14	0.04	0.01	0.0	0.0	0.32	0.35	0.14	0.12
$R_{T,w}(5)$	0.26	0.06	0.01	0.01	0.0	0.2	0.36	0.15	0.13
$R_{o,w}(10)$	0.53	0.68	0.09	0.0	0.0	0.52	0.98	0.58	0.42
$R_{c,w}(10)$	0.41	0.73	0.08	0.0	0.0	0.21	1.0	0.64	0.38
$R_{H,w}(10)$	0.58	0.74	0.13	0.44	0.0	0.59	1.0	1.0	0.56
$R_{L,w}(10)$	0.53	0.75	0.06	0.0	0.0	0.3	1.0	0.51	0.39

$R_{V,w}(10)$	0.03	0.0	0.02	0.23	0.0	0.29	0.4	0.29	0.16
$R_{T,w}(10)$	0.11	0.05	0.01	0.0	0.0	0.31	0.18	0.12	0.1
α_{12}	0.01	0.49	0.0	0.01	0.0	0.76	0.0	0.01	0.16
α_{32}	0.31	0.74	0.0	0.0	0.0	0.41	0.16	0.0	0.2
α_{41}	0.22	0.99	0.03	0.03	0.0	0.43	0.75	0.13	0.32
O	0.64	0.69	0.91	0.18	0.0	0.03	0.95	0.45	0.48
C	0.57	0.67	0.3	0.0	0.0	0.08	1.0	0.07	0.34
H	0.56	0.71	0.5	0.0	0.0	0.0	0.31	0.0	0.26
L	0.62	0.64	0.3	0.0	0.0	0.02	1.0	0.14	0.34
V	0.1	0.39	0.03	0.46	0.0	0.3	0.89	0.49	0.33
T	0.0	0.44	0.04	0.39	0.0	0.2	0.91	0.58	0.32
adv_{20}	0.09	0.52	0.02	0.4	0.0	1.0	0.49	0.37	0.36
α_6	0.14	0.11	0.0	0.04	0.0	0.51	0.04	0.03	0.11
α_{23}	0.64	0.24	0.01	0.15	0.0	0.67	0.25	0.11	0.26
α_{28}	0.19	0.47	0.01	0.0	0.0	0.55	0.93	0.06	0.28
α_{54}	0.1	0.03	0.03	0.75	0.0	0.46	0.64	0.54	0.32
α_{101}	1.0	0.18	0.04	0.89	0.0	0.53	0.65	0.68	0.5

关于证券收益分布原因的讨论

之前机器学习帖子中有一个图展示了过去十年HS300每日收益率的分布直方图，看起来是个钟样，可是为什么这个样呢？这引起了我的兴趣，我觉得有必要探究探究原因。

实际上，收益的分布问题是个严肃而有意义的问题。合理的分布假设能告知风险管理要求的极端价格变化的概率估计，期权交易中Black-Scholes期权定价模型假定资产价格对数正态分布, Modern Portfolio Theory中假定收益为正态分布。还有很多著名模型均建立在该问题之上。

要弄清证券收益分布背后的原理，我们可以追溯到概率论中的中心极限定理（Central Limit Theorem），关于变量的分布结论许多都是依赖于分布的具体类型。但是对于中心极限定理，它不但对于任意分布都成立，还特别提示我们：要特别注意正态分布。我们下面看看，中心极限定律是如何说的。

中心极限定律

随机变量X1,X2,...,XnX1,X2,...,Xn$X_1, X_2, \ldots, X_n$是相互独立的随机变量，并有相同的分布(IID, Independent & Identically Distributed)。分布的期望为μ，方差为σ2σ2σ^2，μ, σ 都为有限值，且σ ≠ 0。这些随机变量的均值为-XX-$\overline{X}$=1n1n$\frac{1}{n}$\sum_{i=1}^n X_i。让。让。让\zeta_n==\frac{\overline{X}-μ}{σ/\sqrt{n}}$，那么

> limn→+∞P(ζn≤z)=Φ(z)limn→+∞P(ζn≤z)=Φ(z)$lim_{n\to+\infty}P(\zeta_n \leq z) = \Phi(z)$

其中Φ(z)Φ(z)$\Phi(z)$是标准正态分布的分布函数。

简单来说，我们寻找n个IID随机变量的均值-XX-$\overline{X}$。当n趋近无穷时，这个均值(一个新的随机变量)趋近一个正态分布。（实际上这n个IID随机变量的和也服从正态分布）

(通过ζnζnζ_n的变换，可以从正态分布的-XX-$\overline{X}$导出标准正态分布ζnζnζ_n。)

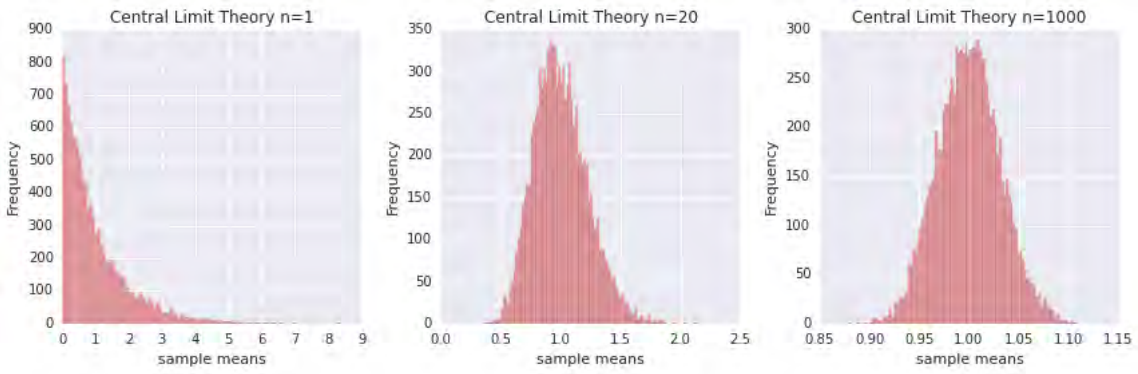
我们进行一个演示：

下面取n个IID随机变量，让它们都符合λ=1的指数分布，并观察它们均值的分布状况。为得到分布，我们使用随机数生成器，来进行10000次采样。每次实验获得一组随机变量的取值，得到一个均值。总共获得10000个均值。绘制均值分布的直方图。

分三种情况，分别让n等于1，20， 100:

```
# Central Limit Theory
# X is exponential distribution with lambda = 1

import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import expon

# Get one sample of (X1 + X2 + ... + XN)/N
def sample_mean(N):
    # exponential distribution, with lambda = 1
    one_sample = expon.rvs(scale = 1, size = N)
    return one_sample.mean()

# Increase N: 1, 20 , 1000.
# Demo of Central Limit Theory in histogram
plt.figure(figsize=(12, 4))
for N, subp in zip([1, 20, 1000], [131, 132, 133]):
    # generate samples
    all_means = np.array([sample_mean(N) for i in range(10000)])
```

```
    # plot figure
    plt.subplot(subp)
    plt.hist(all_means,bins=100,color="red", alpha=0.3)
    plt.title('Central Limit Theory n=%i' % N)
    plt.xlabel('sample means')
    plt.ylabel('Frequency')

plt.tight_layout()
```



在第一种情况下,-XX-$\overline{X}$=X1/1=X1X1/1=X1$X_1/1 = X_1$，即-XX-$\overline{X}$本身是指数分布。

在第二、三种情况下，均值的分布越来越偏离一个指数分布，分布的形状不断趋近于一个正态分布。

如果你有兴趣，可以测试均值分布等其他分布，你会发现结果一致。

收益分布推导

有了中心极限定理，下面可以以此为基础进行推导了。我们定义收益RtRtR_t:

> Rt=ln(Pt/Pt?1)Rt=ln(Pt/Pt?1)$R_t = ln(P_t/P_{t-1})$

这里注意，我们这样定义有别于传统的定义方法，具体原因，这里 有详细的解释。对于某一天的收益而言，它由更小区间时间段的收益累计而成，时间间隔Δ=1/TΔ=1/TΔ = $1/T$的变量ln(Pt+Δ/Pt)ln(Pt+Δ/Pt)$ln(P_{t+\Delta}/P_t)$,这样的小间隔连续复利回报可看做随机变量（无所谓什么分布），对该天收益有：

> ln(Pt+Δ/Pt)+ln(Pt+2Δ/Pt+Δ)+ln(Pt+3Δ/Pt+2Δ)+...+ln(Pt+TΔ/Pt+(T?1)Δ)=ln(Pt+TΔ/Pt)ln(Pt+Δ/Pt)+ln(Pt+2Δ/Pt+Δ)+ln(Pt+3Δ/Pt+2Δ)+...+ln(Pt+TΔ/Pt+(T?1)Δ)=ln(Pt+TΔ/Pt)
> $ln(P_{t+\Delta}/P_t) + ln(P_{t+2\Delta}/P_{t+\Delta}) + ln(P_{t+3\Delta}/P_{t+2\Delta}) + \ldots + ln(P_{t+T\Delta}/P_{t+(T-1)\Delta}) = ln(P_{t+T\Delta}/P_t)$

而

> ln(Pt+TΔ/Pt)=ln(Pt+1/Pt)=Rt+1ln(Pt+TΔ/Pt)=ln(Pt+1/Pt)=Rt+1$ln(P_{t+T\Delta}/P_t) = ln(P_{t+1}/P_t) = R_{t+1}$

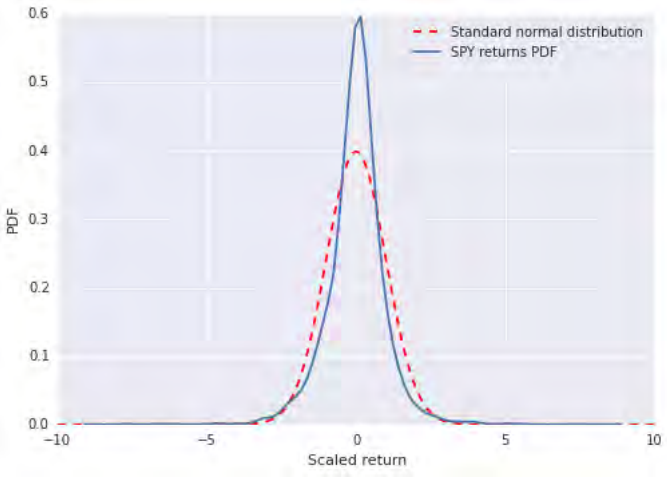
当T足够大时，每日收益可以看做无数小的交易区间的收益之和，这些和（一个新的变量）的分布，根据中心极限定理，从理论上解释为什么大量日收益服从正态分布。

收益分布验证

根据上面的一番推导，理论上可以说得通了，但是实际情况什么样，这需要动手验证，我们以SP500指数为对象，研究自1993年以来，它的每日收益的分布状况并将分布数据标准化后得到概率密度函数曲线与标准正态分布曲线进行比较。

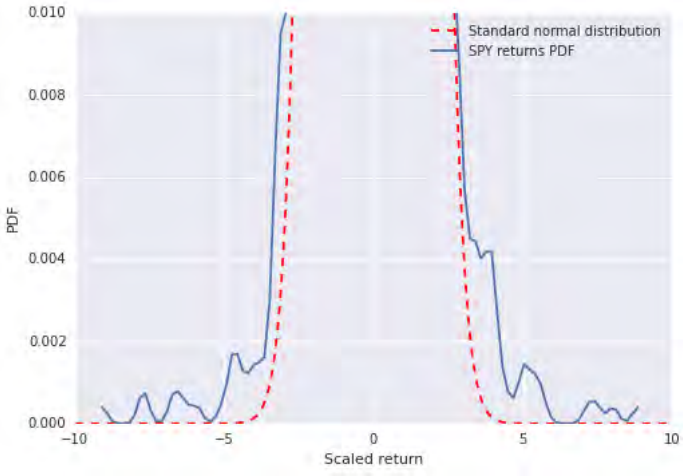
```
import numpy as np
import pandas as pd
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
from scipy.stats.kde import gaussian_kde

mu = 0
sigma = 1
normaldata = np.linspace(-10,10,1000)
plt.plot(normaldata, mlab.normpdf(normaldata, mu, sigma), 'r--')

df_us_long = get_price('SPY.US', country='us', start_date='1993-01-29', end_date='2015-08-10', frequency='daily')
returns_us_ln = np.log(df_us_long['ClosingPx'] / df_us_long['OpeningPx'])
returns_us_ln_normalized = (returns_us_ln - np.mean(returns_us_ln))/np.std(returns_us_ln)

data = returns_us_ln_normalized
kde = gaussian_kde( data )
dist_space = np.linspace( min(data), max(data), 100 )
plt.plot( dist_space, kde(dist_space) )
plt.legend(['Standard normal distribution', 'SPY returns PDF'])
plt.xlabel('Scaled return')
```

```
plt.ylabel('PDF')
plt.show()
```



介于底部看得不是很清楚，我们调整后有图：

```
mu = 0
sigma = 1
normaldata = np.linspace(-10,10,1000)
plt.plot(normaldata, mlab.normpdf(normaldata, mu, sigma), 'r--')
df_us_long = get_price('SPY.US', country='us', start_date='1993-01-29', end_date='2015-08-10', frequency='daily')
returns_us_ln = np.log(df_us_long['ClosingPx'] / df_us_long['OpeningPx'])
returns_us_ln_normalized = (returns_us_ln - np.mean(returns_us_ln))/np.std(returns_us_ln)
data = returns_us_ln_normalized
kde = gaussian_kde( data )
dist_space = np.linspace( min(data), max(data), 100 )
plt.plot( dist_space, kde(dist_space) )
plt.legend(['Standard normal distribution', 'SPY returns PDF'])
plt.xlabel('Scaled return')
plt.ylabel('PDF')
plt.ylim(0, 0.01)
plt.show()
```



我们可以看见与标准正态分布比较有明显的尖顶及肥尾现象。可见对于顶部及尾部而言正态分布的假设并不是那么完美，例如1987年10月19，SP500暴跌20.5%，如果我们以20年为观察年限的话，我们可粗略认为每20年出现一次，而如果我们依据正态分布的假设理论计算，结果会是2×10^{76}2×1076 2×10^{76}年!

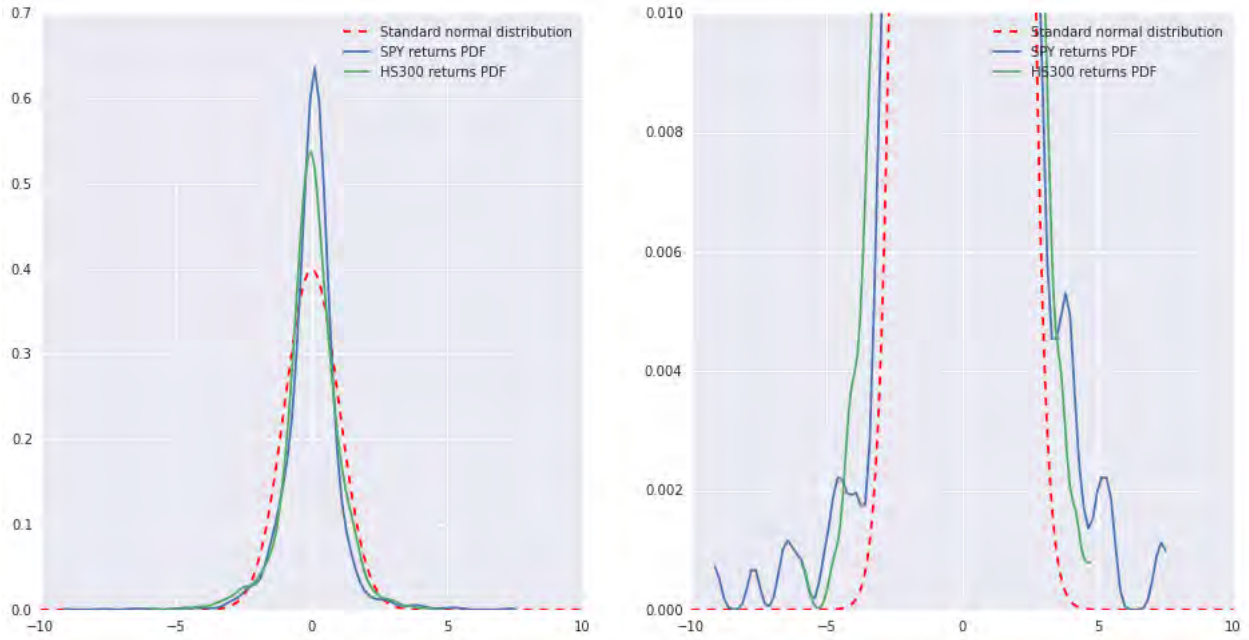
收益分布比较

我们可以将SPY500与HS300收益数据分布进行比较，数据起始时间统一从2005-01-05开始：

```
import numpy as np
import pandas as pd
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
from scipy.stats.kde import gaussian_kde

fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(16, 8))

mu = 0
sigma = 1
normaldata = np.linspace(-10,10,1000)
```

```
ax0.plot(normaldata, mlab.normpdf(normaldata, mu, sigma), 'r--')

df_us_short = get_price('SPY.US', country='us', start_date='2005-01-05', end_date='2015-08-10', frequency='daily')
returns_us_ln = np.log(df_us_short['ClosingPx'] / df_us_short['OpeningPx'])
returns_us_ln_normalized = (returns_us_ln - np.mean(returns_us_ln))/np.std(returns_us_ln)

df_cn_short = get_price('CSI300.INDX', country='cn', start_date='2005-01-05', end_date='2015-08-10', frequency='daily')
returns_cn_ln = np.log(df_cn_short['ClosingPx'] / df_cn_short['OpeningPx'])
returns_cn_ln_normalized = (returns_cn_ln - np.mean(returns_cn_ln))/np.std(returns_cn_ln)

data = returns_us_ln_normalized
kde = gaussian_kde( data )
dist_space = np.linspace( min(data), max(data), 100 )
ax0.plot( dist_space, kde(dist_space) )

data = returns_cn_ln_normalized
kde = gaussian_kde( data )
dist_space = np.linspace( min(data), max(data), 100 )
ax0.plot( dist_space, kde(dist_space) )
ax0.legend(['Standard normal distribution', 'SPY returns PDF', 'HS300 returns PDF'])


ax1.plot(normaldata, mlab.normpdf(normaldata, mu, sigma), 'r--')

data = returns_us_ln_normalized
kde = gaussian_kde( data )
dist_space = np.linspace( min(data), max(data), 100 )
ax1.plot( dist_space, kde(dist_space) )

data = returns_cn_ln_normalized
kde = gaussian_kde( data )
dist_space = np.linspace( min(data), max(data), 100 )
ax1.plot( dist_space, kde(dist_space) )
ax1.set_ylim((0, 0.01))
ax1.legend(['Standard normal distribution', 'SPY returns PDF', 'HS300 returns PDF'])
```

```
<matplotlib.legend.Legend at 0x7f6335553fd0>
```



我们再想想中心极限定理，回过头思考中心极限定理中提到的条件，有如下三个：

1.随机变量必须出自同一分布

2.必须独立

3.有限的均值与方差

而对于实际的金融数据，这三个条件是有回旋余地的，比如，变量可以不是完全独立分布，只要不存在个别变量较其他变量而言会极大程度影响到均值，甚至，变量之间有微弱的关系。事实上，中心极限定律是一个还在积极研究中的领域，而对于收益分布的假设模型后来也有不少学者做了补充与完善，有兴趣可以查阅关于广义的中心极限定理以及稳定分布的论文与资料。

另外一个问题.....

在讨论收益问题查阅资料的时候遇到了关于价格分布的讨论，有学者使用中心极限定理给出证明，但我测试了SP500的表现却差距很大，我想原因可能是假设部分存在漏洞或是理论与现实本就存在出入，这里给出证明的过程希望组里大神可以指点一些。

我们用RiRiR_i代表从第i?1i?1$i-1$天到第ii$i$天的股票收益，我们有：

> S1=S0(1+R1)S1=S0(1+R1)$S_1 = S_0(1+R_1)$

S1S1S_1代表股票的初始价格，那么nn$n$天后，有：

> Sn=S0∏ni=1(1+Ri)Sn=S0∏i=1n(1+Ri)$S_n = S_0 \prod_{i=1}^{n}(1+R_i)$

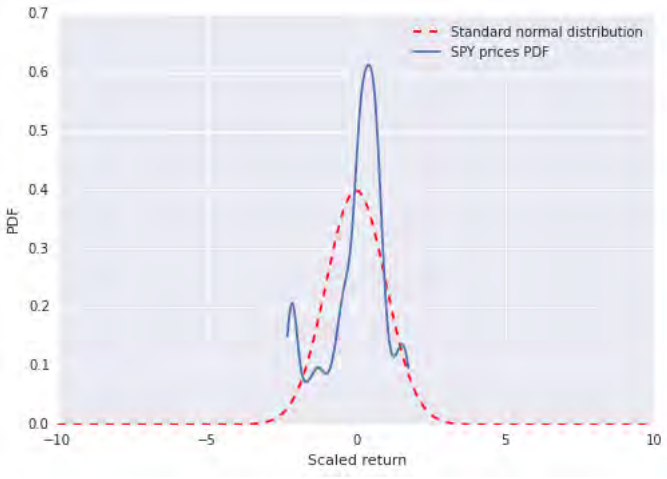
股票价格由由初始值与根据市场不确定性存在的n个因素共同作用得到，每个因素是一个随机的回报，我们等式两边取对数：

> ln(Sn)=ln(S0)+∑ni=1ln(1+Ri)ln(Sn)=ln(S0)+∑i=1nln(1+Ri)$ln(S_n) = ln(S_0) + \sum_{i=1}^{n} ln(1+R_i)$

再对照中心极限定理，如果每个RiRiR_i随机，则ln(1+Ri)ln(1+Ri)$ln(1+R_i)$随机，因而ln(Sn)ln(Sn)$ln(S_n)$为大量随机变量之和，只要RiRiR_i独立同分布且ln(1+Ri)ln(1+Ri)$ln(1+R_i)$的均值方差有限，根据中心极限定理ln(Sn)ln(Sn)$ln(S_n)$为正态分布。

以下是使用SP500的数据验证ln(Sn)ln(Sn)$ln(S_n)$的分布，我想没有其他问题的话，这就是理论与现实的差距吧：

```
mu = 0
sigma = 1
normaldata = np.linspace(-10,10,1000)
plt.plot(normaldata, mlab.normpdf(normaldata, mu, sigma), 'r--')

df_us_long = get_price('SPY.US', country='us', start_date='1993-01-29', end_date='2015-08-10', frequency='daily')['ClosingPx']
df_us_long_ln = np.log(df_us_long)
df_us_long_ln_normalized = (df_us_long_ln - np.mean(df_us_long_ln))/np.std(df_us_long_ln)

data = df_us_long_ln_normalized
kde = gaussian_kde( data )
dist_space = np.linspace( min(data), max(data), 100 )
plt.plot( dist_space, kde(dist_space) )
plt.legend(['Standard normal distribution', 'SPY prices PDF'])
plt.xlabel('Scaled return')
plt.ylabel('PDF')
plt.show()
```



主要参考：

[1]Wilmott, P 2006 Paul Wilmott on Quantitative Finance, second edition

[2]Mandelbrot, B & Hudson, R 2004 The (Mis)Behaviour of Markets: A Fractal View of Risk, Ruin and Reward. Profile Books

[3]Feller, W 1968 An Introduction to Probability Theory and Its Applications, third edition. John Wiley & Sons Inc, New York

十字星形态的简单研究¶

所谓十字星，就是指实体柱很短、上下影线较长的蜡烛图形态。这里认为是多空双方争夺激烈、僵持不下，前期趋势已成强弩之末，趋势反转即将来临。以底部十字星为例，在股票价格经历较长下跌之后，如果出现十字星形态，表明市场空方力量已开始衰竭，多方在逐步蓄积反攻的力量，多空博弈从空方占优演变为势均力敌，股价走势转向的可能性也迅速升高。

超额十字星的计算方法

为了定义超额十字星，我们首先需计算每只股票的超额蜡烛图（或称超额K线）。对于特定股票第T日的超额蜡烛图，其计算步骤如下： （1）以股票第T-1日的收盘价P(T-1,Close)*为基准，计算股票第T日中截止至第t分钟的累积收益率：* Rs(T,t) = P(T,t)/P(T-1,Close)-1； （2）以指数第T-1日的收盘价Q(T-1,Close) *为基准，计算指数第T日中截止至第t分钟的累积收益率：* Ri(T,t) = Q(T,t)/Q(T-1,Close)-1； （3）基于上述两个变量，计算股票第T日中第t分钟的累积超额收益：Re(T,t)= Rs(T,t)-Ri_(T,t)； （4）取第T日累积超额收益的开盘值、最高值、最低值、收盘值，构成第T日的超额蜡烛图。

In [2]:

```
#库导入
import pandas as pd
import numpy as np
import time
import datetime
import seaborn as sns
```

In [3]:

```
#计算一支股票在一定时间内每一天的超额收益十字星
def abnormal_return(stk,index,startdate,enddate):
    tradingdates = get_trading_dates(start_date=startdate, end_date=enddate)
    for date in tradingdates:
        Tday= get_price(stk, start_date=date, end_date=date, frequency='1m', fields='ClosingPx')
        lastday = get_price(stk, start_date=get_previous_trading_date(date), end_date=get_previous_trading_date(date), frequency='1d'
        Rs = Tday/lastday.values-1
        index_Tday= get_price(index, start_date=date,end_date=date, frequency='1m', fields='ClosingPx')
        index_lastday = get_price(index, start_date=get_previous_trading_date(date), end_date=get_previous_trading_date(date), freque
        Ri =index_Tday/index_lastday.values-1
        Re = Rs - Ri

        if date == tradingdates[0]:
            dates = pd.date_range(date,periods=1)
            df = pd.DataFrame(index=dates,columns=['open','close','high','low','bar','upline','downline'])
            df['high'] =  Re.max()
            df['low'] = Re.min()
            df['open'] = Re.ix[0]
            df['close'] = Re.ix[-1]
            df.ix[dates,'bar'] = abs(Re.ix[0]-Re.ix[-1])
            if Re.ix[0]>Re.ix[-1]:
                df.ix[dates,'upline'] = Re.max()-Re.ix[0]
                df.ix[dates,'downline'] = Re.ix[-1] - Re.min()
            else:
                df.ix[dates,'upline'] = Re.max()-Re.ix[-1]
                df.ix[dates,'downline'] = Re.ix[0] - Re.min()

        else:
            dates = pd.date_range(date,periods=1)
            df_1 = pd.DataFrame(index=dates,columns=['open','close','high','low','bar','upline','downline'])
            df_1['high'] =  Re.max()
            df_1['low'] = Re.min()
            df_1['open'] = Re.ix[0]
            df_1['close'] = Re.ix[-1]
            df_1.ix[dates,'bar'] = abs(Re.ix[0]-Re.ix[-1])
            df = pd.concat([df,df_1],axis=0)
            if Re.ix[0]>Re.ix[-1]:
                df.ix[dates,'upline'] = Re.max()-Re.ix[0]
                df.ix[dates,'downline'] = Re.ix[-1] - Re.min()
            else:
                df.ix[dates,'upline'] = Re.max()-Re.ix[-1]
                df.ix[dates,'downline'] = Re.ix[0] - Re.min()

    return df
```

In [4]:

```
#识别个股十字星
def distinguish_star(df):
    #方便计算前后收益
    df['No'] = np.arange(1, len(df)+1)
    #识别符合条件的十字星
    df_eligible = df[df.bar<0.001]
    df_eligible = df_eligible[df_eligible.upline>df_eligible.bar*3]
    df_eligible = df_eligible[df_eligible.downline>df_eligible.bar*3]
    #前20日超额涨幅
    for i in df_eligible.index:
        b=df_eligible.loc[i,'No']
        if b > 20 :
            i_20 = df[df.No == b - 20].index
            a_0 = df.loc[i,'close']
            a_20 = df.loc[i_20,'close'].values
            df_eligible.loc[i,'pre20'] = a_0 - a_20
    #前20日超额涨幅
    for i in df_eligible.index:
        b=df_eligible.loc[i,'No']
        if  b < df.ix[-1,'No']-20:
            i_20 = df[df.No == b + 20].index
            a_0 = df.loc[i,'close']
            a_20 = df.loc[i_20,'close'].values
```

```
            df_eligible.loc[i,'next20'] = a_0 - a_20
    #前后都缺一个数据，砍头去尾
    #df_eligible = df_eligible.ix[1:-1]
    return df_eligible
```

In [6]:

```
stocks = index_components('399905.XSHE')
index =  '399905.XSHE'#中证500
startdate = '2016-9-1'
enddate = '2016-11-25'
df = pd.DataFrame()
for stk in stocks:
    a = abnormal_return(stk, index,startdate,enddate)
    a = distinguish_star(a)
    df = pd.concat([df, a], axis=0)
df
```

Out[6]:

	No	bar	close	downline	high	low	next20	open	pre20	
2016-10-18	27	0.00079774	-0.003770	0.00249208	0.008187	-0.006262	-0.005002	-0.002972	0.001894	0.0
2016-10-21	30	0.000970781	0.001413	0.00504358	0.006563	-0.003631	0.010432	0.002383	-0.010142	0.0
2016-10-27	34	0.000818754	0.004157	0.00332732	0.012912	0.000829	0.013692	0.004975	0.007205	0.0
2016-11-15	47	0.000267991	0.001232	0.00740224	0.002138	-0.006438	NaN	0.000964	0.005002	0.0
2016-09-12	8	0.000636764	0.001410	0.012914	0.005389	-0.011504	-0.000744	0.002047	NaN	0.0
2016-09-21	13	0.000564488	-0.003969	0.00486459	0.000500	-0.008833	-0.030573	-0.003404	NaN	0.0
2016-10-18	27	0.000214608	0.003807	0.00628861	0.007978	-0.002696	0.011139	0.003592	0.007759	0.0
2016-11-01	37	0.000138128	-0.001252	0.00456401	0.005113	-0.005816	NaN	-0.001114	0.015419	0.0
2016-11-02	38	0.000211409	-0.002675	0.0071915	-0.000543	-0.009867	NaN	-0.002464	0.004020	0.0
2016-11-07	41	0.000305295	0.000848	0.00701669	0.005692	-0.006168	NaN	0.001154	-0.001203	0.0
2016-10-14	25	0.000845177	-0.000488	0.00356247	0.009963	-0.004050	0.030127	0.000357	-0.011280	0.0
2016-10-20	29	0.000700113	-0.003276	0.0155403	0.002995	-0.018816	-0.006919	-0.002576	0.016067	0.0
2016-11-24	54	0.000586006	-0.004261	0.00522788	0.001193	-0.009489	NaN	-0.003675	0.018970	0.0
2016-09-08	6	0.000105506	-0.000450	0.00607526	0.007156	-0.006525	0.007919	-0.000344	NaN	0.0
2016-09-21	13	0.000564488	-0.003377	0.00545054	0.005326	-0.008828	0.010758	-0.002813	NaN	0.0
2016-10-14	25	0.000369214	0.004267	0.00896215	0.025011	-0.004696	-0.011524	0.004636	-0.003037	0.0
2016-10-20	29	0.00023063	0.000511	0.0135073	0.013321	-0.012996	-0.067991	0.000742	-0.048191	0.0
2016-10-24	31	0.000727063	-0.005178	0.00838323	0.001363	-0.014288	-0.005292	-0.005905	-0.036011	0.0
2016-10-12	23	3.48944e-05	0.000568	0.00690284	0.003820	-0.006369	-0.010141	0.000533	0.006049	0.0
2016-11-02	38	0.000659388	0.000575	0.00797619	0.003527	-0.008061	NaN	-0.000085	0.004066	0.0
2016-11-11	45	0.000193605	-0.001089	0.00256766	0.003306	-0.003851	NaN	-0.001283	0.007974	0.0

	No	bar	close	downline	high	low	next20	open	pre20	
2016-11-18	50	0.000641578	0.000647	0.00764976	0.006352	-0.007644	NaN	0.000005	0.002954	0.0
2016-11-25	55	0.000100626	-0.004495	0.00260213	0.007634	-0.007097	NaN	-0.004394	0.005382	0.0
2016-09-05	3	0.00020701	-0.003016	0.0043616	-0.000573	-0.007378	-0.008153	-0.002809	NaN	0.0
2016-09-14	10	0.000829046	0.002123	0.00548877	0.008352	-0.003366	0.006859	0.002952	NaN	0.0
2016-10-19	28	0.000776833	0.000612	0.00469935	0.005156	-0.004864	-0.006480	-0.000164	-0.003246	0.0
2016-11-07	41	0.000305295	-0.002535	0.0112942	0.003538	-0.013829	NaN	-0.002229	0.034357	0.0
2016-11-08	42	0.000579561	-0.002164	0.00544842	0.003421	-0.007612	NaN	-0.001584	0.001299	0.0
2016-11-09	43	0.000404826	-0.000740	0.00124617	0.010240	-0.001986	NaN	-0.000335	-0.005877	0.0
2016-11-17	49	0.000716673	-0.002889	0.00742067	0.002564	-0.010310	NaN	-0.002172	0.001174	0.0
...	...	...	...	...	...	...	...	...	...	...
2016-09-29	19	0.000405036	0.005248	0.00857541	0.009771	-0.003732	0.002637	0.004843	NaN	0.0
2016-10-10	21	0.000353476	-0.000871	0.00492891	0.001998	-0.005800	-0.012189	-0.000518	0.005951	0.0
2016-10-11	22	0.000893326	-0.002983	0.00294019	0.005045	-0.006817	0.006228	-0.003877	-0.005929	0.0
2016-11-01	37	0.000513696	0.003787	0.00332305	0.008657	-0.000049	NaN	0.003274	0.007069	0.0
2016-11-04	40	0.000221849	-0.000236	0.00261349	0.002844	-0.003072	NaN	-0.000458	-0.002671	0.0
2016-11-21	51	0.000401302	-0.000547	0.00694253	0.003594	-0.007489	NaN	-0.000146	-0.006729	0.0
2016-09-29	19	4.13928e-05	-0.002787	0.00617661	0.003703	-0.008964	-0.014448	-0.002746	NaN	0.0
2016-10-10	21	0.000586688	-0.002809	0.00825505	0.003330	-0.011651	-0.001846	-0.003396	-0.003616	0.0
2016-10-31	36	4.9729e-05	-0.001628	0.00403167	0.013651	-0.005709	NaN	-0.001678	-0.015054	0.0
2016-11-01	37	0.00053519	0.002917	0.00402879	0.014210	-0.001112	NaN	0.003452	0.005603	0.0
2016-09-02	2	0.000854168	0.000524	0.00902901	0.004768	-0.009359	0.001063	-0.000330	NaN	0.0
2016-09-21	13	0.000257833	0.000035	0.00455663	0.004762	-0.004522	-0.007117	0.000293	NaN	0.0
2016-09-22	14	0.000494773	-0.001697	0.00362974	0.007605	-0.005327	-0.000656	-0.001202	NaN	0.0
2016-09-28	18	0.000565111	0.000581	0.00406779	0.005105	-0.003486	0.003188	0.001147	NaN	0.0
2016-09-29	19	4.13928e-05	-0.000279	0.00292331	0.009352	-0.003203	-0.002042	-0.000238	NaN	0.0
2016-10-10	21	0.000575985	0.002825	0.00304836	0.007299	-0.000800	0.000411	0.002249	0.014439	0.0
2016-10-24	31	0.000760739	-0.003781	0.00282309	0.000378	-0.007365	-0.019246	-0.004542	0.003395	0.0
2016-10-27	34	0.000895743	-0.001041	0.00390836	0.003683	-0.004949	-0.004304	-0.000145	0.000656	0.0
2016-11-03	39	0.000534562	0.001763	0.00312974	0.008293	-0.001367	NaN	0.002297	0.002042	0.0

	No	bar	close	downline	high	low	next20	open	pre20	
2016-11-24	54	0.000213575	0.003263	0.00553358	0.005765	-0.002271	NaN	0.003477	0.004304	0.0
2016-09-12	8	5.17113e-05	-0.007713	0.00675064	0.005134	-0.014464	-0.047022	-0.007661	NaN	0.0
2016-09-19	11	0.00021113	0.002253	0.0101472	0.007446	-0.008105	-0.001974	0.002042	NaN	0.0
2016-09-27	17	0.000428801	-0.005663	0.00498969	0.001271	-0.011081	-0.001716	-0.006092	NaN	0.0
2016-09-12	8	0.000539117	-0.011235	0.00905839	-0.007475	-0.020293	-0.001961	-0.010696	NaN	0.0
2016-09-14	10	0.000328796	-0.001067	0.0096075	0.004199	-0.010674	0.003107	-0.000738	NaN	0.0
2016-10-26	33	0.000189334	0.001176	0.00405326	0.006450	-0.002877	0.010020	0.001365	0.012747	0.0
2016-11-01	37	0.000489511	0.001344	0.0048214	0.006146	-0.003967	NaN	0.000854	-0.013650	0.0
2016-11-10	44	0.000318871	-0.002922	0.00446798	0.001373	-0.007390	NaN	-0.002603	-0.033727	0.0
2016-11-11	45	0.000504047	0.012624	0.00349084	0.038857	0.008629	NaN	0.012120	0.015919	0.0
2016-11-18	50	0.000674624	-0.002607	0.00560334	0.002171	-0.008211	NaN	-0.001933	0.001567	0.0

2003 rows × 10 columns

In [8]:

```
len(df)
```

Out[8]:

```
22
```

In [7]:

```
#把数据存起来
df.to_csv('十字星蜡烛图.csv')
```

In [8]:

```
#读取数据
from six import StringIO
from six import BytesIO
body = get_file('十字星蜡烛图.csv')
data=pd.read_csv(BytesIO(body))
data
```

Out[8]:

	Unnamed: 0	No	bar	close	downline	high	low	next20	open	pre2
0	2016-10-18	27	0.000798	-0.003770	0.002492	0.008187	-0.006262	-0.005002	-0.002972	0.001894
1	2016-10-21	30	0.000971	0.001413	0.005044	0.006563	-0.003631	0.010432	0.002383	-0.01014
2	2016-10-27	34	0.000819	0.004157	0.003327	0.012912	0.000829	0.013692	0.004975	0.007205
3	2016-11-15	47	0.000268	0.001232	0.007402	0.002138	-0.006438	NaN	0.000964	0.005002
4	2016-09-12	8	0.000637	0.001410	0.012914	0.005389	-0.011504	-0.000744	0.002047	NaN
5	2016-09-21	13	0.000564	-0.003969	0.004865	0.000500	-0.008833	-0.030573	-0.003404	NaN
6	2016-10-18	27	0.000215	0.003807	0.006289	0.007978	-0.002696	0.011139	0.003592	0.007759

	Unnamed: 0	No	bar	close	downline	high	low	next20	open	pre2
7	2016-11-01	37	0.000138	-0.001252	0.004564	0.005113	-0.005816	NaN	-0.001114	0.01541
8	2016-11-02	38	0.000211	-0.002675	0.007191	-0.000543	-0.009867	NaN	-0.002464	0.00402
9	2016-11-07	41	0.000305	0.000848	0.007017	0.005692	-0.006168	NaN	0.001154	-0.00120
10	2016-10-14	25	0.000845	-0.000488	0.003562	0.009963	-0.004050	0.030127	0.000357	-0.01128
11	2016-10-20	29	0.000700	-0.003276	0.015540	0.002995	-0.018816	-0.006919	-0.002576	0.01606
12	2016-11-24	54	0.000586	-0.004261	0.005228	0.001193	-0.009489	NaN	-0.003675	0.01897
13	2016-09-08	6	0.000106	-0.000450	0.006075	0.007156	-0.006525	0.007919	-0.000344	NaN
14	2016-09-21	13	0.000564	-0.003377	0.005451	0.005326	-0.008828	0.010758	-0.002813	NaN
15	2016-10-14	25	0.000369	0.004267	0.008962	0.025011	-0.004696	-0.011524	0.004636	-0.00303
16	2016-10-20	29	0.000231	0.000511	0.013507	0.013321	-0.012996	-0.067991	0.000742	-0.04819
17	2016-10-24	31	0.000727	-0.005178	0.008383	0.001363	-0.014288	-0.005292	-0.005905	-0.03601
18	2016-10-12	23	0.000035	0.000568	0.006903	0.003820	-0.006369	-0.010141	0.000533	0.00604
19	2016-11-02	38	0.000659	0.000575	0.007976	0.003527	-0.008061	NaN	-0.000085	0.00406
20	2016-11-11	45	0.000194	-0.001089	0.002568	0.003306	-0.003851	NaN	-0.001283	0.00797
21	2016-11-18	50	0.000642	0.000647	0.007650	0.006352	-0.007644	NaN	0.000005	0.00295
22	2016-11-25	55	0.000101	-0.004495	0.002602	0.007634	-0.007097	NaN	-0.004394	0.00538
23	2016-09-05	3	0.000207	-0.003016	0.004362	-0.000573	-0.007378	-0.008153	-0.002809	NaN
24	2016-09-14	10	0.000829	0.002123	0.005489	0.008352	-0.003366	0.006859	0.002952	NaN
25	2016-10-19	28	0.000777	0.000612	0.004699	0.005156	-0.004864	-0.006480	-0.000164	-0.00324
26	2016-11-07	41	0.000305	-0.002535	0.011294	0.003538	-0.013829	NaN	-0.002229	0.03435
27	2016-11-08	42	0.000580	-0.002164	0.005448	0.003421	-0.007612	NaN	-0.001584	0.00129
28	2016-11-09	43	0.000405	-0.000740	0.001246	0.010240	-0.001986	NaN	-0.000335	-0.00587
29	2016-11-17	49	0.000717	-0.002889	0.007421	0.002564	-0.010310	NaN	-0.002172	0.00117
...	...	...	...	...	...	...	...	...	...	...
1973	2016-09-29	19	0.000405	0.005248	0.008575	0.009771	-0.003732	0.002637	0.004843	NaN
1974	2016-10-10	21	0.000353	-0.000871	0.004929	0.001998	-0.005800	-0.012189	-0.000518	0.00595
1975	2016-10-11	22	0.000893	-0.002983	0.002940	0.005045	-0.006817	0.006228	-0.003877	-0.00592
1976	2016-11-01	37	0.000514	0.003787	0.003323	0.008657	-0.000049	NaN	0.003274	0.00706
1977	2016-11-04	40	0.000222	-0.000236	0.002613	0.002844	-0.003072	NaN	-0.000458	-0.00267

	Unnamed: 0	No	bar	close	downline	high	low	next20	open	pre2
1978	2016-11-21	51	0.000401	-0.000547	0.006943	0.003594	-0.007489	NaN	-0.000146	-0.00672
1979	2016-09-29	19	0.000041	-0.002787	0.006177	0.003703	-0.008964	-0.014448	-0.002746	NaN
1980	2016-10-10	21	0.000587	-0.002809	0.008255	0.003330	-0.011651	-0.001846	-0.003396	-0.00361
1981	2016-10-31	36	0.000050	-0.001628	0.004032	0.013651	-0.005709	NaN	-0.001678	-0.01505
1982	2016-11-01	37	0.000535	0.002917	0.004029	0.014210	-0.001112	NaN	0.003452	0.005603
1983	2016-09-02	2	0.000854	0.000524	0.009029	0.004768	-0.009359	0.001063	-0.000330	NaN
1984	2016-09-21	13	0.000258	0.000035	0.004557	0.004762	-0.004522	-0.007117	0.000293	NaN
1985	2016-09-22	14	0.000495	-0.001697	0.003630	0.007605	-0.005327	-0.000656	-0.001202	NaN
1986	2016-09-28	18	0.000565	0.000581	0.004068	0.005105	-0.003486	0.003188	0.001147	NaN
1987	2016-09-29	19	0.000041	-0.000279	0.002923	0.009352	-0.003203	-0.002042	-0.000238	NaN
1988	2016-10-10	21	0.000576	0.002825	0.003048	0.007299	-0.000800	0.000411	0.002249	0.014439
1989	2016-10-24	31	0.000761	-0.003781	0.002823	0.000378	-0.007365	-0.019246	-0.004542	0.003395
1990	2016-10-27	34	0.000896	-0.001041	0.003908	0.003683	-0.004949	-0.004304	-0.000145	0.000656
1991	2016-11-03	39	0.000535	0.001763	0.003130	0.008293	-0.001367	NaN	0.002297	0.002042
1992	2016-11-24	54	0.000214	0.003263	0.005534	0.005765	-0.002271	NaN	0.003477	0.004304
1993	2016-09-12	8	0.000052	-0.007713	0.006751	0.005134	-0.014464	-0.047022	-0.007661	NaN
1994	2016-09-19	11	0.000211	0.002253	0.010147	0.007446	-0.008105	-0.001974	0.002042	NaN
1995	2016-09-27	17	0.000429	-0.005663	0.004990	0.001271	-0.011081	-0.001716	-0.006092	NaN
1996	2016-09-12	8	0.000539	-0.011235	0.009058	-0.007475	-0.020293	-0.001961	-0.010696	NaN
1997	2016-09-14	10	0.000329	-0.001067	0.009608	0.004199	-0.010674	0.003107	-0.000738	NaN
1998	2016-10-26	33	0.000189	0.001176	0.004053	0.006450	-0.002877	0.010020	0.001365	0.012747
1999	2016-11-01	37	0.000490	0.001344	0.004821	0.006146	-0.003967	NaN	0.000854	-0.01365
2000	2016-11-10	44	0.000319	-0.002922	0.004468	0.001373	-0.007390	NaN	-0.002603	-0.03372
2001	2016-11-11	45	0.000504	0.012624	0.003491	0.038857	0.008629	NaN	0.012120	0.015919
2002	2016-11-18	50	0.000675	-0.002607	0.005603	0.002171	-0.008211	NaN	-0.001933	0.001567

2003 rows × 11 columns

In [11]:

```
data = data[data.pre20 < 0]
data
```

Out[11]:

	Unnamed: 0	No	bar	close	downline	high	low	next20	open	pre2
1	2016-10-21	30	0.000971	0.001413	0.005044	0.006563	-0.003631	0.010432	0.002383	-0.01014
9	2016-11-07	41	0.000305	0.000848	0.007017	0.005692	-0.006168	NaN	0.001154	-0.00120
10	2016-10-14	25	0.000845	-0.000488	0.003562	0.009963	-0.004050	0.030127	0.000357	-0.01128
15	2016-10-14	25	0.000369	0.004267	0.008962	0.025011	-0.004696	-0.011524	0.004636	-0.00303
16	2016-10-20	29	0.000231	0.000511	0.013507	0.013321	-0.012996	-0.067991	0.000742	-0.04819
17	2016-10-24	31	0.000727	-0.005178	0.008383	0.001363	-0.014288	-0.005292	-0.005905	-0.03601
25	2016-10-19	28	0.000777	0.000612	0.004699	0.005156	-0.004864	-0.006480	-0.000164	-0.00324
28	2016-11-09	43	0.000405	-0.000740	0.001246	0.010240	-0.001986	NaN	-0.000335	-0.00587
32	2016-11-16	48	0.000515	0.002750	0.007597	0.011324	-0.004846	NaN	0.003265	-0.00186
33	2016-10-18	27	0.000583	-0.012786	0.002551	0.001859	-0.015338	-0.014888	-0.012203	-0.03554
40	2016-11-16	48	0.000469	-0.002171	0.002159	0.002625	-0.004330	NaN	-0.001702	-0.00595
45	2016-10-31	36	0.000909	-0.001988	0.009070	0.003731	-0.011059	NaN	-0.001080	-0.01483
47	2016-10-17	26	0.000013	-0.003381	0.002299	0.002578	-0.005693	-0.080159	-0.003393	-0.01164
48	2016-11-08	42	0.000626	0.001609	0.006969	0.004473	-0.005986	NaN	0.000983	-0.00281
49	2016-11-07	41	0.000305	-0.000963	0.002803	0.003681	-0.003767	NaN	-0.000658	-0.03737
51	2016-11-02	38	0.000897	0.004901	0.011086	0.012621	-0.007082	NaN	0.004004	-0.04671
52	2016-11-03	39	0.000489	-0.005506	0.014324	-0.001247	-0.020319	NaN	-0.005995	-0.03835
54	2016-11-18	50	0.000249	-0.002399	0.007338	0.004994	-0.009737	NaN	-0.002150	-0.01238
58	2016-10-21	30	0.000465	-0.007672	0.007212	-0.002406	-0.014884	-0.004436	-0.007206	-0.01714
65	2016-11-10	44	0.000187	-0.001195	0.008200	0.004600	-0.009395	NaN	-0.001008	-0.00290
67	2016-11-21	51	0.000617	-0.010909	0.002643	0.000278	-0.013552	NaN	-0.010292	-0.03102
71	2016-11-24	54	0.000018	-0.013320	0.002637	-0.006773	-0.015957	NaN	-0.013302	-0.00820
81	2016-11-03	39	0.000377	-0.002766	0.005016	0.004506	-0.007781	NaN	-0.002389	-0.00443
82	2016-11-14	46	0.000374	-0.002282	0.016615	-0.000118	-0.019271	NaN	-0.002655	-0.01265
83	2016-11-22	52	0.000475	-0.001931	0.006439	0.000791	-0.008846	NaN	-0.002407	-0.02172
92	2016-11-02	38	0.000197	0.000635	0.008563	0.002303	-0.008125	NaN	0.000438	-0.00026
95	2016-11-21	51	0.000668	0.000946	0.003725	0.012448	-0.003446	NaN	0.000279	-0.00176
96	2016-11-22	52	0.000439	0.001828	0.005100	0.009302	-0.003272	NaN	0.002266	-0.00449
97	2016-11-25	55	0.000080	0.000211	0.010224	0.006349	-0.010094	NaN	0.000131	-0.00178

	Unnamed: 0	No	bar	close	downline	high	low	next20	open	pre2
103	2016-10-28	35	0.000133	0.005344	0.013268	0.011861	-0.007923	NaN	0.005477	-0.02964
...	...	...	...	...	...	...	...	...	...	...
1878	2016-10-17	26	0.000861	0.008709	0.007887	0.021834	0.000822	-0.085675	0.009570	-0.00538
1880	2016-11-21	51	0.000807	0.003567	0.012202	0.017646	-0.009442	NaN	0.002760	-0.03731
1882	2016-11-07	41	0.000305	0.001248	0.002353	0.011023	-0.001105	NaN	0.001553	-0.00284
1883	2016-11-14	46	0.000424	0.003397	0.009806	0.008098	-0.006833	NaN	0.002973	-0.00531
1888	2016-11-03	39	0.000666	0.000192	0.004033	0.006258	-0.004508	NaN	-0.000475	-0.00790
1894	2016-11-23	53	0.000488	-0.003042	0.008321	-0.001185	-0.011851	NaN	-0.003530	-0.02028
1896	2016-10-14	25	0.000811	0.002365	0.003132	0.009537	-0.000767	-0.014723	0.003177	-0.01942
1900	2016-10-11	22	0.000039	0.004456	0.013314	0.008419	-0.008857	-0.037308	0.004495	-0.00087
1905	2016-10-13	24	0.000241	-0.002483	0.005701	0.001448	-0.008184	-0.002806	-0.002242	-0.02995
1908	2016-10-13	24	0.000241	-0.000323	0.002504	0.002217	-0.002827	0.012599	-0.000083	-0.01107
1915	2016-10-25	32	0.000507	-0.002262	0.004349	0.002844	-0.006611	-0.003734	-0.001755	-0.02136
1929	2016-11-18	50	0.000070	-0.000559	0.002831	0.005383	-0.003460	NaN	-0.000628	-0.00745
1938	2016-11-02	38	0.000197	-0.003543	0.002251	0.002900	-0.005794	NaN	-0.003346	-0.00071
1941	2016-10-20	29	0.000665	0.003019	0.002862	0.007607	0.000157	0.012531	0.003685	-0.02356
1943	2016-10-12	23	0.000007	-0.003739	0.003059	0.007798	-0.006806	-0.019094	-0.003747	-0.02209
1946	2016-11-15	47	0.000720	-0.003210	0.004632	0.003058	-0.007841	NaN	-0.002489	-0.00277
1948	2016-10-10	21	0.000376	0.002255	0.003476	0.015210	-0.001596	0.004594	0.001880	-0.00734
1949	2016-10-18	27	0.000876	-0.001043	0.005166	0.004163	-0.006209	-0.032124	-0.000167	-0.00542
1950	2016-10-25	32	0.000711	-0.010319	0.006101	-0.003082	-0.017132	-0.005389	-0.011031	-0.01790
1951	2016-10-26	33	0.000370	-0.004374	0.007323	-0.001106	-0.011697	0.001434	-0.004004	-0.01273
1957	2016-10-20	29	0.000231	-0.000459	0.001089	0.003078	-0.001548	-0.001621	-0.000228	-0.00089
1965	2016-11-22	52	0.000138	-0.002269	0.007186	0.003832	-0.009592	NaN	-0.002407	-0.00778
1970	2016-11-04	40	0.000206	-0.001971	0.005055	-0.000329	-0.007026	NaN	-0.001765	-0.00346
1975	2016-10-11	22	0.000893	-0.002983	0.002940	0.005045	-0.006817	0.006228	-0.003877	-0.00592
1977	2016-11-04	40	0.000222	-0.000236	0.002613	0.002844	-0.003072	NaN	-0.000458	-0.00267
1978	2016-11-21	51	0.000401	-0.000547	0.006943	0.003594	-0.007489	NaN	-0.000146	-0.00672
1980	2016-10-10	21	0.000587	-0.002809	0.008255	0.003330	-0.011651	-0.001846	-0.003396	-0.00361

	Unnamed: 0	No	bar	close	downline	high	low	next20	open	pre2
1981	2016-10-31	36	0.000050	-0.001628	0.004032	0.013651	-0.005709	NaN	-0.001678	-0.01505
1999	2016-11-01	37	0.000490	0.001344	0.004821	0.006146	-0.003967	NaN	0.000854	-0.01365
2000	2016-11-10	44	0.000319	-0.002922	0.004468	0.001373	-0.007390	NaN	-0.002603	-0.03372

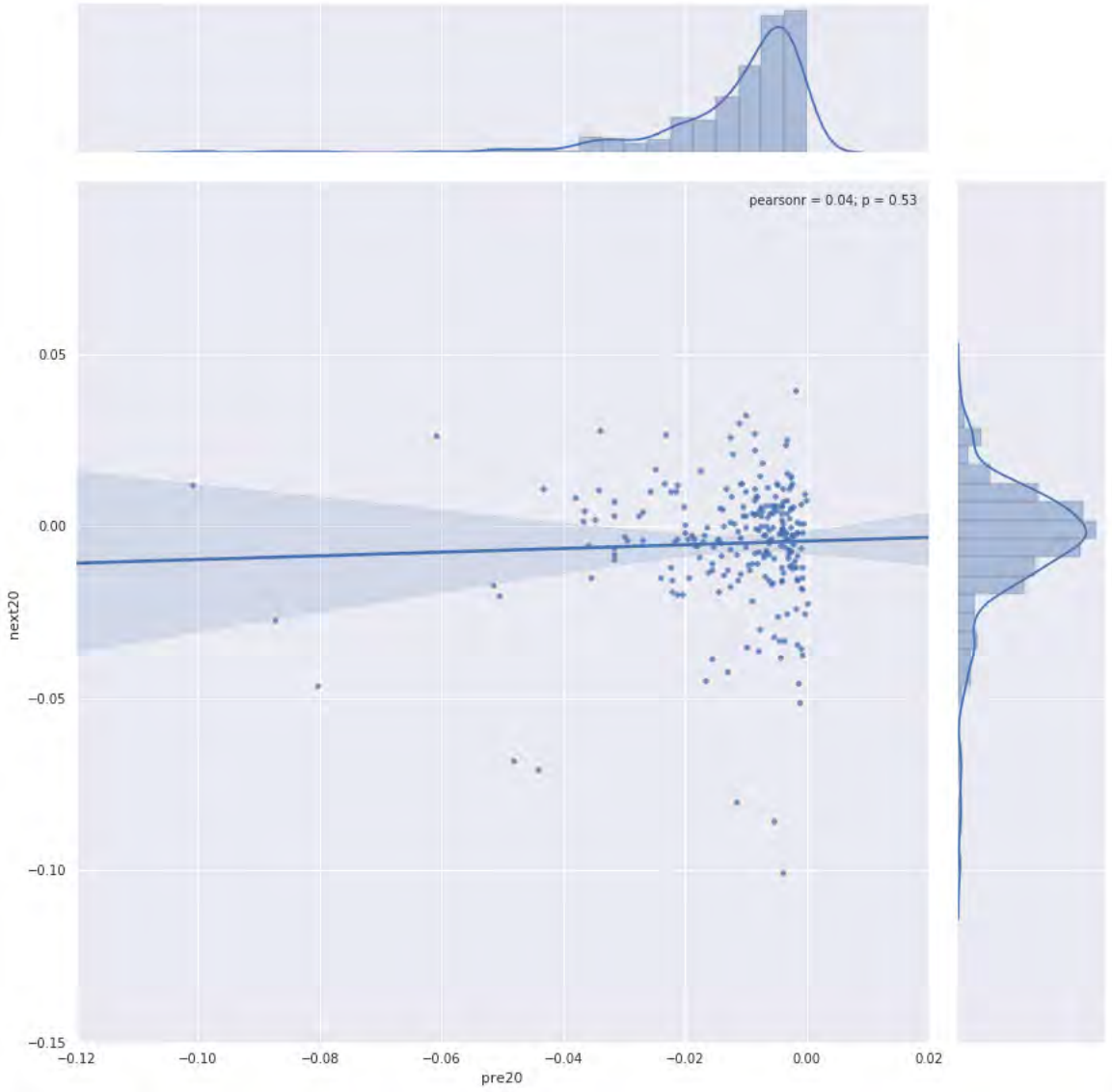
565 rows × 11 columns

In [12]:

```
sns.jointplot(data['pre20'], data['next20'], kind='reg', size=12)
```

Out[12]:

```
<seaborn.axisgrid.JointGrid at 0x7fb1604b9eb8>
```



不太明显，只有两个月的数据样本，但是回归明显向上偏，可见因子还是有效果 下面我选择了中证500成分股中的20支来计算

In [5]:

```
stocks_1 = ['601886.XSHG','601126.XSHG','000099.XSHE','600466.XSHG','600587.XSHG','002353.XSHE','002293.XSHE', '002489.XSHE', '002050
 '002063.XSHE']
index =  '399905.XSHE'#中证500
startdate_1 = '2014-1-1'
enddate_1 = '2016-11-25'
df_1 = pd.DataFrame()
for stk in stocks_1:
    a = abnormal_return(stk, index,startdate_1,enddate_1)
    a = distinguish_star(a)
```

```
    df_1 = pd.concat([df_1, a], axis=0)
df_1
```

Out[5]:

	open	close	high	low	bar	upline	downline	No	pre20
2014-01-07	-0.001897	-0.001943	0.006458	-0.006651	4.55517e-05	0.00835467	0.00470884	4	NaN
2014-02-07	-0.005527	-0.006007	0.013378	-0.009961	0.000480204	0.0189052	0.00395399	22	0.007638
2014-03-05	0.000421	0.001246	0.012940	-0.006317	0.000824954	0.0116936	0.00673872	40	0.015679
2014-03-25	-0.009716	-0.009470	0.013240	-0.010665	0.000246501	0.0227101	0.000949013	54	-0.025961
2014-04-11	0.001609	0.001562	0.007111	-0.001851	4.68696e-05	0.00550171	0.00341342	66	0.006432
2014-04-29	-0.008551	-0.008279	0.006034	-0.012187	0.000271631	0.0143131	0.00363623	78	-0.013229
2014-05-12	0.002395	0.002756	0.017588	-0.004310	0.000360935	0.0148322	0.00670472	85	0.012006
2014-05-14	0.001632	0.002367	0.006712	-0.002074	0.000734458	0.00434597	0.00370598	87	0.016891
2014-05-23	-0.004080	-0.004345	0.000119	-0.008800	0.000265041	0.00419809	0.00445587	94	-0.013341
2014-05-30	0.005732	0.006018	0.016081	-0.000791	0.00028621	0.0100633	0.00652331	99	0.010092
2014-06-03	0.001756	0.001703	0.006032	-0.003826	5.31727e-05	0.00427542	0.00552878	100	-0.018557
2014-06-11	-0.003509	-0.003131	0.000217	-0.008075	0.000377309	0.00334814	0.00456638	106	-0.000009
2014-07-23	-0.002892	-0.002879	0.006577	-0.005467	1.26583e-05	0.00945605	0.00257562	136	-0.009436
2014-11-12	-0.004968	-0.004038	0.002243	-0.008986	0.000930737	0.00628091	0.00401753	210	-0.008778
2014-11-13	0.020005	0.019190	0.025552	0.014111	0.000815201	0.00554659	0.00507876	211	0.028630
2014-11-21	-0.004224	-0.004553	-0.001483	-0.009746	0.000329642	0.00274062	0.00519264	217	-0.003224
2015-01-06	0.000965	-0.000034	0.011175	-0.012660	0.000998943	0.0102104	0.0126261	247	0.019187
2015-01-07	-0.004445	-0.003814	0.009787	-0.012481	0.000631839	0.0136004	0.00803581	248	-0.009749
2015-01-12	-0.001993	-0.001395	0.003410	-0.008329	0.000597669	0.00480551	0.00633593	251	0.011189
2015-01-21	-0.003244	-0.004131	0.002246	-0.008659	0.000887216	0.00548921	0.00452844	258	-0.006634
2015-10-27	-0.030401	-0.030548	-0.003477	-0.061991	0.000146632	0.0269246	0.0314432	442	-0.038208
2015-12-25	-0.003435	-0.004276	0.011034	-0.009270	0.000841036	0.014469	0.00499451	485	-0.050652
2016-02-16	0.008519	0.009004	0.013755	-0.016159	0.000484969	0.00475163	0.0246778	516	0.019557
2016-02-18	-0.000986	-0.000500	0.014788	-0.012698	0.000485495	0.0152881	0.0117118	518	-0.042430
2016-08-04	-0.002302	-0.001616	0.009708	-0.005003	0.000686598	0.0113239	0.00270023	634	-0.016891
2016-08-05	0.000088	0.000812	0.007674	-0.006572	0.000723881	0.00686153	0.00666059	635	0.006899
2016-08-12	-0.002550	-0.003317	0.015110	-0.006441	0.000767386	0.0176602	0.00312398	640	-0.022925

	open	close	high	low	bar	upline	downline	No	pre20
2016-08-18	0.000405	0.000585	0.028629	-0.007071	0.000179878	0.0280438	0.00747644	644	0.014234
2016-08-24	-0.001738	-0.000990	0.002079	-0.005246	0.000748208	0.00306858	0.00350864	648	0.036781
2016-09-19	-0.006974	-0.006850	0.008454	-0.010255	0.00012408	0.0153043	0.00328066	664	-0.007435
...	...	...	...	...	...	...	...	...	...
2014-03-14	0.005339	0.004651	0.011939	-0.001552	0.000688738	0.00659919	0.00620248	47	-0.034051
2014-03-26	-0.001957	-0.002595	0.000969	-0.004665	0.000638664	0.00292573	0.00206933	55	-0.018953
2014-04-02	-0.001095	-0.000655	0.001874	-0.005735	0.000440058	0.00252931	0.0046401	60	-0.002772
2014-06-27	-0.009957	-0.009092	0.000361	-0.014798	0.000865306	0.00945291	0.00484102	118	0.046355
2014-07-30	0.005364	0.006136	0.008639	-0.012577	0.000771817	0.00250293	0.0179418	141	0.035309
2014-08-18	0.001273	0.001144	0.008594	-0.005845	0.00012896	0.00732149	0.00698941	154	0.013711
2014-09-11	0.004093	0.003650	0.014485	0.000533	0.000443532	0.010391	0.00311663	171	0.017122
2014-09-25	-0.004175	-0.003376	0.010172	-0.009249	0.000799809	0.0135475	0.00507392	181	0.045331
2014-11-28	-0.004615	-0.003657	0.011943	-0.016064	0.000958095	0.0156003	0.0114492	222	0.009065
2015-01-27	-0.006059	-0.005392	0.004997	-0.023593	0.000667311	0.0103882	0.017534	262	0.010074
2015-04-16	-0.021033	-0.021159	0.023717	-0.039796	0.000125134	0.0447502	0.018637	313	-0.048245
2015-05-12	-0.011428	-0.012081	0.046723	-0.014717	0.000653123	0.0581516	0.00263569	330	-0.001851
2015-07-23	-0.012344	-0.011370	0.000708	-0.025381	0.000973568	0.0120778	0.0130374	381	0.033617
2015-08-13	0.011248	0.011441	0.020079	-0.001688	0.000193306	0.00863767	0.0129358	396	-0.040084
2015-09-02	-0.013187	-0.012656	0.027920	-0.036755	0.0005314	0.0405758	0.0235673	410	0.003805
2015-10-13	-0.012480	-0.012406	0.004185	-0.018233	7.38099e-05	0.016591	0.00575265	432	-0.067521
2015-12-17	0.006922	0.007816	0.015884	-0.001925	0.000894005	0.00806864	0.00884642	479	-0.010617
2016-02-04	0.003553	0.004552	0.014893	-0.002413	0.000998968	0.0103405	0.00596647	513	0.018235
2016-03-03	-0.001259	-0.001851	0.020796	-0.006639	0.00059226	0.0220553	0.0047874	528	0.003676
2016-03-25	0.003776	0.003463	0.006021	-0.007714	0.000312261	0.00224559	0.0111773	544	0.046411
2016-04-14	0.004278	0.003922	0.012181	-0.003184	0.000355497	0.00790288	0.00710585	557	0.005424
2016-04-15	-0.004720	-0.004970	-0.002328	-0.011955	0.000249798	0.00239256	0.00698465	558	-0.025893
2016-04-26	-0.004821	-0.004057	-0.000586	-0.012377	0.000763958	0.00347106	0.00755643	565	0.005957
2016-06-01	-0.006262	-0.005929	0.003318	-0.013084	0.00033289	0.0092476	0.00682147	590	-0.001883
2016-07-11	-0.001556	-0.001341	0.013781	-0.003138	0.000214684	0.0151221	0.00158217	616	-0.021395

	open	close	high	low	bar	upline	downline	No	pre20	
2016-08-18	0.000405	-0.000315	0.003329	-0.005805	0.000720212	0.00292416	0.00548998	644	-0.012813	
2016-08-30	-0.002213	-0.002587	0.006481	-0.004237	0.000373993	0.00869423	0.00165023	652	0.000442	
2016-09-28	-0.002059	-0.002126	0.001013	-0.004687	6.68724e-05	0.00307291	0.00256019	671	-0.004542	
2016-10-12	-0.000907	-0.001598	0.004475	-0.004918	0.000690257	0.00538284	0.00332033	676	0.002437	
2016-10-13	0.000641	0.000400	0.005512	-0.003841	0.000240639	0.00487045	0.00424137	677	-0.014095	

602 rows × 10 columns

In [8]:

```
df_1 = df_1[df_1.pre20<0]
df_1.head()
```

Out[8]:

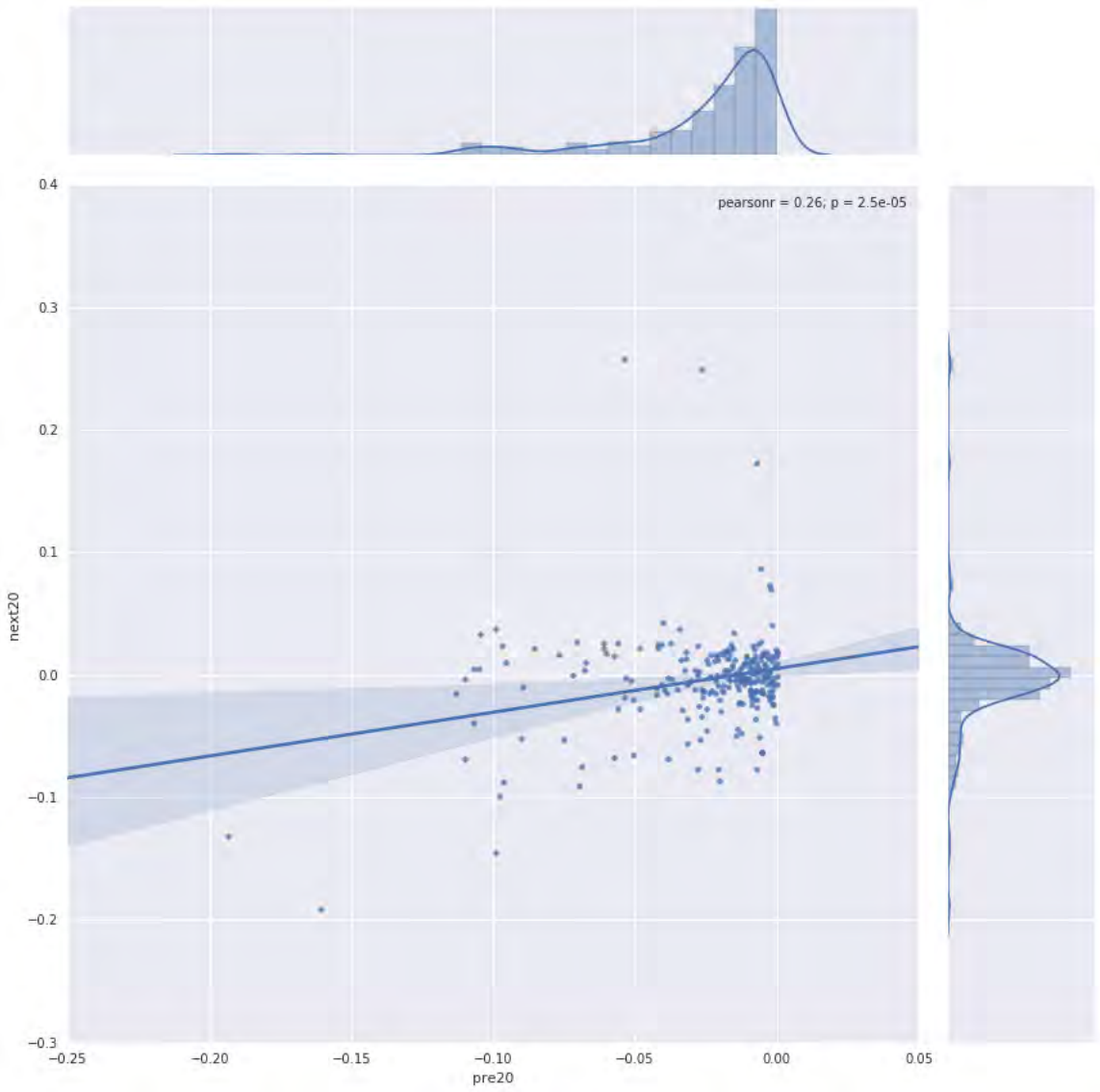
	open	close	high	low	bar	upline	downline	No	pre20	
2014-03-25	-0.009716	-0.009470	0.013240	-0.010665	0.000246501	0.0227101	0.000949013	54	-0.025961	
2014-04-29	-0.008551	-0.008279	0.006034	-0.012187	0.000271631	0.0143131	0.00363623	78	-0.013229	
2014-05-23	-0.004080	-0.004345	0.000119	-0.008800	0.000265041	0.00419809	0.00445587	94	-0.013341	
2014-06-03	0.001756	0.001703	0.006032	-0.003826	5.31727e-05	0.00427542	0.00552878	100	-0.018557	
2014-06-11	-0.003509	-0.003131	0.000217	-0.008075	0.000377309	0.00334814	0.00456638	106	-0.000009	

In [7]:

```
sns.jointplot(df_1['pre20'], df_1['next20'], kind='reg', size=12)
```

Out[7]:

```
<seaborn.axisgrid.JointGrid at 0x7fbf5d5265c0>
```



虽然样本还是少了一些，但是应该已经足够说明问题了 这个因子是比较有效的

统计套利（一）利用相关系数进行配对交易¶

根据统计数据，对价差进行买卖，而不去做股票本身趋势的预测，是否能做到旱涝保收呢。下面是利用股票对之间的相关系数来进行配对交易的研究。 1，首先想到利用统计套利，可能会想到两只股票的相关系数是否会让两只股票的走势有一种特定关系。

In [2]:

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

首先相关系数的公式定义为下，将协方差标准化即得到我们需要的相关系数

$$\frac{Cov(X,Y)}{std(X)std(Y)}$$

$$= \frac{Cov(X,Y)}{\sqrt{var(X)}\sqrt{var(Y)}}$$

$$= \frac{Cov(X,Y)}{\sqrt{Cov(X,X)}\sqrt{Cov(Y,Y)}}$$

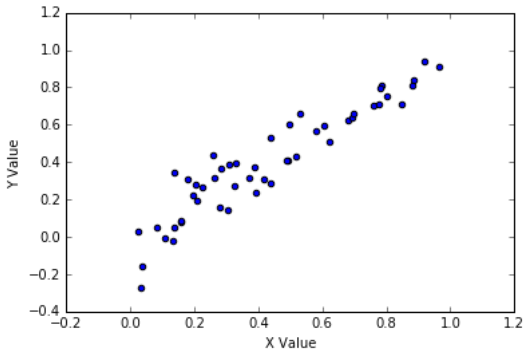
我们来看看相关系数高的数据集具体长什么样

In [3]:

```
X = np.random.rand(50)
Y = X + np.random.normal(0, 0.1, 50)

plt.scatter(X,Y)
plt.xlabel('X Value')
plt.ylabel('Y Value')
```

```
print ('相关系数：' + str(np.corrcoef(X, Y)[0, 1]))
```

```
相关系数：0.938486943102
```

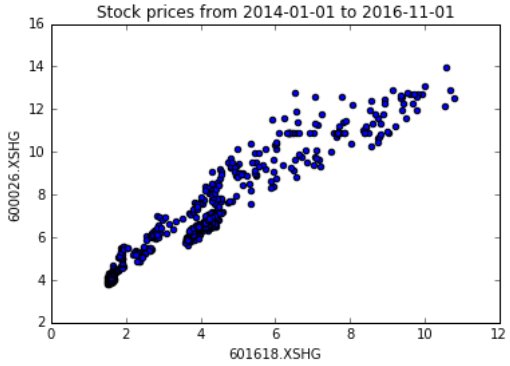


从图像上看，数据基本都落在一条直线上那么它们之间的相关性就会很高。 接下来我们来看看两只股票价格之间相关性高长什么样

In [4]:

```
start = '2014-01-01'#此处时间一定要与回测的时间相对应，因为不同时间可能相关性不一致
end = '2016-11-01'
stock1='601618.XSHG'
stock2='600026.XSHG'
a1 = get_price(stock1,fields='ClosingPx', start_date=start, end_date=end)
a2 = get_price(stock2,fields='ClosingPx', start_date=start, end_date=end)
#上图
plt.scatter(a1,a2)
plt.xlabel(stock1)
plt.ylabel(stock2)
plt.title('Stock prices from ' + start + ' to ' + end)
print (stock1+"与"+stock2+"之间的相关系数: ", np.corrcoef(a1,a2)[0,1])
```

```
601618.XSHG与600026.XSHG之间的相关系数:  0.968302764073
```



同样数据大多都集中在一条直线上

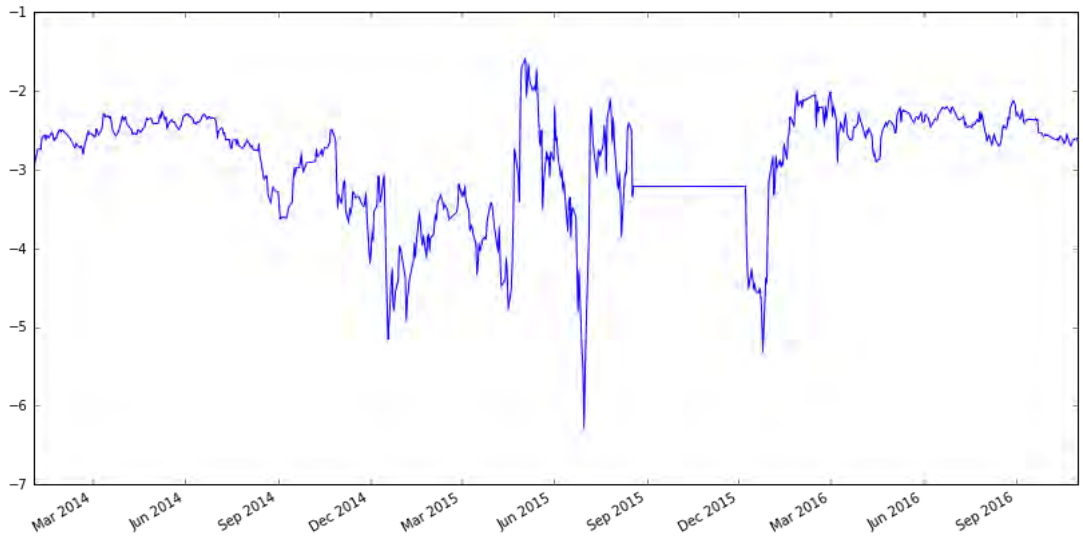
找到相关性高的股票对，我们要来研究它们之间的价差，因为这是我们策略套利的关键

In [5]:

```
a3=a1-a2
a3.plot(figsize=(14,7))
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f846c6f2588>
```



从图中看出，所以相关系数高，两者之间的价差不一定会围绕一个常数波动，价差会具有一定的变异性，即价差序列是非平稳的。 我们来检验下价差的平稳性。

In [6]:

```
from statsmodels.tsa.stattools import adfuller
adftest = adfuller(a3)#使用adf单位根检验平稳性
result = pd.Series(adftest[0:4], index=['Test Statistic','p-value','Lags Used','Number of Observations Used'])
for key,value in adftest[4].items():
        result['Critical Value (%s)'%key] = value
print(result)
```

```
Test Statistic                  -3.603560
p-value                          0.005693
Lags Used                        8.000000
Number of Observations Used    681.000000
Critical Value (5%)             -2.865793
Critical Value (1%)             -3.439989
Critical Value (10%)            -2.569035
dtype: float64
```

上面使用了adf单位根检验，具体的我们会在下一部分协整套利部分来具体讲解，此处检验出来价差非平稳，套利策略的数据基础不好。

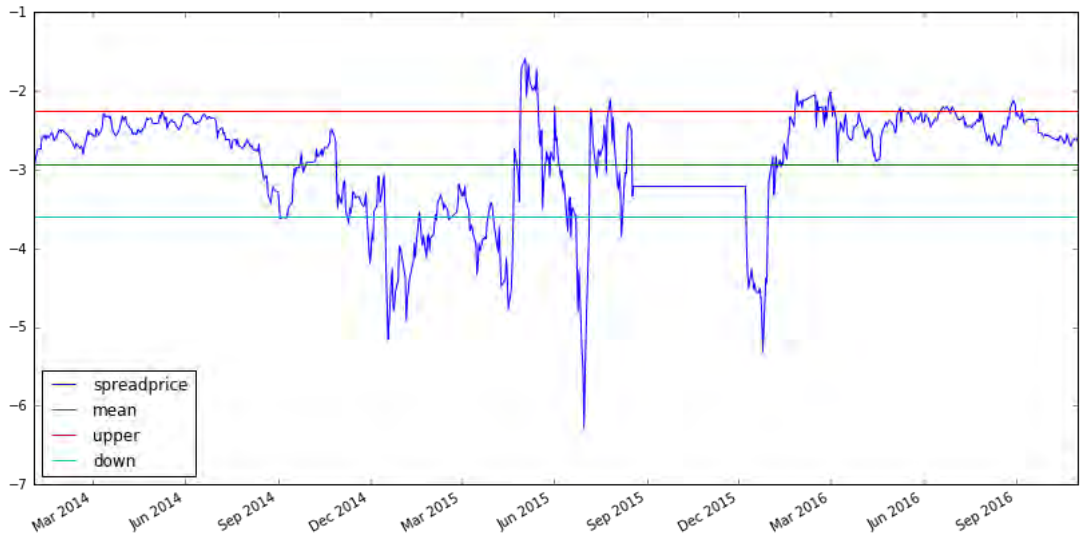
进一步的我们来看看以均值加减一倍标准差是否包含了大部分的差价区间

In [7]:

```
mean=np.mean(a3)
std=np.std(a3)
up=mean+std
down=mean-std
time=a3.index
mean_line=pd.Series(mean,index=time)
up_line=pd.Series(up,index=time)
down_line=pd.Series(down,index=time)
set=pd.concat([a3,mean_line,up_line,down_line],axis=1)
set.columns=['spreadprice','mean','upper','down']
set.plot(figsize=(14,7))
```

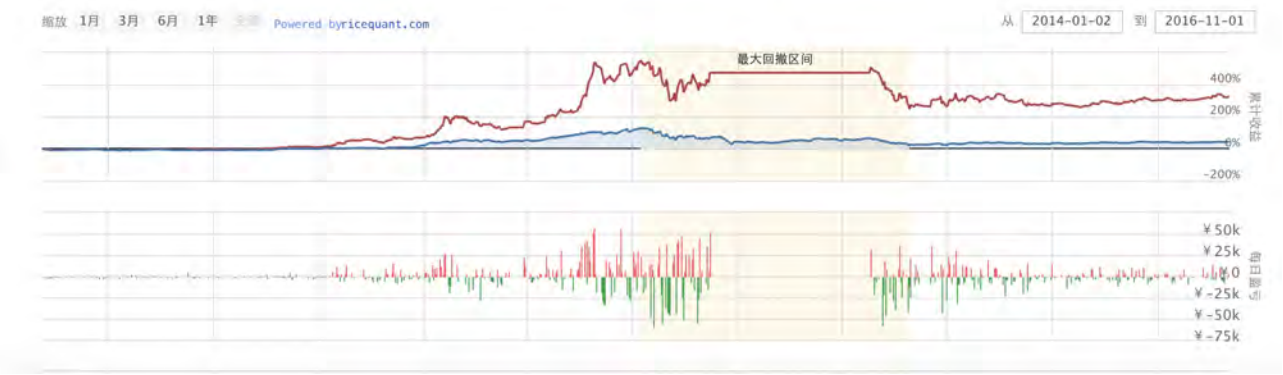
Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f845517ad30>
```



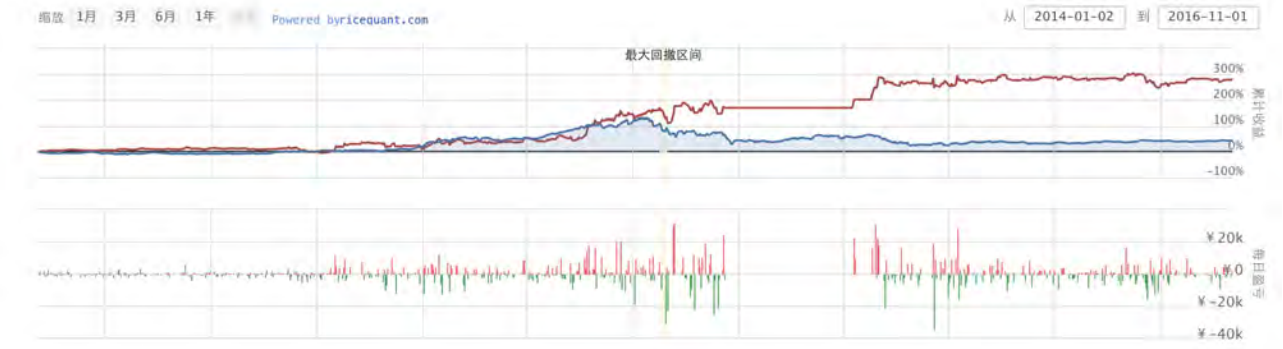
可以看到虽然包含了大部分价差区间，但是开仓次数太少，并且在2014年股票的差价都是在上开仓线附近小幅波动，会造成频繁开仓使得成本十分高。 同时观察2015年价差出现极端值，此时如果开仓，价差不收敛，如果没做到好的平仓条件此时会造成大量亏损。 尽管看图像上通过相关系数来做配对交易不太理想，我们还是通过回测来看看具体结果。

回测收益	回测年化收益	基准收益	基准年化收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Volatility	最大回撤	Tracking Error	Downside Risk
324.649%	66.525%	44.514%	13.866%	0.5229	1.0383	1.2435	2.1081	1.3183	0.5	-46.379% (2015-06-08, 2016-01-28)	0.3994	0.2949



但是因为中国市场上无法做空，但配对交易需要运用卖空来构建对冲仓位，我们试试在允许做空的情况下策略表现会不会相对较好一点

回测收益	回测年化收益	基准收益	基准年化收益	Alpha	Beta	Sharpe	Sortino	Information Ratio	Volatility	最大回撤	Tracking Error	Downside Risk
280.271%	60.167%	44.514%	13.866%	0.5541	0.0423	1.3476	1.2706	0.9274	0.4142	-21.289% (2015-06-24, 2015-07-01)	0.4993	0.4393



从回测结果对比来看可卖空时收益曲线相对平滑点，最大回测也相对较小，其中亏损来源于价差无法收敛，所以最大回撤在于价差在极端值时出现，这就是价差非平稳时进行套利的弊端，当数据集是平稳序列时此会改善很多。

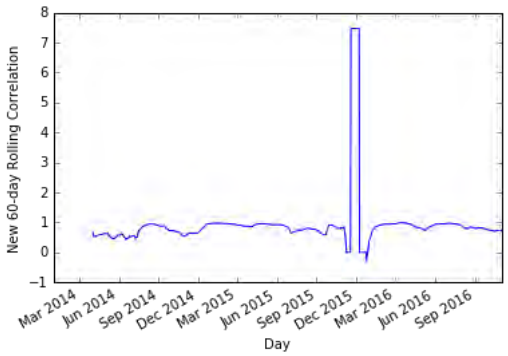
中国中冶和中海发展在2014-1-1和2016-11-1的相关性是0.968.让我们计算其60日滚动相关性来看看两者如何变化。

In [8]:

```
rolling_correlation_cn = pd.rolling_corr(a1, a2, 60)
rolling_correlation_cn.plot()
plt.xlabel('Day')
plt.ylabel('New 60-day Rolling Correlation')
```

Out[8]:

```
<matplotlib.text.Text at 0x7f8454e8ba58>
```



从相关系数的60日滚动相关性来看其实这两只股票相关性高是因为他们在2015年一段时间同时停牌，造成这段时间的相关系数为1，从而拉高了整个时间段内的相关系数，不过两只股票同时停牌也能说明一定问题，比如他们可能会有重组的可能，当然这是后话了。 不过选择股票对时应尽量选取没有停牌的股票。

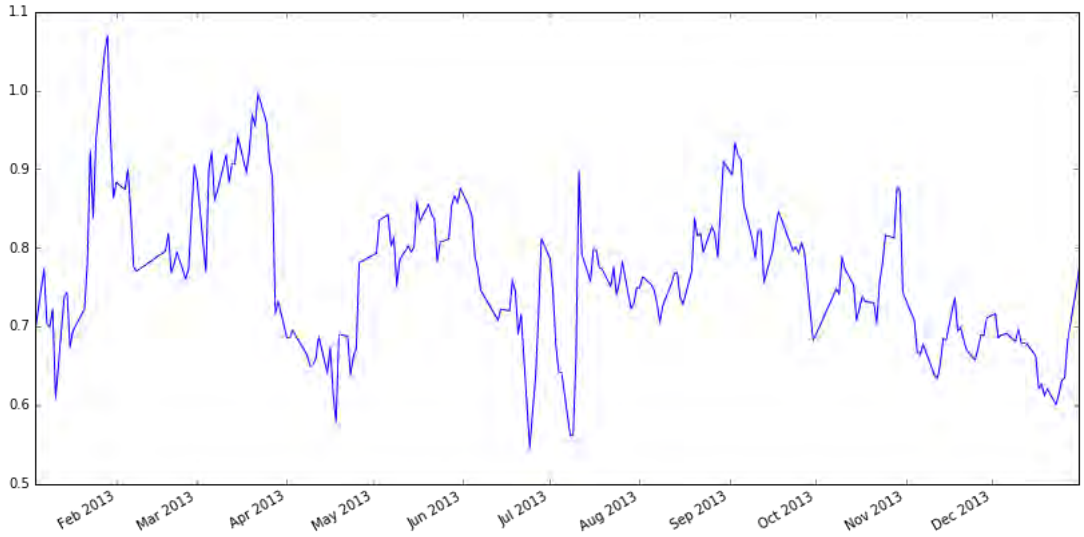
以一个协整关系比较好的股票对601169与601328来作为对照，看看它们之间的差价变动情况。

In [9]:

```
begindate="2013-01-01"
endDate="2014-01-01"
data1=get_price('601169.XSHG', start_date=begindate, end_date=endDate,fields='ClosingPx')
data2=get_price('601328.XSHG', start_date=begindate, end_date=endDate,fields='ClosingPx')
diff=data1-data2
diff.plot(figsize=(14,7))
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8454e95e10>
```



这两只股票满足协整性，作为配对交易的股票对比较适合，接下来下一部分我们就会谈到利用协整关系来套利

统计套利（二），利用协整关系进行配对交易¶

在前一篇当中利用相关系数来进行套利，看到价差并不为平稳序列，回测结果也就不是很好，所以想到利用协整关系来构建股票的线性组合，使得股价差为平稳序列，从而在真正意义上构建一个套利策略。看到有其他小伙伴也做过类似研究，但是都是以样本内得到的结果去回测样本内的数据，所以会有一定的不真实性。此研究以14年到15年数据作为样本来检验协整性，当然协整关系是一个动态过程，所以检验出来的协整性只能在统计意义上满足样本内数据，但我们假设这个协整性还继续保持，以16年数据来进行回测来看看实际效果如何。 之前我们谈到了利用两只股票之间的相关系数进行配对交易，但我们能通过两只相关性较高的股票对之间的差价图看出，相关性高他们之间的价差并不一定会是一个平稳序列，简单来说我们无法利用这个不收敛的价差来进行套利。那么要进行真正统计意义上的套利得需要什么样的条件呢，那么就引出了我们的协整关系。首先谈协整我们得先知道平稳这一个概念，一个平稳序列是指数据的期望不会随时间改变，数据的方差与协方差不会随时间而改变，并且固定一个时间，往前与往后进行回归都是相同的。所以平稳的股票是有很多统计上的好性质，可以进行套利操作，当股价达到一定高度时就要卖出，当股价低于一定数值时就需要买入，也就是俗称的低买高卖。但是一般在现实生活中不存在平稳的股票，所以很难去预测他在什么地方是高点什么地方是低点，这也就解释了为什么这么多人都是买高卖低了。 虽然一般不存在平稳的股票，但是几支股票的线性组合就可能是一个平稳序列，这就是我们一般意义上提到的协整关系，通过一些股票的组合来构造平稳的序列来进行套利。但是这个平稳性也是在一定时间段来说的，我们只能根据以往的数据来判断在这段时间内是协整的，但是无法断言在之后交易的时段也是协整的。

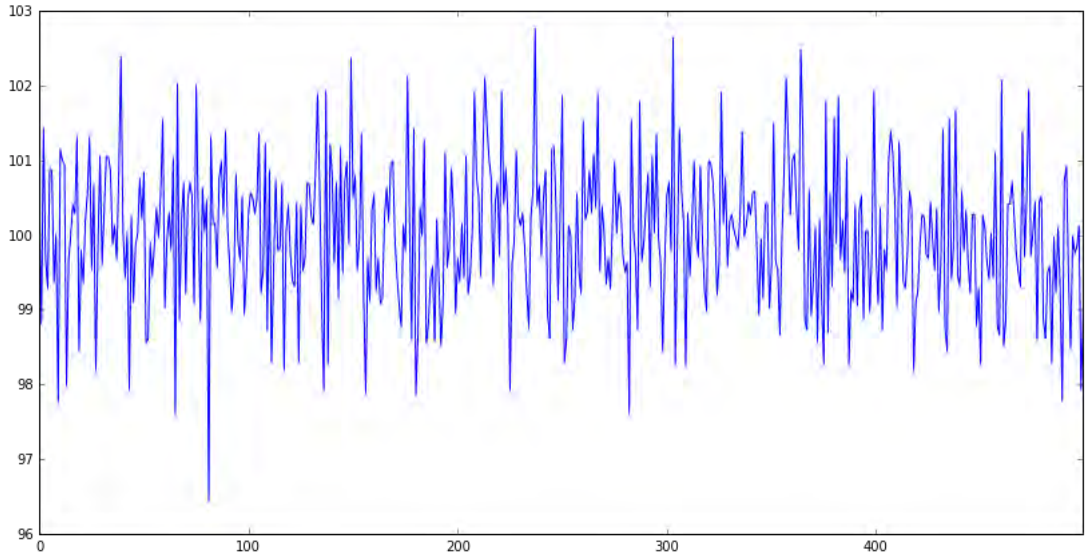
```
我们来只管看一下一个平稳的序列长什么样，最简单的一个噪声序列他就是一个平稳的序列。
```

In [1]:

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

np.random.seed(1234)
x = np.random.normal(0, 1, 500)
X = pd.Series(x) + 100
X.plot(figsize=(14,7))
```

Out[1]:

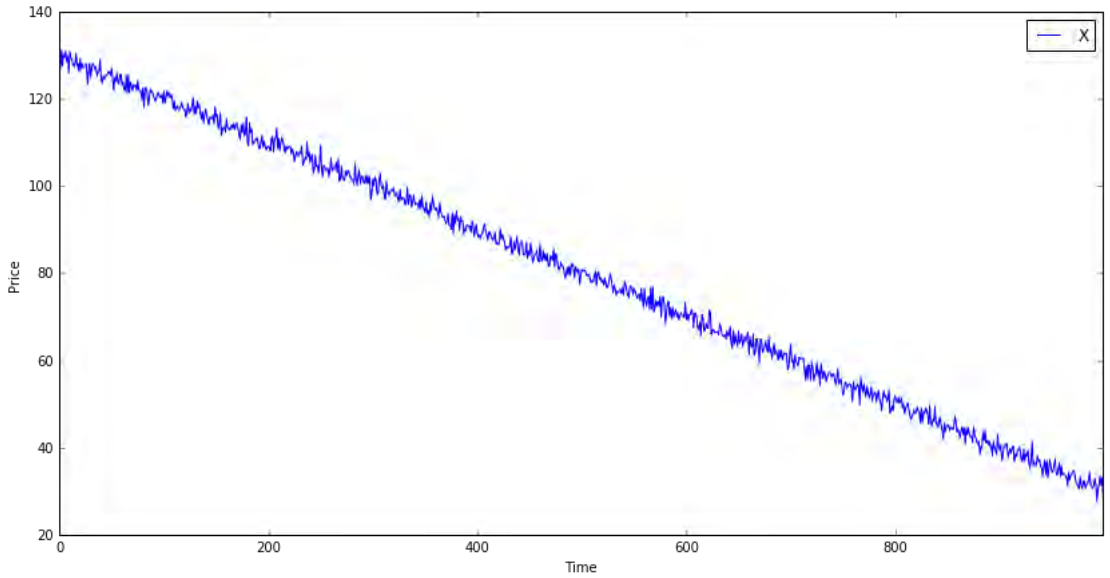
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdf8c8fd208>
```



上图构建了一个以100为均值的噪声序列，可以看出这个序列的期望与方差协方差均不随时间而改变。我们来看看两个序列本身是非平稳的序列是怎么样的。

In [2]:

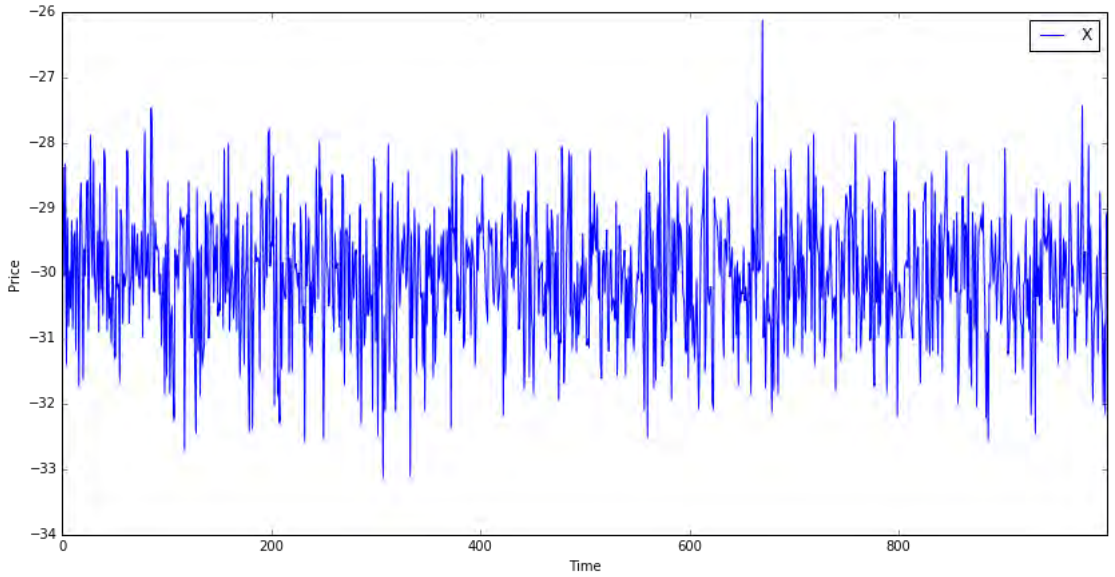
```
np.random.seed(1234)
x = np.random.normal(0, 1, 1000)
y = np.random.normal(0, 1, 1000)
X = pd.Series(x) + 100
Y = X + y + 30
for i in range(1000):
    X[i] = X[i] - i/10
    Y[i] = Y[i] - i/10
X.plot(figsize=(14,7));
plt.hold()
Y.plot(figsize=(14,7));
plt.xlabel("Time");
plt.ylabel("Price");
plt.legend(["X", "Y"]);
```



可以看出期望是随时间递减的，明显不是平稳序列，我们通过差分来看看是否能构造平稳序列。

In [3]:

```
Z=X-Y
Z.plot(figsize=(14,7));
plt.xlabel("Time");
plt.ylabel("Price");
plt.legend(["X", "Y"]);
```



通过差分我们得到的就是一个跟噪声序列长的一样的平稳序列了。 那么一般股票序列是非平稳的我们怎么去找协整关系构建平稳的序列呢。

In [4]:

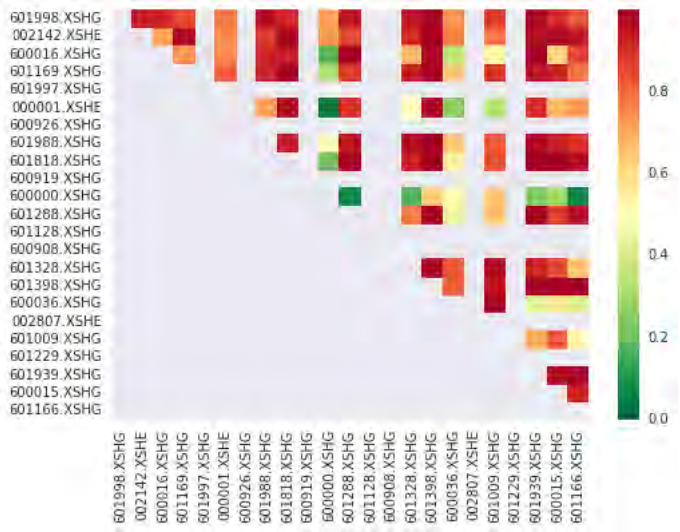
```
import statsmodels.api as sm
import seaborn as sns
def find_cointegrated_pairs(dataframe):
    # 得到DataFrame长度
    n = dataframe.shape[1]
    # 初始化p值矩阵
    pvalue_matrix = np.ones((n, n))
    # 抽取列的名称
    keys = dataframe.keys()
    # 初始化强协整组
    pairs = []
    # 对于每一个i
    for i in range(n):
        # 对于大于i的j
        for j in range(i+1, n):
            # 获取相应的两只股票的价格Series
            stock1 = dataframe[keys[i]]
            stock2 = dataframe[keys[j]]
            # 分析它们的协整关系
            result = sm.tsa.stattools.coint(stock1, stock2)
```

```
            # 取出并记录p值
            pvalue = result[1]
            pvalue_matrix[i, j] = pvalue
            # 如果p值小于0.05
            if pvalue < 0.05:
                # 记录股票对和相应的p值
                pairs.append((keys[i], keys[j], pvalue))
    # 返回结果
    return pvalue_matrix, pairs
```

In [60]:

```
#选取所需要分析的板块，比如我们要看能源行业中具有协整关系的股票对，就为J66，可以更改为自己想研究的行业，选取网址在这https://www.ricequant.com/d
selected_plate=industry('J66')
stock_list = list(selected_plate)
#以14到15年时间的数据作为样本看这段时间的协整股票对。
prices_df = get_price(stock_list, start_date="2014-01-01", end_date="2015-12-01",adjust_type='internal')['ClosingPx']
#获得强协整性股票对
pvalues, pairs = find_cointegrated_pairs(prices_df)
#以热力图来形象表示
sns.heatmap(1-pvalues, xticklabels=stock_list, yticklabels=stock_list, cmap='RdYlGn_r', mask = (pvalues == 1))
#打印出强协整股票对
print (pairs)
```

```
[('601998.XSHG', '601988.XSHG', 0.028768046322897745), ('601998.XSHG', '601398.XSHG', 0.013629324146940534), ('601998.XSHG', '601939.
```



从选出的股票里挑选p值最低的，也就是我们认为最符合协整关系的来看看他们之间的股价走势是怎么样的.

In [62]:

```
stock_sample1 = prices_df['601398.XSHG']
stock_sample2 = prices_df['601939.XSHG']
stock_sample1.plot(figsize=(14,7));
plt.hold()
stock_sample2.plot(figsize=(14,7));
plt.xlabel("Time");
plt.ylabel("Price");
plt.legend(['601398.XSHG', '601939.XSsHG']);
```

要利用这组股票对进行配对交易，就得看看他们之间的存在什么样的线性关系。这里我们通过股价进行最小残差平方和（OLS）方法来进行回归。

In [63]:

```
x = stock_sample1
y = stock_sample2
X = sm.add_constant(x)
result = (sm.OLS(y,X)).fit()
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            601939.XSHG   R-squared:                       0.978
Model:                            OLS   Adj. R-squared:                  0.978
Method:                 Least Squares   F-statistic:                 2.028e+04
Date:                Sun, 20 Nov 2016   Prob (F-statistic):               0.00
Time:                        06:56:26   Log-Likelihood:                179.27
No. Observations:                 467   AIC:                             -354.5
Df Residuals:                     465   BIC:                             -346.2
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          -1.2796      0.045    -28.141      0.000        -1.369    -1.190
601398.XSHG     1.5254      0.011    142.402      0.000         1.504     1.546
==============================================================================
Omnibus:                      137.194   Durbin-Watson:                   0.225
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              676.507
Skew:                           1.192   Prob(JB):                    1.25e-147
Kurtosis:                       8.393   Cond. No.                         26.6
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

r-squred为0.978，说明我们拟合的参数解释了97.8%的样本数据，说明拟合结果比较好，AIC与BIC检验较小说明回归效果好，再来看看系数是否统计显著，const的系数的绝对值除标准误差std err明显大于自由度为465的t统计量，说明统计显著。至此，我们能认为我们回归得到的结果是比较好的了。

接下来我们看看线性组合的价差是否是在均值周围波动。

In [67]:

```
diff=y-1.5254*x
mean=np.mean(diff)
std=np.std(diff)
print(std)
up=mean+std
down=mean-std
time=diff.index
mean_line=pd.Series(mean,index=time)
up_line=pd.Series(up,index=time)
down_line=pd.Series(down,index=time)
set=pd.concat([diff,mean_line,up_line,down_line],axis=1)
set.plot(figsize=(15,7))
```

```
0.164835349769
```

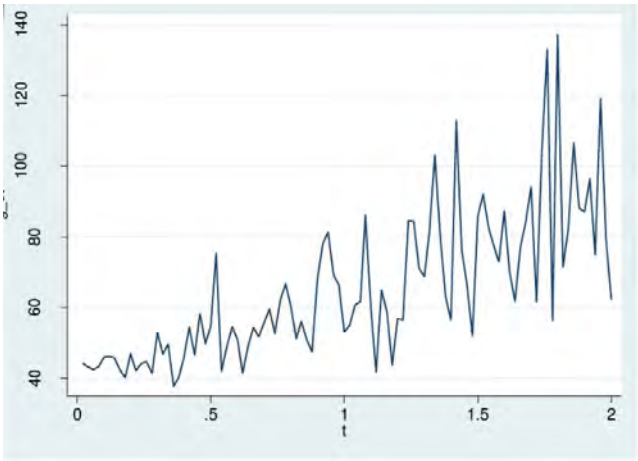
Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdf609ac470>
```


```
至此，我们完成了配对交易的准备步骤，找到了协整股票对之间的线性关系以及股价差满足的模型，我们以'601818.XSHG'的股价减去拟合的系数0.6319倍'601988.XSHG'的股价，
```

【期货策略】经典的跨期套利策略

根据经典的配对交易原理，只要两只标的的价差格序列满足平稳性，那么这两个标的可以构造成一个很好的配对交易对。从股票之中找这样的股票对在之前写的策略中有实现——链接在这统计套利（二），利用协整关系进行配对交易 ，那么利用不同期限到期的股指期货会不会更有效的，因为不同期限的股指期货的标的都是同一个指数，根据套利定价理论，他们的必要收益率应该相差不大，因为不同期限的合约除了需要承受的时间风险更多其他都是相同的，假设全体股市这一整体是服从维纳过程，他的漂移项就应为无风险收益——利率



那么自然时间更长的合约的按照利率折现后会比时间更短的合约多出时间的风险溢价。长期来看风险溢价是一个稳定水平，那么他们的价格波动自然会趋于一致性。所以很自然构建出利用跨期套利策略，差价绝对值高于一定水平就开仓，差价收敛到平均水平就平仓。

下面就是策略具体实现了，我们先选择适合作为跨期套利的合约，利用协整性检验，观察得到的p value

In [4]:
```
start='2016-07-01'
end='2016-07-15'
current=get_price('IC1607', start_date=start,end_date=end, frequency='1d').ClosingPx
next=get_price('IC1608', start_date=start,end_date=end, frequency='1d').ClosingPx
print(current)
print(next)
# get_price('000300.XSHG',start_date=start,end_date=end,frequency='1d')
```

```
2016-07-01    6039.8
2016-07-04    6159.6
2016-07-05    6202.4
2016-07-06    6226.6
2016-07-07    6279.4
2016-07-08    6265.0
2016-07-11    6248.0
2016-07-12    6368.6
2016-07-13    6416.4
2016-07-14    6442.2
2016-07-15    6439.8
Name: ClosingPx, dtype: float64
2016-07-01    5941.6
2016-07-04    6058.8
2016-07-05    6105.0
2016-07-06    6126.6
2016-07-07    6170.2
2016-07-08    6160.8
2016-07-11    6149.6
2016-07-12    6252.8
2016-07-13    6310.2
2016-07-14    6336.8
2016-07-15    6319.0
Name: ClosingPx, dtype: float64
```

In [5]:
```
import statsmodels.api as sm
import seaborn as sns
result = sm.tsa.stattools.coint(current, next)
# 取出并记录p值
pvalue = result[1]
print(pvalue)
```

```
0.00539825418891
```

In [6]:
```
start='2016-07-01'
end='2016-07-15'
current=get_price('IH1607', start_date=start,end_date=end, frequency='1d').ClosingPx
next=get_price('IH1608', start_date=start,end_date=end, frequency='1d').ClosingPx
print(current)
print(next)
# get_price('000300.XSHG',start_date=start,end_date=end,frequency='1d')
```

```
2016-07-01    2105.0
2016-07-04    2138.8
2016-07-05    2145.6
2016-07-06    2144.2
2016-07-07    2145.8
2016-07-08    2137.8
2016-07-11    2146.0
2016-07-12    2193.4
2016-07-13    2192.0
2016-07-14    2192.4
2016-07-15    2191.4
Name: ClosingPx, dtype: float64
2016-07-01    2088.6
2016-07-04    2124.4
2016-07-05    2130.8
2016-07-06    2126.0
2016-07-07    2128.4
2016-07-08    2121.6
2016-07-11    2134.2
2016-07-12    2185.8
2016-07-13    2186.6
2016-07-14    2186.8
2016-07-15    2186.8
Name: ClosingPx, dtype: float64
```

In [7]:
```
import statsmodels.api as sm
import seaborn as sns
result = sm.tsa.stattools.coint(current, next)
# 取出并记录p值
pvalue = result[1]
print(pvalue)
```

```
0.322612303218
```

In [2]:
```
start='2016-07-01'
end='2016-07-15'
current=get_price('IF1607', start_date=start,end_date=end, frequency='1d').ClosingPx
next=get_price('IC1608', start_date=start,end_date=end, frequency='1d').ClosingPx
print(current)
print(next)
# get_price('000300.XSHG',start_date=start,end_date=end,frequency='1d')
```

```
2016-07-01    3101.0
2016-07-04    3170.0
2016-07-05    3172.8
2016-07-06    3184.2
2016-07-07    3185.6
2016-07-08    3171.4
2016-07-11    3176.6
2016-07-12    3262.6
2016-07-13    3272.4
2016-07-14    3270.8
2016-07-15    3276.4
Name: ClosingPx, dtype: float64
2016-07-01    5941.6
2016-07-04    6058.8
2016-07-05    6105.0
2016-07-06    6126.6
2016-07-07    6170.2
2016-07-08    6160.8
2016-07-11    6149.6
2016-07-12    6252.8
2016-07-13    6310.2
2016-07-14    6336.8
2016-07-15    6319.0
Name: ClosingPx, dtype: float64
```

In [3]:
```
import statsmodels.api as sm
import seaborn as sns
result = sm.tsa.stattools.coint(current, next)
# 取出并记录p值
pvalue = result[1]
print(pvalue)
```

```
0.451551627557
```

可以看到IF与IH的合约的p value都比较高，都不能拒绝原假设——序列不是协整的，但是IC合约具有很低的p value 可以作为我们跨期套利的合约。 在策略中，为避免极端情况下，差价平稳性实效导致严重亏损，在设定的时间窗口的长度上再去检验当月合约与下月合约在这段时间内是否是具有协整的，如满足协整形才进行开仓操作，但是即使不满足也可平仓，因为在极端行情时价格就不满足我们构建的模型了，所以此时还是尽早出场为妙。为避免价差持续扩大造成亏损加重，设置一个止损的临界值，为1.1倍标准差。

在构建的这样的portfolio下，是对于价格的一次差分进行建模，相比于对价格趋势建模更容易，在维度上也实现了降维操作，也就无惧市场是晴是阴，只需考虑市场是否还有效。

https://www.ricequant.com/community/topic/2223/

商品期货跨品种研究2.0¶

本研究主要内容：【研究较长，可重点关注多组合部分】 

序言¶

这里是南方科技大学量化协会带来的第1篇干货，之前协会成员 L_7Joseph为各位筐友带来了跨品种策略1.0版本，现在， L_7Joseph欠大家的2.0版本来啦。

我们协会每周1~2次为大家带来纯干货，欢迎大家关注南方科技大学量化协会。

回顾¶

首先我们先复习一下关于期货跨品种的一些概念。

套利原理： 通俗地讲，就是两个合约相关性很好，突然市场出了一个bug，破坏了两个合约之间的平衡状态，进场套利；等待市场回复，平仓出场。

简单来说，也算一种均值回复的想法。

专业的说法是：跨品种套利是指利用两种（或多种）不同的、但相互关联的商品之间的合约进行套利交易，即买入某一商品期货合约，同时卖出另一相互关联的商品期货合约，以期在有利时机同时将这两个合约对冲平仓获利。

套利类型： 此处仅关注跨品种套利 

跨品种套利的鲁棒性¶

在上一篇文章中，我们利用相关性选择了如下三对组合：CU~ZN,CU~RU,Y~P；并且对其中组合Y~P进行了实际回测。在2.0版本中，我们放松了阈值约束，扩大了研究对象，以测试策略思路在多组合上的表现。

这里不得不对各位筐友坦诚了： 上一篇研究中，我们仅以利用相关性进行品种选择，但其实这是不完整的，一个完整的组合选择应该是：在以相关性强的前提下做协整分析，为了研究快捷方便，我们在此先略去这个过程，之后的研究中，我们会深入研究协整检验。

回到潜在套利组合的选择上，我们放松限制，看看有多少组合、

In [1]:

```
label_shanghai=['CU','AL','ZN','PB','NI','SN','AU','AG','RB','WR','HC','FU','BU','RU']
label_dalian=['A','B','C','CS','M','Y','P','JD','FB','BB','L','V','J','JM','I','PP']
label_zhengzhou=['WH','PM','CF','SR','TA','OI','RI','MA','FG','RS','RM','ZC','JR','LR','SF','SM']
```

In [2]:

```
label_total=[]
[label_total.append(each+'88') for each in label_shanghai]
[label_total.append(each+'88') for each in label_dalian]
[label_total.append(each+'88') for each in label_zhengzhou]
```

Out[2]:

```
[None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None]
```

In [3]:

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

In [4]:

```
start_date='20110101'
end_date='20140101'
data=[]
```

In [5]:

```
for each in label_total:
    data.append(get_price(each,start_date=start_date,end_date=end_date,fields='close',frequency='1d'))
```

In [6]:

```
futuredata=pd.DataFrame(data,index=label_total).T
```

In [7]:

```
futuredata.head()
```

Out[7]:

	CU88	AL88	ZN88	PB88	NI88	SN88	AU88	AG88	RB88	WR88	...	RI88	MA88
2011-01-04	72400.0	16895.0	19750.0	NaN	NaN	NaN	307.81	NaN	4822.0	4780.0	...	NaN	NaN
2011-01-05	71280.0	16840.0	19475.0	NaN	NaN	NaN	299.95	NaN	4793.0	4779.0	...	NaN	NaN
2011-01-06	72020.0	16950.0	19840.0	NaN	NaN	NaN	298.91	NaN	4809.0	4790.0	...	NaN	NaN
2011-01-07	69780.0	16850.0	19130.0	NaN	NaN	NaN	297.01	NaN	4793.0	4556.0	...	NaN	NaN
2011-01-10	70040.0	16880.0	19115.0	NaN	NaN	NaN	299.21	NaN	4834.0	4701.0	...	NaN	NaN

5 rows × 46 columns

In [8]:

```
badfluidity = [each+'88' for each in ['WH','RI','LR','JR','FB','BB','PB','SF','SM','SN','BU','WR']] # 流动性差的品种
```

In [9]:

```
futuredata.drop(badfluidity,axis=1,inplace=True)
```

In [10]:

```
futuredata.shape
```

Out[10]:

```
(725, 34)
```

In [11]:

```
futuredata.dropna(axis=1,inplace=True) # 去除数据不全的
```

In [12]:

```
futuredata.shape
```

Out[12]:

```
(725, 18)
```

In [13]:

```
# 查看相关系数
corr=futuredata.corr()
```

In [14]:

```
# 画个热力图
import seaborn as sns
import matplotlib.pyplot as plt
```

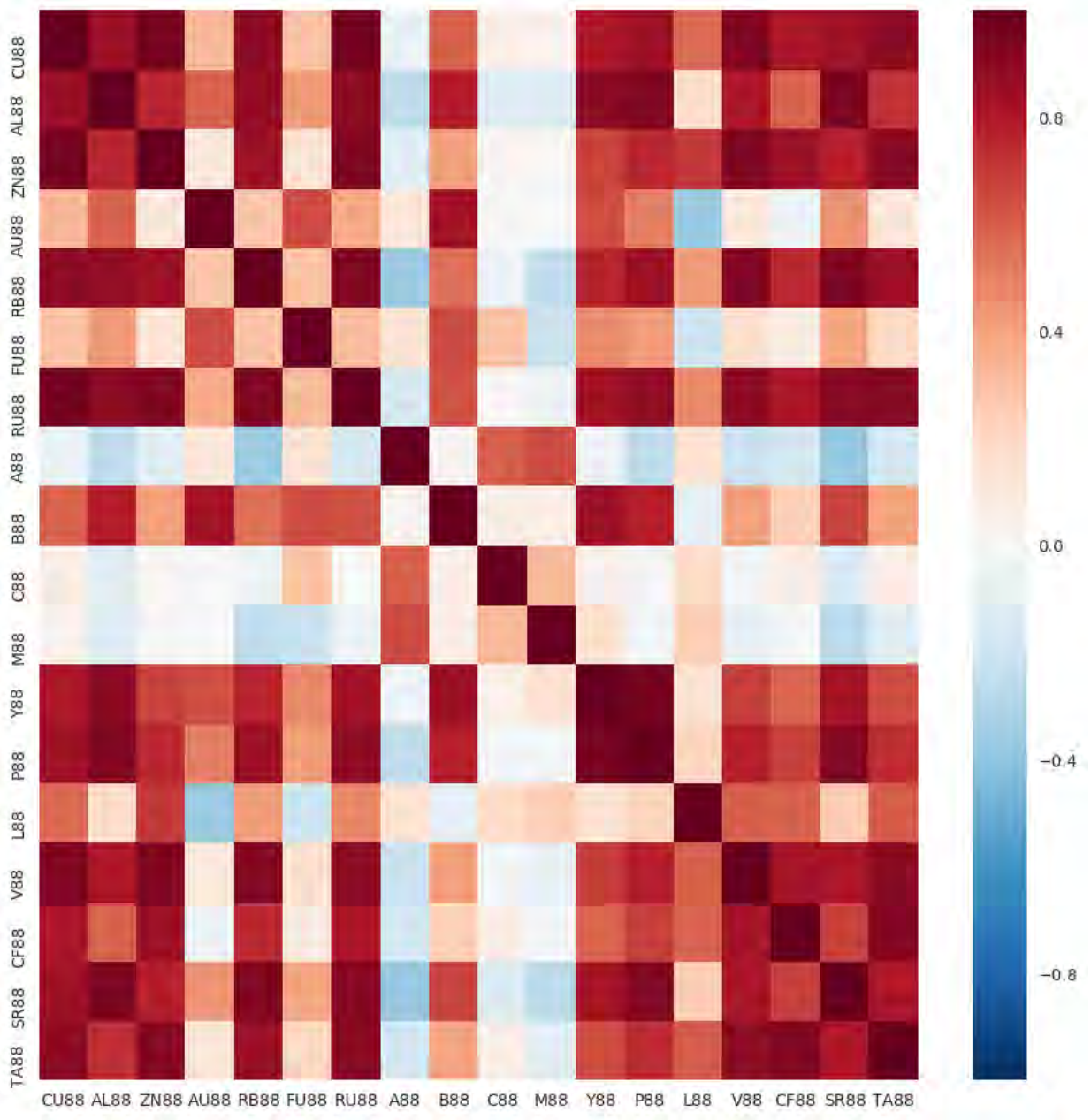
In [15]:

```
plt.figure(figsize=[12,12])
sns.heatmap(corr.values,xticklabels=corr.index.tolist(),yticklabels=corr.index.tolist())
```

```
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f710c892b70>
```



In [16]:

```
print ('相关系数大于0.8的组合对数有' +str((np.sum(corr.values.ravel()>0.8)-18)/2)+'对')
print ('相关系数大于0.9的组合对数有' +str((np.sum(corr.values.ravel()>0.9)-18)/2)+'对')
print ('相关系数大于0.95的组合对数有' +str((np.sum(corr.values.ravel()>0.95)-18)/2)+'对')
```

```
相关系数大于0.8的组合对数有42.0对
相关系数大于0.9的组合对数有17.0对
相关系数大于0.95的组合对数有3.0对
```

In [17]:

```
corr[corr>0.9] # 查看相关系数大于0.9的配对
```

Out[17]:

	CU88	AL88	ZN88	AU88	RB88	FU88	RU88	A88	B88	C88	M88	Y8
CU88	1.000000	NaN	0.954196	NaN	NaN	NaN	0.967376	NaN	NaN	NaN	NaN	NaN
AL88	NaN	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.9046
ZN88	0.954196	NaN	1.000000	NaN	NaN	NaN	0.925969	NaN	NaN	NaN	NaN	NaN
AU88	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
RB88	NaN	NaN	NaN	NaN	1.000000	NaN	0.927235	NaN	NaN	NaN	NaN	NaN
FU88	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN
RU88	0.967376	NaN	0.925969	NaN	0.927235	NaN	1.000000	NaN	NaN	NaN	NaN	NaN

	CU88	AL88	ZN88	AU88	RB88	FU88	RU88	A88	B88	C88	M88	Y8
A88	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN
B88	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN
C88	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN
M88	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN
Y88	NaN	0.904692	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.00000
P88	NaN	0.926772	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.95658
L88	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
V88	0.924649	NaN	0.922217	NaN	0.914545	NaN	NaN	NaN	NaN	NaN	NaN	NaN
CF88	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SR88	NaN	0.943198	NaN	NaN	0.923246	NaN	0.910413	NaN	NaN	NaN	NaN	NaN
TA88	NaN	NaN	0.906723	NaN	NaN	NaN	0.911670	NaN	NaN	NaN	NaN	NaN

获取到17对组合： 

从这里可以看到，各种关系错综复杂，单独调一对出来，相关性都挺高的，但有的解释起来可能没有那么容易，可能存在潜在的伪相关问题等。我们也再次忽略掉，日后再专门讨论。

回测规则：

- 时间：20140101~20170101
- 资金：50,000
- 开仓条件：上下两倍标准差
- 平仓条件：滑动均线
- 频率：分钟线
- 回溯周期：60
- 滑点：0

共测试了7个组合。

Y88--P88： 

RS--OI: 

M--A: 

JM--J: 

OI--Y: 

P--SR: 

Y--AL: 

楼主没有去进行17组回测，实在是苦力活，感兴趣的筐友可以挨个尝试一下。从7组回测结果来看，在暂时不考虑滑点的情况下，套利思想在空间上的鲁棒性还是很强的。

【日间回测？？？】

In [ ]:

```
```

滑点在套利中的影响，坏的影响¶

众所周知，在做交易时，滑点是我们不得不考虑的问题，尽管我们已经采用了流动性极好的合约，但是，有时候，看到，就是买不到，就是这么蛋疼。所以，楼主也测试了在经典万二滑点下策略的表现情况。

结论是：部分组合在万二滑点下依旧强劲，部分组合在万二滑点下基本就稳定亏钱了。【波动小的原因】【或者从每笔平均收益来看？？】

测试结果如下：

Y--P组合，滑点为0： 

Y--P组合，滑点为万二： 

J--JM组合，滑点为0： 

J--JM组合，滑点为万二： 

为什么在万二滑点下，YP组合就跪了而JJM组合还能坚挺呢?

In [17]:

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

In [18]:

```
start_date='20170222'
end_date='20170224'
Y88=get_price('Y88',start_date=start_date,end_date=end_date,fields='close',frequency='1m') #

P88=get_price('P88',start_date=start_date,end_date=end_date,fields='close',frequency='1m') #
```

In [23]:

```
spread=Y88.values-P88.values # 合约配比为1
```

In [24]:

```
window=60
entry_score=2 # 历史标准差的两倍  设置的越高 开仓信号越少；越低开仓越频繁
std=np.zeros(spread.shape)
mean=np.zeros(spread.shape)

up_limit=np.zeros(spread.shape) # 上界
down_limit=np.zeros(spread.shape)# 下界

for i in range(len(spread)):
    if i>=window:
        std[i]=np.std(spread[i-60:i])
        mean[i]=np.mean(spread[i-60:i])
```

```
        up_limit[i] = mean[i] + entry_score * std[i] # 开仓上界
        down_limit[i] = mean[i] - entry_score * std[i]# 开仓下界

    if i<window:
        mean[i]=spread[i]
        up_limit[i] = spread[i]
        down_limit[i] =spread[i]
```

In [25]:

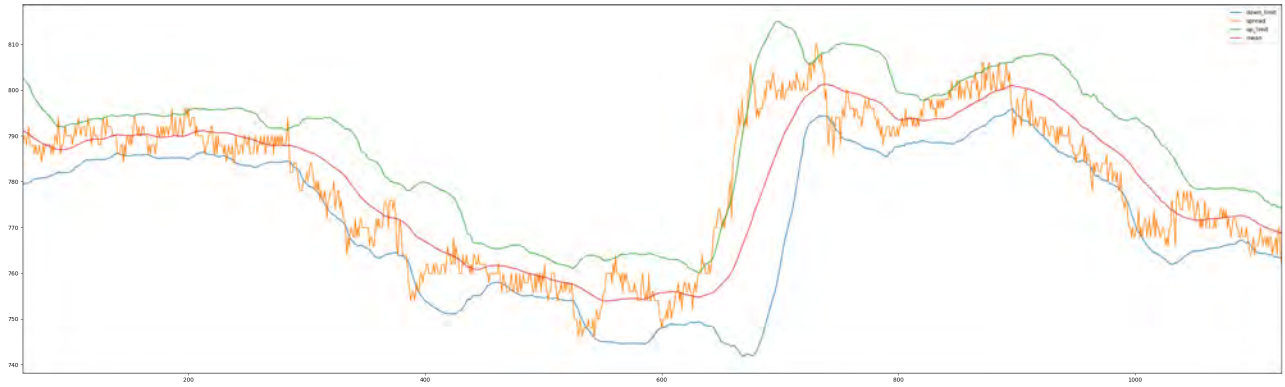
```
data=pd.DataFrame([down_limit,spread,up_limit,mean],index=['down_limit','spread','up_limit','mean']).T
```

In [26]:

```
data[window:].plot(figsize=[40,12])
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1451088470>
```



In [13]:

```
start_date='20170222'
end_date='20170224'
J88=get_price('J88',start_date=start_date,end_date=end_date,fields='close',frequency='1m') #

JM88=get_price('JM88',start_date=start_date,end_date=end_date,fields='close',frequency='1m') #
```

In [27]:

```
spread=J88.values-JM88.values # 合约配比为1
```

In [28]:

```
window=60
entry_score=2 # 历史标准差的两倍  设置的越高 开仓信号越少；越低开仓越频繁
std=np.zeros(spread.shape)
mean=np.zeros(spread.shape)

up_limit=np.zeros(spread.shape) # 上界
down_limit=np.zeros(spread.shape)# 下界

for i in range(len(spread)):
    if i>=window:
        std[i]=np.std(spread[i-60:i])
        mean[i]=np.mean(spread[i-60:i])

        up_limit[i] = mean[i] + entry_score * std[i] # 开仓上界
        down_limit[i] = mean[i] - entry_score * std[i]# 开仓下界

    if i<window:
        mean[i]=spread[i]
        up_limit[i] = spread[i]
        down_limit[i] =spread[i]
```

In [29]:

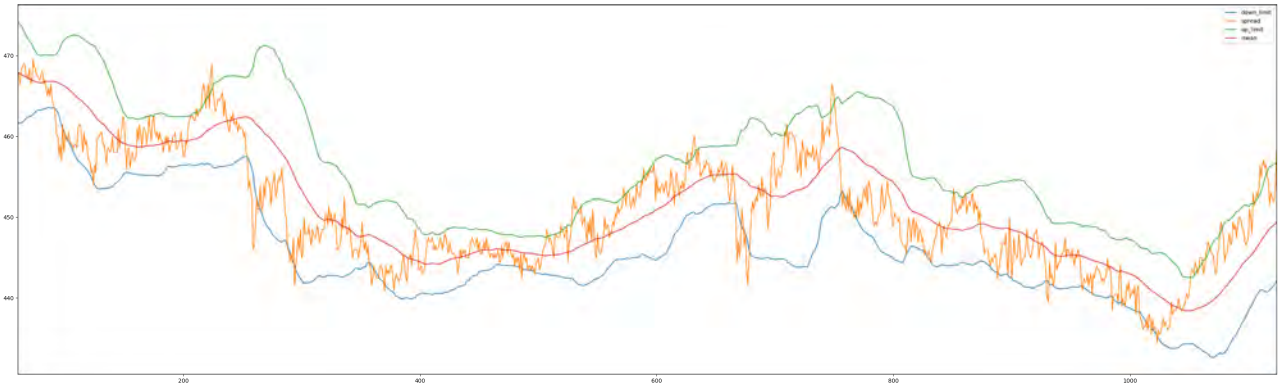
```
data=pd.DataFrame([down_limit,spread,up_limit,mean],index=['down_limit','spread','up_limit','mean']).T
```

In [30]:

```
data[window:].plot(figsize=[40,12])
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1451446c88>
```



对比以上两个图片，我们可以发现，YP组合的平仓信号基本都很突破得非常勉强，而JJM组合的信号相对比较强烈。

当然，楼主觉得这个问题如果需要细致研究的话，得去研究每一笔交易的成交情况，去研究每次平均收益或者单笔收益/单笔费用比值。楼主也查了米筐API，好像想要细致研究每一笔交易撮合情况比较难，楼主也就放弃了。

关于滑点问题小结吧：

套利策略的一个很强的假设就是：微小的利润积累，如果始终以不利的价格成交，那么微弱的利润区间必将被吞噬。所以，如果想把套利策略付诸于实盘的话，一方面，对对象的流动性首先得有强有力保证，其次，有力的IT支持也是必要的。

当然，楼主之前也有了解过一些异步价差的问题、延时成交的问题，如果有机会，再与筐友分享。

开平仓条件的优化方向¶

在上一篇研究中，提到希望能够通过优化开平仓条件来优化收益率曲线，在此我们做了一次尝试。

回到策略示意图： 

在我们的测试策略当中，取得是$a=b=2$，即双方对称开仓；平仓条件也同时取为均衡价差；

于是，根据策略思路我们可采取如下可能的优化方向：

- 设置静态非对称开仓条件
- 设置动态非对称开仓条件（正在探索）
- 从上至下均值回归时，下移平仓条件，即 $\mu \rightarrow (\mu -c \sigma)$
- 从下至上均值回归时，上移平仓条件，即 $\mu \rightarrow (\mu +d \sigma)$
- 当然，上诉c,d也可分别研究动态与静态，eg: 若动能较大时才移动平仓条件，动能较小时不改变平仓条件。

非对称开仓，上2下1.5倍标准差¶

测试对象： J-JM

滑点万二，其他同上。

测试结果： 

对称开仓结果： 

可以看到，对开仓条件进行一些简单的更改，策略表现能够得到一些提升：无论是回测收益、最大回撤、夏普率。但是，如何获得最优的开仓条件则需要一些优化工具来实现了。感兴趣的筐友可以手动多调调。

改变平仓条件¶

更改条件： 将双向平仓线都拉升0.3σ，滑点万二，其他条件不变。

更改后： 

更改前： 

可以看到，拉升平仓条件在本次测试中并没有带来额外收益，但由于楼主的处理比较粗糙，这里仅做玩具策略测试。

当然，还是那句老话，如果想进一步提升策略表现，就不得不去研究每一笔的成交、获利、成本等问题，进一步结合一些优化算法，可以获得历史最优参数。

多组合探索¶

终于到了最后精彩内容了哎！

测试以上内容简直费神费力，并且没有带来显著提高。 

于是，楼主只能把各个策略合在一起了，听人说，这样好像比较靠谱。

理论支持

假设我们有n个投资组合，年化收益率分别为$R_1,\cdots,R_n$,年化波动率分别为:$\sigma_1,\cdots,\sigma_n$，最大回撤分别为：$D_1,\cdots,D_n$。

Case1: *全仓某一个投资组合*i

- 年化收益率:R_i
- 年化波动率：σ_i
- 最大回撤：D_i

Case2:*均匀持仓*n*个投资组合*

年化收益率：

$$R_{total} = \frac{1}{n}\sum_{i=1}^{n} R_i \geq n\ \ min\{R_i\}$$

年华波动率：

$$\sigma_{total} = \sqrt{\frac{1}{n^2}\sum_{i=1}^{n}\sigma_i^2 + \frac{1}{n^2}\sum_{i,j,i\neq j}^{n}\rho_{i,j}\sigma_i\sigma_j} \leq max\{\sigma_i\}$$

最大回撤：

$$D_{total} \leq max\{D_i\}$$

回测时间： 20140101~20170101

滑点：万1

品种：['RS88','OI88','J88','JM88','Y88','P88','OI88','Y88','M88','A88']

对称两倍标准差开仓，不调整平仓条件。

回测结果： 

备注：由于测试的初始资金量不一，风险指标可能不适合用于直接比较。仅供参考。

各位筐友可自行添加组合进行回测，代码可扩展性很强的。

In [ ]:

In [ ]:

In [ ]:

宏观交易入门

宏观交易101第一课丨货币银行体系的资金和信贷

宏观交易员的自我修养之一

货币银行体系的资金和信贷

所谓宏观交易员，是指根据对经济增长、通胀、货币政策和财政政策等宏观观点，在债券、外汇、股指、商品等资产之间寻找机会的交易员。他们往往下注某一类资产相对于其他资产高估/低估，而非某一类资产自身的涨跌。需要特别指出的是，一般我们不认为宏观交易员需要做证券选择策略，例如股票投资中选股策略或者债券投资中的发行体资信研究。因为宏观交易员常用的策略和技巧，与证券选择策略所需要的技巧，互通之处往往很少。

那么宏观交易员最常用的技巧是什么？与很多人的认识相反，我认为Macro Trader看家本领不是宏观经济学，更不是对宏观经济的预测能力。宏观经济学涵盖许多极其宽泛和深远的领域，但是对Trader，我们只关心谁会拿着钱来买我手头的头寸以及如何揣测对方的报价。所以这时候第一步需要了解钱是如何在整个货币体系中间流动的。钱涌去的地方都好商量，钱流出的地方自然要先溜为敬。而且作为Trader，靠着吃饭的技术就是如何在Flow上煽风点火。其实前面CNH的栗子，说的就是宏观交易员心照不宣的一个规则，那就是货币银行模式对宏观交易员的意义可能远大于经济基本面。老前辈Paul Tudor Jones教育我们说，the whole world is simply nothing more than a flow chart for capital. 如果想要理解Flow，我们需要用货币银行学的方法，分析整个货币银行体系的资金和信贷。

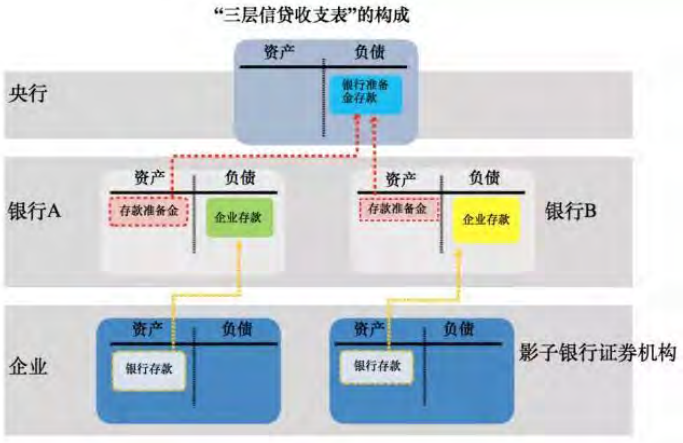
（接下来进入Bibi阶段。诸位需要理解熊猫没有受过正经的科班经济学训练，所有Economics Credits加起来才10个小时，所以离谱的Bibi之处还望指正）

1.商业银行货币信贷体系Flow Chart

回顾一下，货币银行体系最基础的概念是信贷。在没有信贷机制的交易市场，比如居民住房市场，有多少房子就只能买卖多少房子。在有信贷机制的市场，例如货币银行领域，银行即使现在只有比较少的现金，也可以向银行的存款客户承诺大量的兑付。如果流动性管理的好，存款量可以比现金量大十倍百倍甚至千倍。

贷款创造存款是信贷扩张的方式。每次银行向社会客户贷款时，将现金划转到社会客户的账户，形成社会客户的存款，后者再用于支付。这个机制是贷款创造存款，Debit和Credit同时发生。在没有制约的情况下，银行可能希望利用有限的现金无限制扩张Debit和Credit，获取存贷款利差。如果银行负债扩张太快，偿付风险或者流动性风险大幅升高。所以监管部门对银行往往要施加限制，其目的是保障银行负债的安全（此处默默黑2016年1月的CNH存款保证金政策）。最常见的限制是存款准备金，每次信贷创造过程，央行都要求冻结一定比例的资金。此外，对银行信贷收支表中各个项目计算风险，通过银行资本充足率制约银行信贷收支表的扩张，也是常见监管方式。有些货币银行体系，就不采取存款准备金的概念，只计算风险资本。

接下来有两个概念将在Macro Trader的视野里发生重要作用。第一，Debit和Credit扩张和收缩的基础是基础流动性（在货币银行体系里，我们把基础流动性叫做备付金）。第二，Debit和Credit必须是同时扩张或者收缩的。



这个图是熊猫最喜欢的图，作者是华润银行的王超先生，描述的是如何根据信贷收支表理解Fiat Money 的信贷流动机制。在Fiat Money体系里，所有基础货币都是央行的负债。当央行认为存在通缩、流动性恐慌、银根紧缩的风险时（或者Whatever需要放水的理由），央行会选择扩张自己的负债，以此创造更多的基础货币。回顾Debit和Credit同向扩张的概念，央行的扩表还需要扩张资产。最常见操作是利用增发的货币向下一级商业银行购买资产（例如再贷款）。商业银行收到央行的基础货币之后，再以贷款创造存款的方式，用这些基础货币做备付金，扩张出数倍于备付金的信贷。同样这些信贷如果只停留在商业银行层面，也无法形成扩表，必须在企业层面扩张信贷才能成为扩表。企业承担了信贷成本，获得货币信贷资金后，正常情况下，将其用于运营或者投资，企业手里的货币信贷（这时应该称作企业的营运资金）越多，越容易抬升商品和服务的价格，反常情况么，口可口可...可以反思一下2014年以前为什么CNH利率会偏离CNY那么远。

基础货币的流动性，而不是整体信贷规模，是银根松紧的关键。由于银行负有通过信贷机制为基础货币创造流动性的责任，银根收紧一般都是银行体系造成的。我们提到企业拥有的是银行创造出的信贷货币Credit，而不是基础货币Money（除非不开银行账户，只收现金屯在床垫下面），但企业结算，尤其与账户开在其他银行的其他企业结算，是通过基础货币进行的。如果经济活动中突然产生了大量结算需求，一开始货币利率会自然上行，银行由于担心将来的货币利率会进一步走高而尽量减少结算支付，只向企业或者上家催收。这时银行在收缩而不是创造流动性，就会引起银根紧缩。这种形式的银根紧缩并不罕见，在19世纪美联储成立之前，缺乏中央银行的美国货币体系发生过多次，甚至频繁到6、7年左右就有一回。现代货币银行体系中有专门调控流动性和货币利率的中央银行，就把这种无序银根紧缩的现象压制住了。

但是否这种无序银根紧缩现象是否已经被彻底消灭，不需要我们关注了？答案是不仅没有，而且还在央行够不着的地方越玩越High。例如离岸货币体系其实就没有旨在调控流动性和货币利率的中央银行（此处默默黑某央行），所以离岸体系比在岸体系脆弱的多。离岸货币体系的基础货币，往往是由在岸银行搬运的，其主要目的是套利（即使是离岸美元，套利因素也很重）。可以想象，当离岸市场的商业银行出于信心坍塌而收缩信贷的时候，仅仅只用在岸离岸之间的息差吸引同样的商业银行多搬运些在岸货币来抚平离岸市场的银根紧缩，效果将有多差。2008年离岸美元市场的流动性坍缩，其实严重程度远大于美国本土，联储应对的方式是向ECB/BoE/BOJ/SNB/BoC等央行发放大规模的Swap，向离岸供应天文数字一样的美元，再加之各家央行的主要目的是金融稳定而不是套利，才压制住Eurodollar市场的恐慌。有心人士可以对照FED的H4.1表，看看当时投放的USD流动性有多大量。

（离岸货币的部分再继续扯下去，Macro 101怕是收不住了，所以离岸市场，或者说缺乏央行调控的货币银行体系的部分先到这里，反正这个题目其实很常见，之后的2a-7案例更加精彩）

（没忍住，回来再扯一句，现在很多人怀念金本位制，各位Macro Trader想好了在金本位制下如何做Trade吗？提示：读Lords of Finance）

在货币银行体系里，银行由于承担基础货币流动性的职责，所以银行体系的结构是宏观交易员观察Flow的重点，尤其是影响流动性向企业的信贷收支表传播的环节。试举几个观察点如下。第一个，我们非常关注准备金，因为存款准备金影响可用于清算的基础货币总量，直接影响整个货币体系的流动性。需要指出的是，所有影响超额备付金的因素，与存款准备金的地位是相等的。这个方面的例子有2015年8月份央行释放以外币形式缴纳的存款准备金（参阅《熊猫笔记20150917》），又如央行的各种粉。第二个观察点是央行投放基础货币的效率，尤其是在中国这样以量调控而不是价调控的体系。在2013年以前，央行对市场的操作往往通过少数几家大行（包括国开行），如果大行处于各种动机，更希望囤积流动性而不是释放流动性，那么流动性投放的效果就大打折扣。2014年以后，央行创造了各种三个字母的花样投放流动性，很多原本很难从央行获得流动性的银行甚至地方性银行都有渠道获得流动性，作为一个Trader，应该意识到央行流动性投放的效率高了很多。第三个观察点是银行本身的行为。例如债券交易员都非常关注季度末的状况，是因为银行处于出季报核验风险资本的要求，希望尽可能的缩表。2015年10月以后，央行采取平均法考核，这样降低了季度末因素对流动性的影响，注意我在这里没有用"相当于1次降准"这种说法，因为我看到的是结构的调整而不是总量的调整，不应简单类比。

信贷流动性进入到企业层面后，花式会更加多样，Macro Trader分析股指和商品等资产往往需要由此入手。这是最容易产生混乱的地方，在此我的建议是以推演微观层面的企业扩表收表决策入手，不宜简单只看宏观经济数字的变化。比如所谓FED QE印钱流入股市推高股指的谬误长盛不衰，但是如果结合Money and Credit的流动分析微观，我们会意识到QE其实效率极低，联储扩张的基础货币负债大部分停留在商业银行的超额准备金上，导致美国货币信贷体系的超额准备金从2007年的不到500亿美元飙升到2013年的2万亿美元以上，并没能形成超级信贷。但是在微观层面，企业一方面意识到资金利率极低，另一方面也坚信联储在出现经济下行风险时会源源不断提供流动性托底，所以敢于承担更高的财务风险，比如发债回购股票。同时，债市的低收益率会显得股票那点EBITA格外具有吸引力，所以证券投资者们会愿意给股票一个更高的估值。

在正常增长的经济体，企业看到的是投资收益大于机会成本，有动力增加融资并将资金投入营运，企业的CAPEX产生对劳动力和商品的需求，因此推高通胀。多么美妙的故事，可惜现在满地都是不正常的经济体，正常的经济体反而显得不对劲了。最著名的是日本式的资产负债表衰退，日本企业不敢扩大CAPEX，而是将赚到的Retained Earnings拿来补泡沫经济时候留下的窟窿。（著名的奥林巴斯事件，董事会辛辛苦苦做了十几年的帐，被一个好死不死的英国CEO刺破了，还是自己人可靠啊）。一个有意思的现象是日本的利率曲线长期维持在G7中几乎最陡的水平，有些Analyst不假思索的说陡峭的利率曲线反应对通胀的预期，但我不喜欢这种教科书式的说法，通过观察各个期限段的需求，我倾向于认为这是资金严重泛滥，短端利率无节操下行造成，而长端多少还能维持不那么难看的供需。

第二个例子是QE大放水之后的美国企业。按照教科书，企业应该被低廉的利率刺激的勇于一博，上马若干看起来不赚钱的CAPEX项目，从而带动通胀（参阅Bernanke VS Summers（2015））。可是牛不喝水强按头它也不喝，更何况精明的人民群众？反正拿到钱之后也没有定向的说法，为什么不做一点更轻松舒服而且有利于股价的事情，比如说回购股票？如果大量企业把钱拿来做回购股票而不是CAPEX支出，那就是一个通缩因素。

第三个例子更有意思。仍然考虑企业对主营业务没有信心的情形，如果投机成本低、企业有钱但又不愿意发给股东，他会干什么？说不定我们可以考虑干些投机。投机讲究快进快出，所以显然不能做重资产的，这时候金融资产就显得有得天独厚的优势了。在这种情况下，金融资产尤其是企业最熟悉的商品领域会出现大幅的波动，甚至超过宏观的趋势。另一个投机现象是别家地里比自家肥，制造业想做房地产，房地产进军金融业，金融业搞直投...这种Mislocate现象不仅中国有，欧美市场也有，不过欧美市场更多是投资组合资产配置方面的Mislocate.

（又越扯越远了。简单一点说，Macro Trader需要结合Micro层面的信息才能做好Flow分析，只看宏观数据瞎做绝对死菜。提示一下，B家的研究中经常援引上市公司季报中Management Discussion and Analysis的内容。）

因为这个方面的材料很多，所以我不赘述了，只特别说一点。在中国做FICC领域的PM/Trader，往往对这些货币银行现象观察较欧美成熟市场的更深入，这应该说是学习Macro Trading的时候，不成熟市场相对成熟市场的一个优势。我在美元和人民币市场都做过很长时间的债券PM。我的一个体会是，美元市场的自修复能力实在太强，压制了很多货币银行波动现象，所以做美元的Trader尤其是国债Trader，对这些基础的Flow结构并不需要特别关心。这不见得是个好事，好像开车，如果开Tesla，会丧失很多宝贵直觉，形成开车就是操作一系列软件的误区。而在中国市场做债券，就好像开大货车，需要熟知发动机/变速箱/油路/路况甚至天气...

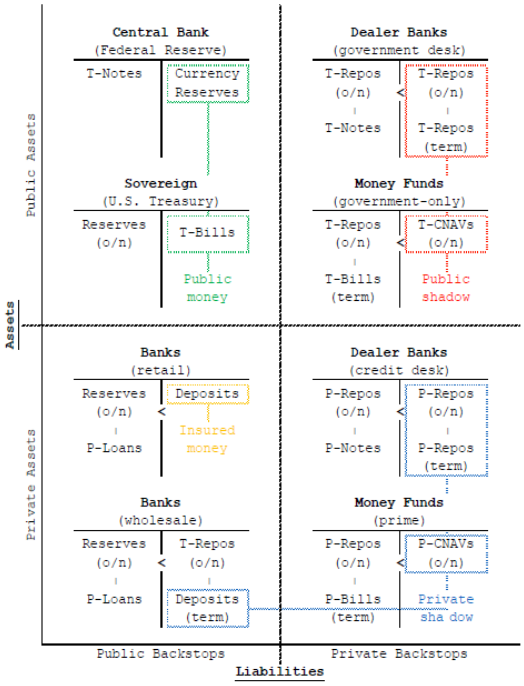
上面提到，中央银行和商业银行的信贷收支，是Macro Trader最重要的观察点。这一方面的资料其实极大丰富，因为主要的央行都会定期发布中央银行和商业银行的数据统计。读中央银行和商业银行的数据，是Macro Trader的一项基本功。建议各位读者都认真读一下主要央行发布的金融数据，至少读一下FED和PBoC的。读FED数据的时候，建议对照FED Z1 Flow of Funds数据读，更有利于建立对美元货币体系在央行/银行/企业传导机制的理解。结构和机制比总量重要！结构和机制比总量重要！结构和机制比总量重要！重要的话说三遍。所以熊猫一直反对简单比较FED/ECB/BoJ/PBoC的M0/M1/M2，这几个货币体系差别实在太大，风马牛不相及。此外，读中国的金融数据，也需要秉着结构和机制比总量重要的观点，专注于梳理信贷流向，而不是见到M0/M1/M2就一惊一乍。

在上面的分析中，我有意绕过了传统的货币银行分析方法如M0/M1/M2分析等。这一个方面是因为这些内容足够丰富，读者可以自行寻找，不需要我这样根本没受过科班训练的来Bibi（其实深刻的分析还是少，滥竽充数的多，对照Flow Chart就能发现市面上很多材料完全混淆了货币银行概念）。另一个方面，Macro Trader必须有能力改造拿到的模型或者数据，以求描绘一个更准确的图景。以企业信贷为例，企业持有的资金，是现金/通知理财/国债，对企业的决策有区别吗？在M0/M1/M2中可是完全不一样。

2.影子银行货币信贷体系Flow Chart

以上谈到传统的商业银行信贷Flow，在现代金融体系里，还有一个同等重要的角色是证券体系，或者说是影子银行信贷体系。其特点是证券承担了一部分信贷创造的功能。这个方面最详尽的一篇论文是Pozsar(2014)，实在是太全面所以就不再赘述，只简单引用一个图，记述几个要点。

Figure 1: The Money Matrix[15]



Source: Pozsar (2014)

要点之一，在影子银行信贷体系中，除了中央银行账户上的现金是基础货币，国债也同样能起到基础货币的作用。这是因为所有的现代货币银行体系中，国债都享有与现金等同的风险资本，而且，在主要的货币银行体系，国债交易和回购的流动性和清算效率极高，所以国债同样可以作为基础货币，承担一部分信贷派生的作用。由此立即有一个推论，那就是一个价调控而且债券清算效率高的货币体系，政策利率等于隔夜国债回购利率。国债回购利率的重要性甚至大于政策利率，因为政策利率往往只存在于央行和一小部分商业银行的特定操作中，但是国债回购利率将顺着债券交易一直延伸到资本市场每个角落，甚至影响到境外投资者，然后各种其他融资交易按照不同的Capital Charge，以国债回购利率为Benchmark定价。（黑一下所谓“利率走廊”的概念，熊猫认为给基准利率加上沿下沿是纯画蛇添足的想法，做利率市场化，建设好回购机制就行了。所谓“利率走廊”表明某央行仍然是一个量调控思维）。

其二，影子银行，或者说非银行金融机构，其交易结算，同样需要实物货币，而且影子银行虽然可以派生信贷，但是往往不能从直接央行获得信贷（极罕见的情况例外），所以其角色在前面所述的商业银行货币流动图里，可以认为是处在企业的层级。

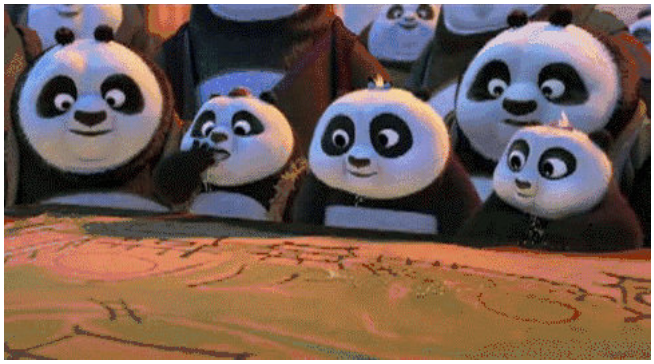
其三，影子银行参与货币信贷体系，但是影子银行体系的统计和运行机制远不如商业银行统计那么成熟。很多时候影子银行统计本身就是一个潜在障碍，甚至可能误导货币政策。

这些概念很枯燥。再枯燥也要学！因为忽视这个系统会严重缺失对Flow Chart 的把握。现在我们复盘2013年6月钱荒的时候，会众口一词认为央行收紧货币政策去杠杆，造成流动性崩塌。但是如果研读2013年2季度的金融数据，再对比阅读《货币政策报告》等重要的央行文件，会发现当时信贷和流动性似乎还是显著增长的，只不过增速略略下降而已。如果把影子银行拼回当时的图景，我的感觉是：央行的货币政策方向并没有发生改变（被广受攻击的那20亿央票更多是个心理作用），只是对影子银行体系的活动严重估计不足。而当时的货币银行体系，其实非常缺乏弹性，一方面流动性传导受大行的效率制约，另一方面又受季度末因素的影响。在商业银行体系上飞速生长的影子银行体系面临政策压制和季度末回表，都需要大量的货币资金完成结算，但央行流动性的供应慢于市场预期。再叠加缴税和一些偶然事件，最终导致缺乏弹性的商业银行体系出现大幅流动性收缩。（参阅《货币数量、利率调控与政策转型》）

在成熟市场，影子银行体系的自由更多，功能更强，因此效应也更多样，更需要Macro Trader认真观察。2009年联储大幅度开启QE的时候，很多参与者悲叹史无前例的恶性通胀即将来临，黄金被炒到1900的高位。然而6年之后的今天，美国通胀历经诸多折腾也只是朝着2%苦苦挣扎。这个方面的原因固然多样，但是从一个Macro Trader的角度，如果忽视商业银行体系和货币银行体系的效率，只按照基础货币谈通胀，绝对是有问题的。我们固然看到了商业银行体系充满了现金，但是企业并没有多少增加资金投入CAPEX的欲望，相反更喜欢拿钱回购股票。此外，影子银行体系一直在坍缩。资产证券化一直是美国企业和居民获得信贷的最高效影子银行渠道，但FED Z.1表显示，美国资产证券化市场的收缩一直持续到2014年。所以作为一个Trader，从商业银行Flow Chart和影子银行Flow Chart，观察到的都是通缩而不是通胀信号。

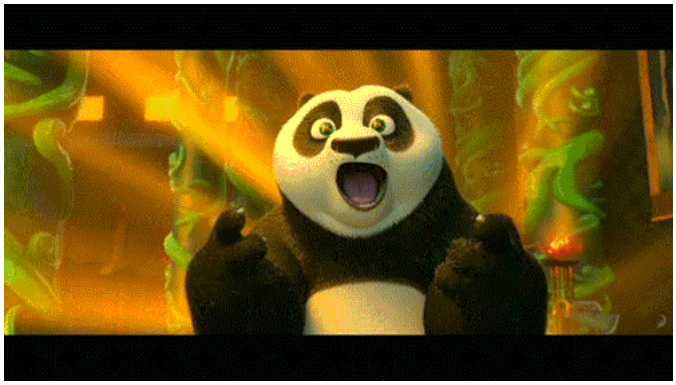
无论是商业银行体系还是影子银行体系，流动性都是Macro Trader最敏感的点。货币系统出现坍缩是以流动性枯竭的形式体现，在商业银行，其形式是银行守住那点宝贵的备付金不放，而不是扩表以贷款创造存款，在影子银行，是Dealer守住流动性好的国债不放，压价甚至拒绝接受可能砸在手里消耗宝贵流动性的公司债。这个过程不是简单的抛售风险资产，抛售是结果不是原因，而且将会在风险补偿上升到合理水平后停止。再度强调，Debit和Credit是同时收缩扩张的，不可能出现只注销Debit或Credit的情况。所以流动性坍缩是一个向Flow Chart上层收缩至基础货币的过程，这个过程有一个非常响亮的名字叫“Flight to Quality”（其实熊猫一直觉得叫做Flight to Liquidity更合适）。这个过程甚至可以创造出很多奇妙的现象。例如2011年和2013年美国国会和政府两度撕逼，一度导致美国国债面临很严重的兑付风险。我们都知道如果公司债传出兑付风险，肯定是腰斩腰斩再腰斩，然而虽然当时美国国债CDS一度升到很可观的水平，但是美国国债不跌反涨，股市和商品倒是跌的稀里哗啦，一度让参与者怀疑是否跌错了。需要注意，在国债违约这样的系统性风险事件面前，所有的金融机构都需要尽量保存流动性，因为完全不知道接下来可能是什么状况，万一货币体系要重构呢？所以银行收缩信贷，交易商抛公司债抢国债，这时候处在Flow Chart最下层的股市和商品自然要倒霉。股市不要喊冤枉，砸的就是你。

在传导机制灵活，自调节能力强大的成熟市场例如欧美市场，Flight to Quality形成的一系列反应被交易员们简化为Risk On/Risk Off，然后资金流和资产价格反应都以Risk Appetite的变化解释。这个做法的好处是极大简化了分析过程，但是也跳过了对市场结构的深入分析。此前我们提到，如果市场机制太顺畅，可能形成一个黑盒子，导致Trader丧失对市场基础结构的感知，最终会影响Trader的直觉。尤其是新的主题开始进入市场的时候，没有做过Flow Chart推演而是从上一代Trader那里学习Risk On/Off分析方法的新Trader们，可能时不时会陷入困惑。事实上，我经常听到勤奋而善于思考的Trader提出的质疑，Risk On/Off模式有效吗？是否应该考虑Decoupling？这时候，Flow Chart方法，就可以解释这些疑惑。



虽然有Lehman等事件的冲击，但是在现代的货币银行体系里，流动性风险的影响越来越大，甚至大于传统的信用风险，即使在Lehman事件中也是（我会告诉你Lehman债券最后回收率多高吗？）。这一方面是商业银行体系所受的监管日趋严格，监管资本、集中清算和有序破产等措施大大降低了信用风险对体系的冲击。另一方面，各国政府和央行在系统性风险上的投鼠忌器，使得越来越多的参与者相信信用风险一旦上升到系统性风险是央行必须出手挽救的。我甚至认为目前的Capital Charge已经足够吸纳潜在的Credit Event形成的冲击而不至于击垮系统，所以Flight to Quality往往可以在风险补偿上升到合理水平后停止。但是监管机构也不是轻易能挟持的。金融危机之后，旨在以宏观审慎角度加强银行监管的措施，例如Basel III等，对银行的流动性资产等提出了很多特别的需求。这些监管措施，在隔离系统风险同时，也极大限制了银行向市场提供流动性的能力，所以形成了一个有意思的现象，就是系统性风险传染性可能下降了，但是金融市场本身应对常规冲击的抵抗力也下降了。Macro Trader将越来越需要关注流动性可能创造的交易机会。

再多说一句，Trader对货币信贷体系中的变化，应该比鲨鱼闻到血还要敏感。货币信贷体系结构的每一点变化，都意味着有一批资金将不得不改道寻求新的路径，如果能够研判出货币信贷变化的传导路径，抢在市场之前Front Run 这批资金，就意味着坐地收钱。反之，如果被新的货币信贷体系逼着改变但是动作太笨拙迟缓，就意味着给鲨鱼们孝敬了一大笔血肉。这才是真正的Macro Trading.



未完待续??下周继续

宏观交易101第二课丨跨境资金流动

宏观交易101第三课丨宏观交易入门（附试题）

宏观交易101第四课丨Cash Study：货币基金改革如何隔山打牛

宏观交易101第五课丨Trading Skills

宏观交易101第六课丨Monetary Economics 考试题参考答案

宏观交易101第七课丨Derivative Securities & Other Skills for Macro Trading

宏观交易101第二课丨跨境资金流动

本周带来的是第二课——跨境资金流动

宏观交易101第一课丨货币银行体系的资金和信贷

Cross-border flows

跨境资金流动

跨境资金流动是汇率的决定因素，但是跨境资金的作用不仅在于汇率，也会对Flow Chart 形成冲击。这个方面，DM和EM有深刻的不同（DM：Developed Market 发达市场，EM：Emerging Market 新兴市场）。简单一点说，DM货币有跨境流动性，可以向EM扩表，而EM没有跨境流动性，只能默默承受。

由于货币系统本身是封闭循环的，所以跨境资金流，也是一个货币循环的过程，而且同样是Debit和Credit同时发生的过程。再次回顾货币银行学基础概念，只有居民和企业还掉债务的时候，才会发生资产负债表收缩，注销信贷的现象。但在跨境资金流中间，没有发生削减信贷收支表的行为。比如日本投资者向银行买USD卖JPY，卖掉的JPY不会被注销，而是回到银行循环，买到的USD也不是凭空冒出来的，而是商业银行增加USD信贷产生的。所以确切的说，这个机制不是单向的买USD卖JPY，而是是向商业银行抵押JPY，商业银行扩USD信贷收支表。这个过程，JPY的信贷收支表没有变化但是由于银行获得了JPY抵押品，JPY的流动性增加了。USD则出现了一个扩表过程。

既然是扩表，从用流动性好的硬通货扩向流动性不好的EM货币就是正道，反之用EM货币扩向硬通货就难于上青天。实际中也是如此。DM经济体金融市场基础建设好，市场流动性好。DM货币往往是硬通货甚至储备货币，可以在本国之外自由流通。其他经济体尤其是本国金融市场建设比较差的EM甚至会借助DM金融市场为本国融资。因此，DM的Flow Chart，可以扩展到本国之外。典型的例子比如美元，Macro Trader们一般认为美元主导了全球市场80%左右的跨境结算量。美元再QE再贬值，全球贸易商都要老老实实拿着美元。欧元的辐射范围不如美元，但是欧洲银行业在东欧和Latam等区域耕耘了上百年，也积攒了相当的影响力。同属国际化货币，日元的地位就尴尬一些，只在亚洲部分区域有影响力。强调一下，只有实体经济愿意接受货币支付，愿意持有货币，才能算是国际化。所谓国际化货币，就是那种即使你知道要贬值，你也要老老实实拿在手里的货币。所以，扩表方向决定了DM/DM、DM/EM和EM/DM是完全不同的Flow Chart.

1DM/DM

当一个DM货币流动性过于泛滥，资产收益率缺乏吸引力的时候，本土投资者抛本币买外币资产。本币信贷收支表不变，外币信贷收支表扩张。通胀虽然可能受扩表影响，但更受进出口价格影响。由于价格表现比货币表现更加直接，一般认为弱汇率货币是一个通胀因素。企业盈利方面，由于企业生产过程是一个（用本币计价的生产要素）创造附加值的过程，所以虽然生产成本可能因为进口原材料价格因素而上升，但是企业往往可以通过更有吸引力的出口价格获取竞争力优势。大部分DM经济体的上市公司都是这个类型，所以DM汇率贬值有利于提振股指。但有两点需要特别指出，一是DM经济体的货币全球流通，汇率对货币银行系统的收表扩表决策影响不大。EM没有这个自由，效果就复杂的多。二是前述汇率对股指的影响是很粗疏的，具体还是要结合经济体模式分析。假设有一个奇特的经济体，上市公司全是进口分销商，汇率贬值肯定对股指是打压而不是提振。

在画DM/DM的Flow Chart图时，我习惯于把两个DM并列，因为理解DM/DM之间的资本流动时，我认为DM之间资本流动冲击其实被金融市场弱化了，即使“货币政策外溢”是一个时下争论很激烈的话题。我注意到DM的金融体系相当完备，这样意味着任何超出合理水平的Capital Flow 都会引起相应的价格变化，套利成本的上升将阻碍一个DM货币向另一个DM货币套息。加拿大是一个很好的栗子。2009年以后FED开启了史无前例的放水操作，但是加拿大的债券、通胀和资产价格似乎并没有受到严重冲击。这一点作为Trader其实很好解释。2009年以后CAD升值的很快，让试图博取USD/CAD息差的套利资金无路可图。而传统上，G7货币之间的FX Alpha很难赚，很少有USD资金会连续大举博CAD汇率。所以，成熟的金融市场其实阻碍了DM之间的货币政策外溢，而Trader确实也没有观察到大量涌向CAD货币体系的USD。反面例子出在试图做资本管制的EM经济体。不管怎么管制，都很难胜过人民群众的智慧。满地泛滥的套利资金开始在2009年以后大举涌入EM经济体，轻松在缺乏调节能力的市场赚取套利。所以，我们见到了一个悖论。开放的资本市场，受DM货币政策外溢的负面效应较小，反而是封闭的EM市场，往往被DM的货币政策拖累。

最极端的跨境资金发生在负利率的情况下。回顾前述商业银行部分，需要刺激商业银行扩表的时候，央行降低资金成本，让商业银行从陡峭的利率曲线获取高额的NIM，希望商业银行喂饱了之后心满意足继续扩表，但是央行很快发现这个速度太慢，于是央行以QE\Operational Twist\Forward Guidance等方式压低长端收益率，迫使商业银行成倍扩表以维护Retained Earnings。可惜在流动性陷阱下，再多的举措也不管用，于是央行干脆对商业银行闲置在央行账户上的富余准备金施加一个负利率成本，以为这样可以拿鞭子抽商业银行扩表。但是商业银行无法将负利率成本移给存款客户，只能受损。极端情况下，商业银行甚至可能选择收缩信贷收支表以减少负利率成本，而将更多的资产转入证券市场。所以负利率往往有利于房贷等资产证券化效率很高的领域，另一些领域，如传统上完全依赖银行信贷的中小企业，反而可能会面临信贷更窘迫的状态。再考虑到负利率本身对货币信贷机制是一个重大改变，同样可能会导致担忧不确定性的商业银行向Flow Chart上一级收缩流动性。这一点，日本银行拆借市场在2月份BoJ负利率决议后意外停滞就是例子。

那么负利率有什么作用？熊猫一直认为负利率是一个汇率工具，导致投资者不仅不愿意持有负利率货币资产，而且努力在外汇市场以Forward/Swap 等方式增加负利率货币的负债（主要央行：我们只调控利率，汇率是市场行为）。但这种货币流动，同样也无助于增加实体经济尤其是中小企业的信贷。所以负利率的故事，可能会告一段落，一是因为商业银行会强力反对负利率，尤其是贷存比超低的日本银行业（贷存比动辄100%以上的意大利银行业表示情绪稳定），二是因为汇率战的负面效果已经在2016年得到共识了。

2.DM/EM

DM对EM的投资，一般是FDI和股权投资等形式，所需要的货币资本，轻松扩表就能获得。所以在DM货币体系Flow Chart里，我习惯于把EM货币体系直接放在DM企业。但是EM拿到的货币资本，大部分不能直接在EM经济体使用，需要向EM的银行兑换为本币，或者流入EM的央行形成外汇储备。所以DM对EM的货币资本，起到的是一个协助EM扩表的效果，资产端形成DM货币资产，负债端需要考虑兑付FDI和股权的资本回报。

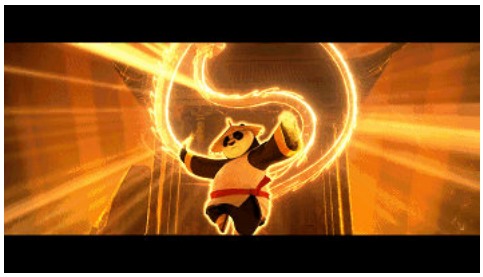
第一，资产端外币收益赶不上负债端FDI/股权总体负债成本，是EM货币银行体系的天生缺陷。这是因为注意EM的央行和银行，无论是银行的Treasury Management还是央行的Reserve Management，对安全性和流动性的考虑都远高于FDI和股权投资，多以债券和货币的形式投资，正常情况下回报率不如FDI和股权投资。好在正常的货币银行体系我们只需要管理流动性，甚至可以将总体兑付成本抛在脑后，毕竟货币银行体系天生就要扩张，而且不可能出现一个国家外资撤个精光的情况对不对？（阿根廷突然打了个喷嚏...）但是如果对EM的货币信心出现怀疑，DM撤资越多，总体兑付成本上升越快，天天嘀咕EM央行外汇储备数字的吃瓜群众如果按捺不住也要跑，将加剧EM货币信心的坍塌。

其二，需要注意到EM之所以需要DM资本是因为EM本身自生能力不足。所以资本自生能力不足的新兴市场，货币贬值往往是一个自我加速的过程，外部资本的投资回报受到威胁而撤资，对经济活动形成压制，同时货币贬值而带来输入型通胀，这是双斧劈柴，产出不足的窘境导致通胀越演越烈，为抑制通胀被迫加息收紧货币，进一步抑制经济，危机加速，再度增加贬值压力。还是一句老话，在你转型成功，强大而且健康到足以向境外输出资本之前，新兴市场跪求外部资本是一个上去就下不来的模式，即使知道自己被绑在美元战车上也要认。

第三，EM的资本流入普遍是顺周期的，而且往往流向股市、商品、房地产等投机领域，这是天生的金融不稳定领域，可想而知资本流出的时候破坏力多大。

第四，EM的资本流出，不可能是向DM支付EM货币，只能是银行和央行支付外汇，因此，EM企业层面的总体信贷收支表，将以相应的货币乘数缩表，这种猛烈的缩表，一般都很难软着陆，届时你会发现日本式慢性资产负债表衰退真是太美好了。

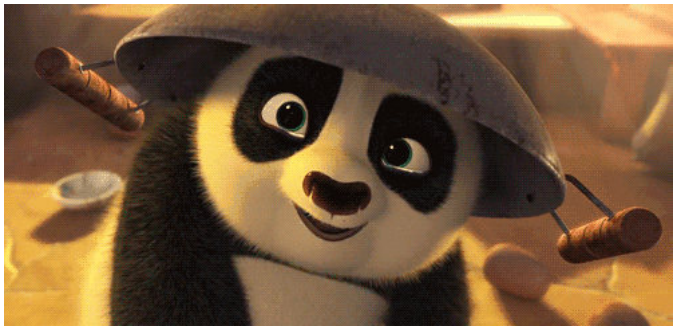
出于对跨境资本冲击的恐惧，很多EM以严厉的资本管制措施，试图限制资本流动。前面我们说到，资本管制阻碍市场正常调节，反而协助DM向EM套息套汇。资本管制更要命的危害是扼杀外汇流动性。资本管制预期下，出于用汇恐慌，所有的进出口商会尽可能的虚报进口瞒报出口以获取外汇，反而会导致外汇加速流出。利用资本管制恐慌摧毁汇率是极为高明的做法，因为最庞大的羊群是本国企业和居民。卢布2014年的危机就是例子。其实俄罗斯的外汇储备、政府赤字和外债等宏观指标其实都极其健康，但是居民对1998年的恐惧还在，在卢布危机的时候跑路的那个叫快。顺便说一句，1998年俄罗斯债务危机，对洋人的债务其实很快就足额偿还了，但是对本国银行和居民是真的赖了帐了。搁上这种宁赠友邦不与家奴的2B，你要是俄罗斯人你能不怕？



3.EM/DM

考虑到EM无法向DM扩表，但正常的经济体又必须维持信贷收支表正常增长。所以EM几乎所有的BoP举措，都是旨在维护一个较为稳定的经常项目流入（如果是印度那样的奇葩，那就是资本项目流入）。那么处在资本外流威胁下的EM该怎么办？主要手法还是提振自身的资产回报率，咬牙吸引外资。2013年，时任RBI行长Rajan为了对抗印度资本外流，采取加息以及奖励外资进入印度债市的措施都颇为可圈可点。还有一些上不了台面的技巧，例如熊猫就曾经脑洞大开设计过旨在稳住境内外汇流动性的Chinadollar体系，建议央行亲自印美元白条。（参阅《熊猫笔记20160719》）

缺乏货币流动性的EM货币体系，不仅要通过低廉的劳动力和原材料为DM货币体系支付资产回报，而且还时不时要承受DM货币政策灌下来的翔。所以熊猫一直对人民币国际化抱有超乎寻常的期望，可惜...



4.结束语

Fiat Money体系下，央行和金融机构的信贷创造和流动性管理存在着若干先天的缺陷。观察这些缺陷，推演资金流将如何被这些缺陷扭曲再Front Run这部分资金，是Macro Trader的饭碗。在写作本文的时候，我正好读到宏观金融学巨擘John Cochrane的一篇Blog，《没有现金的世界》，描绘了一个通过数字货币和区块链搭建的货币体系，这个体系是如此完美，乃至金融体系天生就稳若磐石，央行轻敲键盘立即化通胀于无形，流动性召之即来挥之即去。A world too perfect for monetary economists.　虽然这个图景看起来色彩实在太未来，估计远在若干世纪以后，但是我们似乎已经看到那时Macro Trader作为一个职业的终结。

文末彩蛋??来自熊猫交易员的课程PPT：

Skills for Macro Trading

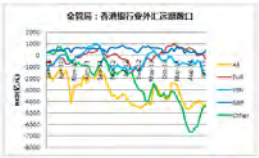
A Lecture in Guanghua *Financial Economics* Course, May 2016

Global Marco Investing

- Fundamental driven, directional bets. Based on inflation, growth, monetary policy, fiscal policy, etc.
- Global equity indices, bond markets, currency markets, and commodity markets. Usually do not take security selection views.
- Why macro matters for all of you? Lessons from RV performance in subprime crisis.

CNH Revisit : A typical case of leveraging and de-leveraging monetary system

A Typical CNH Book
(Nominated in USD)



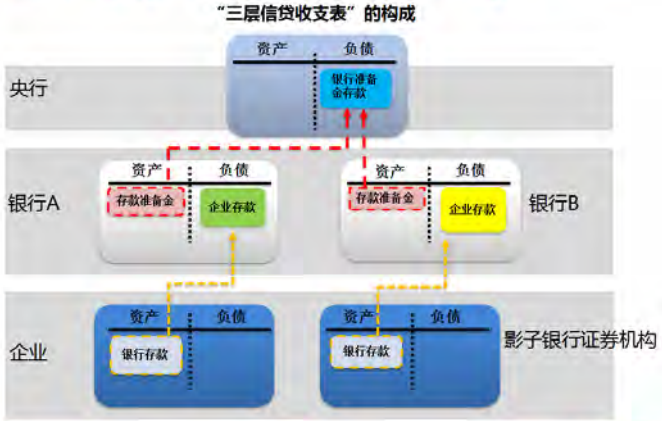
Confusing Voices in Macro: CNH Case, the higher the carry, the worse the currency

JPY Case: FX, FX Basis, Rates, or whatever...

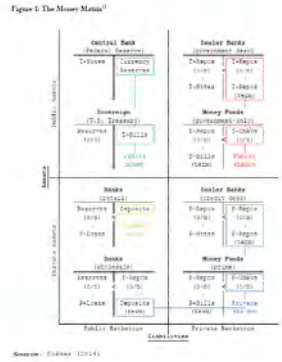
Skills : Monetary Economics

- The whole world is simply nothing more than a flow chart for capital. (Paul Tudor Jones).
- For a trader, money flow is more important than economic data. To get a clear picture of how money flows in the economy, we need monetary economics.

Monetary System: Commercial Banking



Monetary System: Shadow Banking



Case: June 2013 China Liquidity Crunch

Cross Border Case Study: Canada Post 2009

US VS CA: Core CPI

US VS CA: M2

US VS CA: Housing Price

Cross Border：DM VS EM

DM央行

DM金融机构

DM企业和居民

EM央行

EM金融机构

EM企业和居民

Cross Border: EM VS DM

Recommend Readings for Money and Credit

- 伍戈，李斌，(2015)《货币数量、利率调控与政策转型》
- Zoltan Pozsar (2014), "Shadow Banking: The Money View", Office of Financial Research Working Paper
- FED, "Factors Affecting Reserve Balances" H4.1, Weekly
- 中国人民银行，月度"货币统计概览"和"金融机构信贷收支统计"
- FED, Z.1, Financial Accounts of U.S, Quarterly

Skills : Macroeconomics

- Capture the momentum of growth and inflation.
- Skills:
  - The growth story.
    - Case: EM/DM VS EM exChina / EM, why BRICS is a misleading term.
    - Case: Housing in US VS Housing in China.
  - Data Mania
- Models:
  - CFNAI
- Hint: Fight in the short term, live in the mid term, forget the long term.

Recommend Readings for Macroeconomics

- Ray Dalio, "How the Economic Machine Works", http://www.economicprinciples.org/
- Chicago Fed National Activity Index, https://www.chicagofed.org/publications/cfnai/index
- James H Stock, Mark W Watson, (1998), "Forecasting inflation", Journal of Monetary Economics

Skills: Trading, academic part

- Trading != Buy low / Sell high.
- Who is who in this game.
- Volume and liquidity.
- Check, call, raise or fold. Best window for position unwinding.
- Market microstructure knowledge can help in liquidity and trade execution.

Case: Yen as a safe heaven asset

Case: Yen as a safe heaven asset (continue)

Recommend Readings for Trading

- Dennis Botman, Irineu de Carvalho Filho, W. Raphael Lam, (2013),"The Curious Case of the Yen as a Safe Haven Currency: A Forensic Analysis", IMF Working Paper
- Funke, M., Shu, C., Cheng, X., & Eraslan, S., (2015), "Assessing the CNH-CNY pricing differential: role of fundamentals, contagion and policy." , Journal of International Money & Finance
- Ludwig B. Chincarini, (2012), "The Crisis of Crowding: Quant Copycats, Ugly Models, and the New Crash Normal"
- Michael Lewis, (2011), "The Big Short: Inside the Doomsday Machine"
- Margin Call, (2011)
- Billions, (2016)

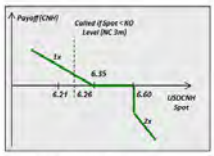
Skills: Derivative Securities

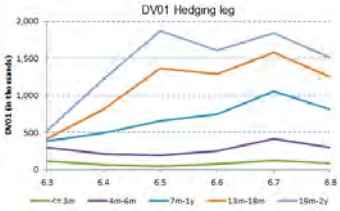
- Usually, macro PM do not need to be financial engineers or quants. Even as they trade large volume of derivatives, most of those instruments are plain vanilla.
- I do not expect macro PM to be stochastic calculus guru. However, as a significant part of trading comes from derivative related flows, the PM need to know what is happening.
- Especially in Asian markets where Greater China investors who like structured FX and FI products, Korean investors who like structured equity index products.

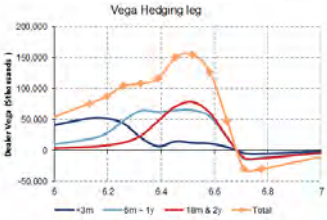
Case: CNH TARF

Tenor 2y / monthly settled
Payoff Client buys 6.35 USDCNH put
Client sells 2x 6.35 USDCNH call EKI @6.60

Large position unwinding will happen around CNH approach 6.5







Other skill sets

- Politics.
- Communication with policy makers.
- Quantitative Analytics.

General Readings on Macro Investing

- Steven Drobny, (2006), "Inside the House of Money: Top Hedge Fund Traders on Profiting in the Global Markets "
- Lasse Heje Pedersen, (2015), "Efficiently Inefficient: How Smart Money Invests and Market Prices Are Determined"

Case: Jan 2016 HKD Raid

HK: Political and Economical Downturn

- Political: Pan-democracy Camp.
- Economical: Tourist income. Financial sector.
- Uncertainties provide the best window for short sellers.

Are they coming for HKD de-pegging?

Hong Kong Economy and Stock Market

- High exposure on real estate and financial service.
- Mortgage market.
- Stock market: High concentration of banking and real estate developers.

Short selling strategies

- Build short positions in cash stock and stock futures.
- Short both HKD and HKD forwards.

Then…

- The intervention by HKMA decrease money supply M0.
- Capital outflow decrease money supply M0, hurt equity market.
- Liquidity crunch in HKD push up HKD HIBOR.
- 80% of the household mortgage was linked to HIBOR, resetting at a high HIBOR will hurt both the household and the housing market.
- Real Estate Sector and Banking Sector will get hurt.

The key is to break the HKD Forward Markets

Average daily turnover of the Hong Kong OTC interest rate derivatives market (by currency)

US$ billion

	Average daily turnover				
	April 2013	Share (%)	April 2010	Share (%)	Change (%)
US dollar	**3.5**	**12.5**	**3.5**	**19.8**	**0.5**
Euro	**0.9**	**3.3**	**0.5**	**2.8**	**79.7**
Japanese yen	**2.4**	**8.6**	**0.1**	**0.5**	**2,328.4**
Hong Kong dollar	**1.8**	**6.4**	**3.2**	**17.4**	**-44.6**
Others	**19.2**	**69.1**	**11.2**	**60.5**	**72.1**
of which: British pound	0.1	0.3	0.4	2.2	-81.1
Australian dollar	7.3	26.4	1.7	9.2	331.5
Chinese renminbi	2.0	7.3	0.8	4.5	146.1
– CNH	0.1	0.2	n.c.	n.a.	n.a.
– CNY	2.0	7.1	0.8	4.5	139.7
Korean won	4.1	14.9	4.7	25.6	-12.9
Indian Rupee	2.0	7.2	n.c.	n.a.	n.a.
Thailand Baht	1.5	5.4	n.c.	n.a.	n.a.
Others	2.1	7.6	3.5	19.9	-39.8
All currencies	**27.8**	**100**	**18.4**	**100**	**50.7**
Other OTC products	**0.1**		**0.02**		
Total OTC interest rate derivatives transactions	**27.9**		**18.5**		**51.1**

Source: HKMA

Allies and Enemies of short sellers

Allies

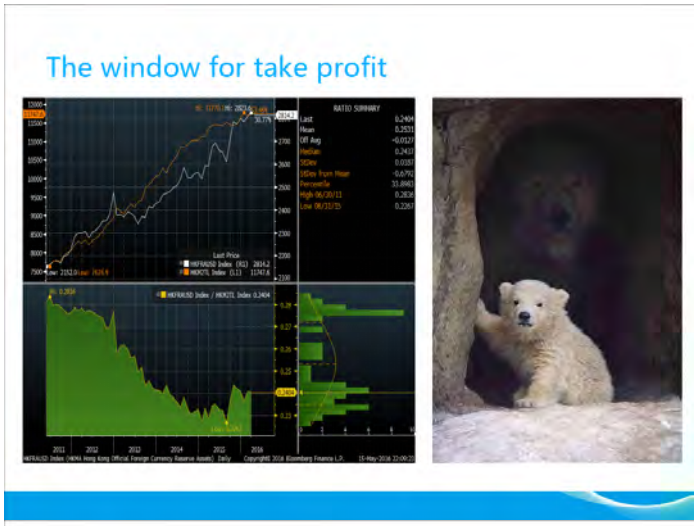
- Domestic investors who worried about the political uncertainties and HKD de-pegging risk.
- International investors who cared about the global risk off brought by CNY depreciation.
- Korean HSCEI structured products.

Enemies

- Value investors.
- Chinese investors who have preference on HK stocks.
- HKMA

The future of macro investing

- Old School Macro, such as George Soros who made global macro sexy, gradually close their book or turn into family office, as their investors can not tolerate the P&L swing.
- New School Macro are guys mostly grew up in the investment banks, either as prop traders or flow traders, or both. They trade traditional macro assets like FICC, trying to get attractive return with low VaR. However the majority of them were not able to achieve the expected return, compared with other hedge fund strategies.
- Tourist Macro are actually bottoms-up guys whose expertise was in another market sector, such as credit (e.g. John Paulson) or long-short equity (e.g. David Einhorn). They are so far the most popular "macro" managers.
- Unconventional monetary policies killed carry & gamma and regulation killed leverage. Traditional macro managers either have to face a low yield, or try security selection strategies they were not familiar with.
- However, with macro issues extending into all sectors, every hedge fund PM is expected to talk about how macro may change his/her area. That is why Tourist Macro become more popular, although their largest P&L contribution may not come from traditional macro strategies.
- Also, the largest uncovered part of macro volatility, CNY interest rates, exchange rates and capital flow, will introduce a lot of global macro stories in the near future. It is time for you guys to show off your China expertise in macro funds, or even tourist macro funds.



放映结束
-END-

未完待续??下周继续

- 《宏观交易101第三课》 -

宏观交易101第三课丨宏观交易入门

宏观交易第三课 | 宏观交易入门

拿来烤人的

Monetary Economics

习题

答题前，希望大家再温习一下前两课内容～

宏观交易101第一课丨货币银行体系的资金和信贷

宏观交易101第二课丨跨境资金流动

1. 文中提到“在没有信贷机制的交易市场，比如居民住房市场，有多少房子就只能买卖多少房子。在有信贷机制的市场，例如货币银行领域，银行即使现在只有比较少的现金，也可以向银行的存款客户承诺大量的兑付。”其实除了货币信贷市场，还有很多市场有类似的信贷机制，使得参与者可以以较少的基础资产，维持较大的偿付或者交割Claim。例如美国国债市场，由于债券回购市场很便利，国债交易员即使库存券很少，也可以向客户提供大量的Offer盘。国债市场之外，请再举出至少三个存在类似信贷机制的市场。中国银行间国债市场是这样的市场吗？

2. 如果把央行主动向市场投放基础货币的利率定义为基础利率下限，那么正常的货币银行体系，是否可能出现信贷利率低于基础利率下限的情况？如果出现了这种情况，意味着什么？作为一个Macro Trader，你打算如何布局头寸？

3. 你是FDIC主席。闺蜜FED主席说货币政策工具中有很多工具能影响银行风险管理，而银行风险监管工具没有能影响宏观通胀的，所以才有“宏观审慎管理”，不可能有“微观通胀管理”。你很不爽这个自大的碧池，那么你打算拿什么驳倒她？

4. 同样是Unsecured Borrowing，为什么LIBOR o/n和Fed Funds Rate之间存在可观的Spread？作为一个Macro Trader，你认为LIBOR和FFR之间有套利机制吗？如果有，你打算如何做这种套利？

5. 为什么金融危机期间Eurodollar的流动性会非常紧，逼的联储大量投放Swap，但Onshore流动性并未紧到迫使联储直接投放流动性的地步（QE是2009年才做的）？那么可观的Onshore/Offshore Spread为什么无法吸引商业银行从Onshore向Eurodollar搬运USD套利？

6. 为什么离岸的LIBOR对在岸的美元Swap、公司债等资产有基准利率意义？在岸美元也有Prime Rate，为什么重要性无法超过离岸的LIBOR？

7. 一些研究表明EM的资本管制措施，比如资本流动税，往往在面临资本流入压力的时候比较有效，在资本流出压力时无效，结合Monetary Economics解释这个现象。

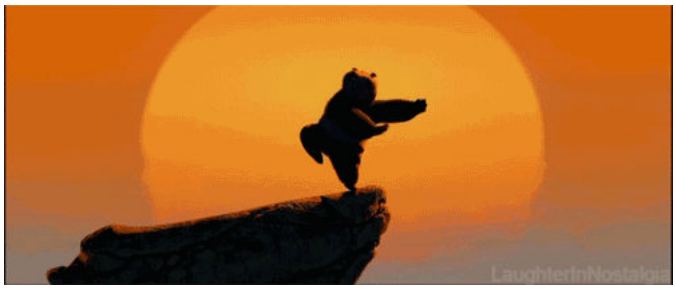
8. 数字货币的狂热粉丝们试图游说政府彻底以数字货币取代现金，实现John Cochrane的《没有现金的世界》（A world without cash http://johnhcochrane.blogspot.com/2016/08/a-world-without-cash.html ）里描绘的美好前景。商业银行强力反击，声称数字化货币将是另一种形式的金本位，按照Monetary Economics篇中的描述，突如其来的流动性紧缩将时不时威胁经济和货币体系。你同意商业银行的意见吗？如果商业银行说的有道理，数字货币体系有没有办法克服这个缺陷？

9. 判断题。如果认为“否”，需要简短说明理由。
（1）. 中国M2/GDP远高于美国，说明中国经济杠杆率远高于美国。
（2）. 受益于Flight to Quality的避险资产或者是货币银行系统的基础货币，或者是有基础货币职能的资产如国债、黄金等。
（3）. 国际化货币必须伴随稳定的经常项目逆差。

好的，习题部分到此结束！接下来进入今天的重头！祝阅读愉快！

宏观交易员的自我修养之三
宏观交易入门

Macroeconomics for T raders
宏观交易入门



如果读者读完Monetary Economics篇后，可以接受Macro Trader是纯粹的Flow Animal这个概念，那么接下来我们就可以为Macroeconomics For Traders给出一个很市侩的视角了：Macroeconomics 对Macro Trader的重要性体现为，许多宏观经济的变化将导致Flow Chart 出现变化。如果这种变化在近期出现的可能性很大，或者有可能改变Flow Chart的结构，那么对Macro Trader就是至关重要的。不过同样市侩的视角也告诉我们：更多、更深刻的宏观经济现象，如果对Flow Chart 没有什么像样的影响，那么就是Macro Trader可以忽略的。

Macroeconomics博大而且庞杂，我不是经济学科班出身但还有点自知之明，不敢胡乱bibi. 所以本篇主要的目的是阐述Macro Trader们将Macro Research 的结论应用于交易策略时常见的一些误区。这样协助Macro Trader多搞些市侩，少玩点情怀。这个方面，更系统一些的阐述可参阅《熊猫笔记20151230：岁末鸡汤之Fundamental》（小编注：该阅读材料将补充在正文后～）。（考虑到本文定位为Macro Trading 101，《岁末鸡汤》似乎可以命名为Macro Trading 202）

1.Time Horizon of Macro Scenarios

宏观情景推演必须有一个Time Horizon，明确这是短期、中期还是长期的因素。很多混乱也是出在Time Horizon层面上。这些混乱中，最常见也是最具有危害性的不是把宏观情景本身的Time Horizon弄错，比如长期弄成中期，而是设计交易策略时无法为正确短期/中期/长期情景找到一个正确的切入点。这个方面，我给Macro Trader的建议是，Fight in the short term, Live in the mid term, Forget about the long term.

虽然宏观经济争论的题目很多，包括人口结构、基础设施和经济转型等许许多多重要的题目，但是Macro Trader作为一个市侩，应该最关注哪些最能影响市场信心的短期信号。典型的数字例如PMI，这个数字对Macroeconomist其实是个很无聊的数字，既不能像Retail Sales/Durable Goods Orders等数字那样协助你解读经济结构，又不能像Savings/Hourly Earnings等数字那样拿来形成输入或者价格参数，拿PMI做个像样的Macro模型都很寒碜。但是由于PMI是一系列市场参与者的信心体现，而这些参与者的信心，直接影响到信贷需求、风险偏好、价格预期等对Macro Trader极其重要的数据，所以PMI是几乎所有Macro Trader都密切关注的数字，尤其是变幻莫测的EM市场。

对Mid term，Macro Trader们需要考虑到Short Term不可能偏离Mid Term太远，而且Mid Term显示的一系列图景随时可能进入Short Term，所以Macro Trader们也需要注意切不能偏离Short T erm图景太远。比如次贷危机之前，从2006年开始，很多美国国债Trader就在讨论美国经济和房地产市场的问题，其

中一个很强悍的论断是美国经济中消费占70%的驱动，考虑到美国消费基本全靠信贷支持，联储加息将导致消费者不堪重负。这个论断极其强悍，但是如果按照这个论调在2006年布局Long Treasury/Curve Steepen/Short Credit等头寸，多半熬不到2007年年中就会止损出局。这是因为短期层面上信贷仍然在源源不断的流入房地产市场，足以支撑这个游戏。所以如果Short Term与Mid Term打起来，Macro Trader是一定要站在Short Term这边的。但是，一旦房地产新增信贷衰竭，Mid Term情景很快就转化为Short Term，这时如果Macro Trader们对Mid Term情景已经推演的很透彻，就能够抢在其他还懵逼着的玩家之前迅速入场，占尽先机坐地收钱。

显然，能够进入Long Term讨论范围的肯定是极为重大的题目，例如人口结构和经济转型等，但是作为一个市侩的Macro Trader，我不希望在交易策略中出现这些题目。如果市场突然热炒某些Long Term的主题，我即使鼓励Macro Trader跟风，也必须要求他们注意区分这是短期投机情绪出现的变化，还是真的长期结构性变化一夜降临了。典型的例子在2007年年初，经历了好几年无聊透顶的高增长低波动之后，一些Macro Analyst开始讨论美国的人口结构和负债，并且预言美国债务增长的不可持续性将导致美国国债利率上扬和美元丧失国际地位，我当时还很感兴趣，但拿到Paper一看，Time Horizon都是十几二十年。姑且不论这个判断是否正确，即使二十年之后美国国债利率真的上行到了10%以上，美元如同手纸一般，这二十年里也可以发生很多意外的事情，比如2008年的全球金融危机，这些事情足以把看空美债的Trader赶出局。如果有Trader仅仅因为不喜欢美国的负债结构就做空美债，愿市场保佑他还活着。

对Long Term主题的矛盾态度，还有Mid Term主题的骑墙态度，都源于Trader的一个职业习惯，那就是Relative Value的收敛性。我们一直强调是市场的不均衡或者失效才给了Trader上下其手追求高收益的机会，那么一个很自然的推论就是，比起捕捉Relative Value，更重要的是考虑Relative Value将如何收敛。如果Relative Value有极大可能被市场资金流推动迅速收敛，就是值得做的Trade，大部分Arbitrage都是这个类型。但是如果Relative Value本身没有收敛机制，或者说市场上只有你一个人孤零零的做这个Trade，是非常危险的，这意味着Relative Value可能偏离均衡水平相当长的时间，而每一次偏离，都意味着你头寸的Mark to Market损失。如果正赶上投资者赎回或者被Call Margin，你就不得不出血斩仓。最典型的例子例如AH股套利，这种教科书一样的套利机会，由于资本管制卡住了收敛机制，资产价格上的不均衡成为常态，就只能停留在纸面上。

所以，回到Short/Mid/Long的话题上，短期信号一般会催生出很强的Flow，如果博的准，Relative Value可以很快收敛，策略能够迅速兑现，这不仅意味着超高的Annualized Return 也意味着风险很小。而中长期只能依靠市场逐步消化，如果市场天然的波动性太多，基于中长期做Trade可能会出现超低的Information Ratio 。考虑到越来越多的Risk Manager和Investor对Information Ratio的热爱，市侩的Macro Trader们不得不放下对重大长期宏观主题的情怀，认认真真的做一个投机者。

2.Macro Stories

明确Time Horizon之后可以开始玩数据了吗？还不能，需要切实理解数据背后的故事，尤其是做FX/Commodity/Cross Asset等需要比较两个经济体的Macro Trade。需要注意每个经济体都有不同的故事，即使是同样的数字，对每个经济体的意义都是不一样的。初学者经常犯的错误是把自己熟悉的宏观故事轻易外推。比如房地产市场，大家都知道房价上升和销量大增对经济是好事，但是房地产对Flow Chart的传导在中美两个经济体就完全不一样。中国经济是投资带动模式，所以房地产的火爆有利于进一步增加基建和房地产投资，从而带动大宗商品尤其是黑色大宗的需求，同时在居民房贷和房地产相关行业产生信贷的旺盛需求。但美国经济是消费模式，房价上涨，更直接的意义在于居民可以拿房产做Cash Out Refinance（2015年2季度，34%的Mortgage Refinance是Cash Out Refinance）。居民拿到的钱可以用来消费，所以美式地产热对消费有很好的提振。这两个模式迥然不同，不可生搬硬套，比如美式地产热如果正碰上油价飙升，对消费的提振作用要大打折扣。而中式房地产热？别为难我，我也不知道这头怪兽啥德性...

这些问题，不仅初学者会犯，大牌人士有时也出于Marketing等原因，有意无意放一些似是而非的说法出来。这些说法因为有大牌人士开光，流毒就更甚。比如BRICS，我一直认为这是近年来为害最广的概念。BRICS是五个形态迥异的经济体，卖油的、卖农产品的、卖矿的、经常项目常年赤字全靠资本项目支撑BoP的...而中国除了工资和人口结构不太像发达国家之外，从产业链上其实是一个制造业超级大国，尽管自己赚取的增加值不高。如果要分析经济增长，把这风马牛不相及五家捏到一起能捏出啥来？所以我看基本面的时候必须一家家分开看，如果时间不够，至少要分成“资源型EM”、“初产品EM”和“中国”来看。这一点应该说也是受B家的影响。他家报告很少把EM囫囵端上来，篇幅再省，也要分“EM ex China”和"China".

所以有一个很有意思的现象，早期有一些做Global Macro，尤其是做EM和Commodity的交易员，会雇佣一些记者来做Research。这些熟悉第一线的记者对Macro Story和Information的解读极有价值。现在研究手段已经极大丰富，Macro Trader可以省下一些实地调研的功夫。但是在《Monetary Economics》篇我们提到，如果做的太舒服，可能形成一个黑盒子，导致Trader丧失对市场基础结构的感知，最终会影响Trader的直觉。所以，一个讲究自我修养的Macro Trader，除了读报告听Conference Call ，还是应该时不时跑跑第一线，锻炼自己的感知。

对Macro Story 的直觉，不是任何模型和数据可以取代的。需要说明一下，Macro Trading用到的Macro模型和方法，其实都很简单，简单到最能呼风唤雨的Macro Trader中没几个Ph.D in Macroeconomics. 决定Trader水平的是对Flow的敏锐感知，这一点必须建筑在观察力上。每次听到“好的经济学家是数学家，糟的经济学家是社会学家”这个说法，我总是心里揣揣。因为对Macro Trader，如果能够像社会学家一样观察，而且能把相关的主要Macro Story分析透彻，再做好相应的推演，就已经很强大了，如果还要做模型来推算，我总是担心会不会走得太远。

3.Macro Model & Data

现在终于可以动手玩数据了。《熊猫笔记20151230：岁末鸡汤之Fundamental》（小编注：该阅读材料将补充在正文后～）曾谈过如何分析Macro数据，所以不打算重复了。只谈在《熊猫笔记20151230：岁末鸡汤之Fundamental》里面没有提到的两个话题。

1. 预测？解读?

如果Macro Trader把自己定义为Flow Animal，重点在于推演宏观经济对Flow Chart的影响，那么有一个很自然的结论是，分析当前的Macro/Flow 作用机制比预测Macro 数字更重要。我们当然很敬仰对增长、通胀等宏观经济运行轨迹预测准确的Analyst，但是如果这个预测结果，对分析Flow Chart帮不上忙，那相当于给Tesla加汽油，并没有什么卵用。如何确保宏观预测的结果是有用的？可能答案不在预测，而在解读。最好的宏观分析师，应该有能力根据宏观经济数据拆解Flow Chart，这本身就是推演Flow Chart的一部分工作。如果这个解读工作足够纯熟，宏观分析始终围绕这Flow Chart这个令人迷醉的题目，那么任何一个Macro发现，几乎立即就能在Flow Chart上找到应用。如果能够做到Macro/Flow浑然一体，甚至不需要在预测方面多花功夫，因为Macro无法跳出Flow，而且Macro Trader可以努力做到即使未能预知Macro方面出现的新变化，也总能够依靠对Flow Chart的把握，抢在懵懵懂懂的其他玩家之前入场。这个方面，再度推荐偶像B家，可以读他们对杠杆周期的阐述，算是管中窥豹。

2. 模型

在《熊猫笔记20151230：岁末鸡汤之Fundamental》里花了很多篇幅谈论模型，主要是两个原因。其一，宏观本身结构超级复杂，而结果又超级简单。不借助模型，很难解构宏观基本面状况。其二，模型本身，是一个很明确的沟通语言，有助于压制随时可能混入讨论的噪音。比如正在和一个Macro Analyst讨论增长的时候突然被另一个Macro Analyst插入一个油价的观点，就很容易打乱整个讨论。如果把油价押到它该去的维度，不管是生产\就业\个人消费\销售与订单，都有利于整盘讨论有建设性的进行。我甚至认为，如果一个宏观分析师完全依赖个人的直觉，即使Track Record再好我也不敢用他的分析押注。但如果我了解宏观分析师的模型和数据，而且能踏实的说他的工作我都知道怎么做只是因为没有时间精力我才直接问他要结论，这样我才敢信赖他的研究结果。

4.补充阅读：岁末鸡汤之Fundamental

我习惯于把经济活动的总体状况称作Fundamental。需要指出的是针对不同的投资策略或者市场，Fundamental 侧重的方向可能有不同。例如国债或者外汇市场，毫无疑问经济增长和通胀就是Fundamental。如果是HY市场，那么行业和公司的财务状况显然要比宏观经济更重要。本篇主要谈我个人在Fundamental研究，主要是宏观经济研究方面，得到的一些教训。

一、Fundamental 的定位

在FICC的领域，Fundamental的重要性如何强调都不为过，因为我们遵循的是Top Down Analysis。几个方面，第一，Fundamental 可能如前所述不能决定策略的Risk/Return ，但是可以判断策略大方向是否存在问题。我不反对做一些逆Fundamental的策略，但是需要注意这个策略应该是基于Valuation或者Technical出现的结构性机会，而且必须注意头寸控制和Time Horizon。第二，Fundamental 往往顺着货币周期催生出很多的交易机会。有一些不看Fundamental的策略，如一些StatArb或者Relative Value策略，在07年以前做的很好。但是金融危机来了都一塌糊涂，这是因为07年以前的信贷扩张和流动性泛滥才是这些策略生存的土壤。金融危机期间的流动性收缩轻松抹掉了这些策略的生存空间。在此特别推荐一下《The Crisis of Crowding》(中译本《从众危机》)，对这些教训的反思极为深刻。顺便说一句，研究宏观但不落实到Money and Credit的都是空炮。第三，主流市场参与者大部分做基于Fundamental的策略，这意味着如果Fundamental 做的好，亏钱的可能性不大。

但是Fundamental 不能通吃一切！不能通吃一切！不能通吃一切！重要的事情说三遍。再度强调，一定要全面考虑所有市场因素，Fundamental 对市场不是决定性作用。我见过很多勤奋的宏观研究员，想按照Fundamental做交易。但如前所述，市场同时受很多非Fundamental因素的影响，可能以很长的时间极大幅度偏离Fundamental。如果你的Book对波动容忍度不高，很容易被拖入完全没有预防的情景。更要命的是如果少考虑了Valuation/Technical等重要的风险因素，将无法建立很好的Risk/Return判断，这对你的投资决策以及执行是一个严重干扰。所以，除非你在做完全不需要考虑市场波动，只需要考虑结构性Premium的产品，例如银行的成本法投资户，或者极为复杂以至于只能用模型估值而且有高Carry保护的结构化Notes，都必须知道Fundamental不是你的全部。还是一句老话，市场价格不是被冥冥之中的手点出来的，而是无数参与者扭打撕咬达成的脆弱均衡。Fundamental隐身在两边默默的上下其手，影响多空力量对比，但是从不直接参战。

二、Fundamental 方法总论

考虑到宏观经济是大家最熟悉的Fundamental，接下来主要用宏观经济研究来阐述一下Fundamental研究的方法论。

首先说教训。我总结我自己犯过的错误，以及观察到其他人走入的歧途，认为宏观经济研究方面，常见的错误如下：

1.数字的奴隶

由于缺乏一个扎实客观的全局性宏观认识而完全被经济数字的跳动所左右，出来与与自己感觉不符的经济数字后或者忙不迭改观点，或者坚持观点但是只能以心虚或盲目的态度坚持。须知即使在经济最差的时候，也从来不缺正面的数字（提示：可以看一下Chicago Fed National Activity Index: Diffusion Index，看看这个指数在+-0.8以外的极端时间有多少）

2.数字复读机

这种情况多见于卖方研究，以纯熟的套路和方法，复述经济数字，但是不花功夫对数字背后的宏观情况进行研判，或者分析归分析、研判归研判。别人问一个数字，可以解释的很清楚，比如将零售销售的数据分解开来详细分析，说不定还做一点扩展分析，做一些数据处理。看起来似乎做了很多工作，其实只是对数据的解释，而不是研究，无益于投资决策。卖方做这样的事是他们天然职责所在，因为卖方需要对不懂的人解释，但如果你已经脱离了“不懂”的层面，急切的想要寻求交易机会，那就不应该满足于仅仅复述这些数据。你应该配合投资决策需求开展宏观研究，而不是做一个功能强大的复读机。编个程序都能做的比数字复读机好。再扯一句，我以前还写过类似的写报告程序，糊弄领导方面真是又快又好。

3.数据挖掘

统计、建模和数学很强，但是缺乏实地经验的宏观分析员，有时候可能陷入数据挖掘的误区，不是通过分析数字背后的脉络，而是盲目应用数据挖掘技巧，试图通过复杂的模型和数字进行预测。我特别反感数据挖掘，是因为这种做法很容易做出黑盒子模型，看似非常漂亮，但是使用者无法借助这个黑盒子理解经济的脉络。对统计方法和建模的应用，必须建立在对经济脉络的熟练感知上。建议借助在Zhihu的一篇很火的文章《美联储怎样预测宏观经济》，感受一下建模能力和经济学直觉要到了什么样的水平，才能有效的运用复杂的数学模型来预测经济。

为了避免以上的问题，读Fundamental ，我的习惯是按照PM读Attribution（ 归因分析)的方法读。顺便说一句，我考察债券PM，第一件事不是看历史业绩数字，而是看历史业绩的归因分析。必须知道哪些收益来自于博Duration/OverweightSpread/Leveraging/Selling Vol，我才能判断这个PM的强项在哪里，在什么样的市场里他的Risk/Return Profile最好。再结合我对市场的判断，决定给他的资金配置以及风险管理要求。读宏观我也习惯于按照Contribution/Attribution的方法读。例如如果读美国经济，需要意识到Demand上升，将会对经济带来持久的带动，Housing是最容易受利率水平影响的因素，利率如果受非美国经济的因素影响（例如欧债危机），可能在Housing领域观察到明显迹象但其他领域则反应平平，Consuming则是油价敏感，若油价大幅下行Consuming将有意外之喜，但是带动效应和可持续性不强。读到手的，不是几个数字的堆砌，而是一个很完整的经济故事。

所以读基本面应该有一个时效性好、全面、客观的量化Model，再围绕这个Model进行诠释或者补充模型之外的因素。注意我没有把“准确”当成一项对Model的要求，因为我习惯与把基本面当成大方向来读，只要Approximately Right就行。在我看来，如果Model很准（尤其是一些预测型的Model），但是时滞太长或者解读太繁琐，也不利于在第一时间寻找确定性的机会。接下来还会以GDP为例，说明对基本面量化Model的看法。

某些产品或者市场，例如HY，受的影响因素非常多，而且往往很不稳定，符合上述标准的模型很难找。一些人士喜欢用“打分卡”的方法建立基本面研究体系。这个方法我不反对，一个个思考打分因子的过程，自然就是分析Contribution的过程，协助你全面考虑整个体系。但我不关心打分卡最后加总起来是什么总分，而是更关心构建打分卡时各个因素是否已经考虑充分。所以Checklist有三个建议，一是不要把太多因子放到打分卡里面，打分卡是协助理解，做太复杂就不好理解了。要有能力分清主次。二就是打分卡结构要简洁可读，超过三层以上的打分卡就很难辨别原意了。三是不要太在意打分卡给每个因子的权重，而是要把心思花在分析各个打分因子上，要清楚目前最关心哪些因子。最后，不要把投资决策的责任用打分卡来机械的完成。人的综合权衡，而不是打分卡，才是投资决策的依据。

不论是量化Model，还是打分卡方法，我的要求都是按Contribution/Attribution　方法，先从分解数字开始，对每个因子形成具体实在的认识，最后综合形成一个完整的经济故事。我称之为Quantitate Data->Qualitative Stories-> Quantitative Assessment。Checklist要求能通过这个方法，回答以下问题:

1.如何看待目前的状态?
2.各个维度的因子在达到目前的状态中起了什么样的作用?
3.按照对各个维度因子的判断，基本面下一步的Base Scenario是什么？最可能的Adverse Scenario可能会是什么状况?
4.数据出现什么样的变化会验证/影响你的Base Scenario/Adverse Scenario判断？考虑到你对自己往往持有过剩的信心，数据如果出现什么样的变化，你将不得不承认自己错了?我可没有空等你找一个台阶下。

按照这种方法读宏观，似乎GDP以及其分项就是一个很好的经济学模型。但GDP存在固有的问题，例如时滞太长、而且分析GDP的方法很难吸收一些重要的先行数字，因此对时刻准备做投资决策的PM/Trader帮助不大。在这个方面我最推崇的模型是Chicago Fed National Activity Index。这个指数系列的介绍参见
https://www.chicagofed.org/publications/cfnai/index. 指数采用的方法来自于James Stock & Mark Watson在1999年发表在Journal of Monetary Economics上的一篇文章“Forecasting Inflation”，之后这套方法在主流经济研究机构流行开来。除了Chicago Fed National Activity Index，还有OECD Leading Indicators，Goldman Sachs的Current Activity Indicator等。我推崇这套方法，并不是出于其预测准确性，虽然很多时候这套方法的确能显示出很不错的准确度（最新的例子可见Federal Reserve Bank of Atlanta的GDPNow模型对美国经济的预测），而是Chicago Fed National Activity Index构建之始就有一个很好的分项，将经济数字分为生产/就业/个人消费/销售与订单等四个维度，全套模型很完整的体现了四个维度的Contribution/Attribution。从这个角度出发，我甚至认为Goldman Sachs的Current Activity Indicator等方法只注意预测准确性而忽视分项的构建和解读，把这套方法当成一个数据挖掘的工具，放弃了最重要的部分，实在是买椟还珠。

三、因子分析方法论

其实按照前述观点，宏观研究中维度或者因子分析，例如分析就业/销售/房地产等因子，其重要性大大高于预测整体数字。此外，还有一些专门因子，虽然不一定被纳入整体模型，但是对市场的影响极大（例如CPI等），这些因子也是需要花功夫琢磨的。因子研究难度比整体研究要高。整体预测模型还可以

借助数学或统计方法抓出一个囫囵数字，但因子分析就是扎扎实实考研究员的研究直觉、经济学功底和分析能力了。在因子研究这个方面，我的观点和方法也一直在变化，在20151019的鸡汤里就提到过“普通”“文艺”“2B”的方式。我一度极为痴迷于“文艺”方法，现在更倾向于升级版的“普通”方法。

因子研究最常见的方法是分析在统计部门在这些领域报告的数字，以对这些数字的解读确定观点。Checklist要求研究员能够把这个方法掌握的很纯熟，因为这些统计数字是研究的材料，对数字的掌握是研究员的基本功。但是在20151019的鸡汤里，我把这个方法是归入“普通研究员”。想进入“文艺研究员”的境界，需要在这个方法之上继续提高，因为这个方法有其固有的问题。

1.统计部门的工作习惯、统计方法和使用习惯

研究员是经济数字的用户，但这个用户对生产者，也就是统计部门，没有影响力。统计部门有自己的工作倾向和工作习惯，不会因为适应研究员的需求而改变。甚至统计部门的一些考虑，例如严谨性等，会影响其经济数字的经济意义。在20151016的段子里举过一个例子，因为过于看重统计方法的严谨性，导致外汇局的外债统计数字在研究负债结构时经济意义不如BIS的数字。因此在使用某一个经济数字之前，研究员首先需要理清自己的需求，从这个具体需求出发使用这个数字。作为使用者，我认为研究员除了要理解每个数字的来源、方式等，还需要具备一定的深度研究能力。一是数据再处理能力。这个方面最常见的例子是调整非经济活动因素对经济数字的影响，例如按日历调整工业数字以消除春节因素的影响，分析天气对Payroll的影响等。二是根据自己的需求发掘新的统计数字。我在做MBS的时候，UBS有一个特别的MBS研究员以Google Search关键字出现频率判断美国居民再融资需求。另一个例子是在中国市场，有所谓“方便面指数”等。这个方面我不建议研究员走火入魔的追求新数字，但是至少应该对新方法的价值有一个评判。三是根据自己的需求，用已有的数字编制新指数的能力。我在这里举了一个我最欣赏的机构，B家的例子，是居民住房投资的数字。市场上没有一个统计机构统计住房投资数字，但是B家研究中需要这个数字，他们就用其他的统计数字再加工提炼，做成一个符合自己需求的，具有明显经济意义的数字。类似的模型和数字在B家的研究报告有上千个，构成了B家独树一帜的研究方法，保证其宏观研究的深度。

坦白说，这些能力都或多或少属于“文艺”型，对研究员的要求不低。

2.研究员的工作倾向

说一个本人经历过的例子。某次一家大行的著名Mortgage研究员发布了一篇报告，认为一个权威数字的变化，显示某类型机构的近期有很大的资产配置转换。其实这些变化是因为我的一些融资交易形成的，与所谓资产配置转换完全风马牛不相及。正好这家大行的交易员帮我安排了这些融资交易，所以我很好奇的问他，你知道这些数字的变化是怎么回事，为什么不提醒你们的研究员。交易员的回答说，你不懂研究员的苦。第一，为了保证研究工作的严谨，他无法采用一些非公开渠道得知的消息，知道的再多也没法写上去。第二，一些过于深刻的报告可能引起意外的后果，影响银行与客户的关系，或者银行的声誉，搞不好工作就丢了。为了保护自己，比较稳妥的办法是追随主流，同时适当的显示自己的亮点。因此硬着头皮写一份风险不大但是意义也不大的口水报告，是很常见的事情。

这一点常见于卖方研究员，但需要指出的是，买方研究员出现这种倾向的也不少。毕竟买方研究员的研究不直接产生P&L，而是需要推给PM/Trader。如果风格与PM/Trader合不上，会徒劳无功。在这种情况下，买方研究员也会有压力把重心不是放在发展自己的“文艺”体系，而是按照口水报告的风格做研究。因为如果研究员想换工作，曲高和寡的“文艺”方法反而比大家耳熟能详的口水体系转换成本要高。

这是对“普通”型研究的批评。但“文艺”型研究也有其死穴，在20151019的段子里有提到。首先是巨大的投入，成本相当高，周期相当长，甚至对管理都是很高的要求，因为“文艺”型研究需要大量的资源和团队协作，一个人绝对做不出来，甚至不是实力极其雄厚的机构也负担不起。再就是曲高和寡，这反而导致文艺研究员换工作时面临劣势。甚至文艺型PM/Trader都会面临类似的问题，因为新机构可能没有他习惯的研究资源和研究支持。考虑到这些原因，我后来更多的建议采取“抬扛”方法，试图用好“普通”型研究资源。这个方法就是专门找卖方研究员抬扛。对PM/Trader 尤其是Flow Trader来说，一忌讳看的东西不够全面，二忌讳对主流观点理解不透彻（注意：可以偏离主流观点，但偏离的越远越需要对主流观点的全面透彻理解）。对现有的卖方研究结果做严苛的审查，本身就是一个弥补自己片面性的过程，但是这个过程需要以鸡蛋里挑骨头的抬杠方式，而不是锦上添花的合作补充方式进行。而且，需要找尽可能多的卖方。对手卖方不一定需要是文艺型、曲高和寡的研究员，最好是在普通型PM/Trader中受欢迎的普通研究员，这样抬一圈杠下来，对主流观点的倾向和缺陷都心中有数了，足以迎合其倾向而防备其缺陷。这是一个投入较小而效率相对较高的办法，算是把普通型研究用出文艺范的思路。

不论是“普通”型方法，还是“文艺”型方法，其目的都不是预测某一个因子或者是赌某一个因子，而是分析这个因子的变化是否会引起某些结构性不均衡，或者成为结构性转换的先行迹象。最好赚的钱不是来自于赌方向，而是结构性的转换带来的风险小而收益确定的机会。Checklist要求问研究员这样几个问题：

1.在本期重要数据出来前，你有没有根据其他数字或者模型形成自己的一个预测或者Base Scenario，这个预测方法是稳定的还是动辄更改，是否定期回顾这些方法，是否保存预测结果检验和预测方法修正的记录?

2.做好预测后，判断经济处在周期的哪一段。在什么样的经济环境下、目前的数字看起来是比较合理的；一般来说，像这样的情景或者数字，会持续多久，然后在某个情况下出现反弹或者扭转。

3.在从Bottom回溯Top过程中，这些因子是如何影响你对整体图景的判断的。你的整体图景判断对哪一类重要数据最敏感，或者说哪一类的数据变化与你的Base Scenario相差较大时最有可能证实或者证伪你的观点。现在还有10分钟出数据，如果出现Surprise，在什么情况下，你会坚持原始观点，而在什么情况下，你会修正原始观点。

四、政策研究的补充

政策研究不是我的强项，Checklist只提出了一个方法。金融危机的一个巨大副作用，就是政策制定者们从原有的监管角色越来越多的进入直接市场操作，例如QE。一个直接的后果是很多依赖市场机制Converge的模型不再有效，PM/Trader不得不进入直接博政策方向的角度，使得政策研究的重要性大大提高。

这些政策都是有极大争议性的，而且我一直有一个感觉，可能政策制定者自己内部也非常纠结，而且做出会很多其实很愚蠢的政策。所以我的第一个建议是作为一个PM，千万不能表现出国师的态度，觉得自己比政策制定者高明，然后去根据自己的高明政策建议做投资决策。更多时候需要考虑，如果我处在政策制定人的角色，我所受的制约是什么，如何降低我而不是市场的风险。或者说，要有能力理解愚蠢的政策。第二是要避免在一个角度上下太多功夫，例如只研究流动性投放政策而忽视财政支持政策。要注意研究的全面性，尤其要注意，如果某一个政策可能在市场上造成极大的不均衡，这种不均衡显示出什么样的机会和风险。

因为政策不一定是很正确，反而很可能是愚蠢的，所以一个人天然是很难把政策推演的非常全面和透彻的。Checklist的一个建议是找两组人来，各扮演正反方，让大家去辩论。辩论的受益人不是两组选手，而是围观群众。如果两组人能够把每一派的动机和倾向阐述的比较透彻，围观群众就能有一个很好的机制全面的理解政策。同样，这个方法的目的不是去预测政策可能出在什么地方，更多的是通过这种方法理解政策制定的动机以及可能出现的后续。这样你才能在以后政策变动时有足够的手段保护自己。

Checklist是2012年写的，写得完全是应对外国政策的东西，不知道接下来这套方法会不会用到我们自己的经济分析上。

5.结束语

Fiat Money体系下，央行和金融机构的信贷创造和流动性管理存在着若干先天的缺陷。观察这些缺陷，推演资金流将如何被这些缺陷扭曲再Front Run这部分资金，是Macro Trader的饭碗。在写作本文的时候，我正好读到宏观金融学巨擘John Cochrane的一篇Blog，《没有现金的世界》，描绘了一个通过数字货币和区块链搭建的货币体系，这个体系是如此完美，乃至金融体系天生就稳若磐石，央行轻敲键盘立即化通胀于无形，流动性召之即来挥之即去。A world too perfect for monetary economists.　虽然这个图景看起来色彩实在太未来，估计远在若干世纪以后，但是我们似乎已经看到那时Macro Trader作为一个职业的终结。

参考资料
1. 熊猫笔记，20151230，20160606
2. Ray Dalio, How the Economic Machine Works, http://www.economicprinciples.org/
3. Federal Reserve Bank of Chicago, Chicago Fed National Activity Index, https://www.chicagofed.org/publications/cfnai/index
4. James H Stock, Mark W Watson, Forecasting inflation , Journal of Monetary Economics, 1998

宏观交易101第四课丨Cash Study： 货币基金改革如何隔山打牛

相信经过前面三节课（戳蓝字查看前三课：一、货币银行体系的资金和信贷，二、跨境资金流动，三、宏观交易入门）的捶打，大家都对宏观交易有了进一步的了解。

证券基金系统的改革，证券基金市场表示情绪稳定，利率市场鸭梨山大...

Onshore 利率市场的变化，Onshore 玩家表示情绪稳定，Offshore 市场鸭梨山大...

日本银行业美元流动性的紧张，日本银行业表示积极解决，美国国债市场鸭梨山大...

鸭梨山大怎么办？且看“宏观交易员的自我修养”之四：《Cash Study： 货币基金改革如何隔山打牛》！

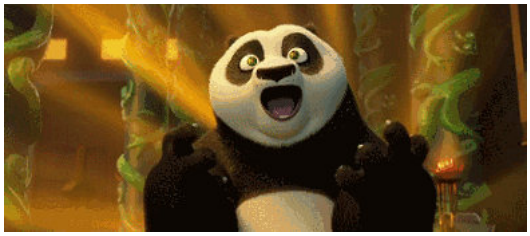
宏观交易员的自我修养之四
Cash Study

货币基金改革如何隔山打牛

文/熊猫交易员

熊猫笔记

Case Study
货币基金改革如何
隔山打牛



《Monetary Economics》篇多次提到，如果Flow Chart的结构发生变化，遵循原有Flow Chart流动的资金将不得不改道。如果受影响的资金很多，这批资金将搅动着洪荒之力，透过Flow Chart里那些最脆弱的环节，裹胁大群无辜吃瓜群众，在整个市场呼啸。市场里各式玩家的来来去去看似波澜壮阔，其实也就是海面上的风浪。而Flow Chart的变化，相当于海啸，事先没有征兆但突然就排山倒海搅动天地。2016第三季度由于美国货币基金2a-7改革而引发的一系列变动，是一个绝佳的例子，说明货币传导机制里一个不起眼的变化，由于发生在最脆弱的环节，其传导和连带效应是如何让整个市场都受到意外影响的。

1.货币基金：不是银行，胜似银行

货币基金的管理极其无趣，然而货币基金的独特机制和独享特权，让每一个债券基金经理都眼红。《Monetary Economics》篇提到，判断是否有信贷创造，是依据Dealer是否能仅以少量的基础资产就能维持大量的偿付请求。最典型的例子是银行，只需要一点流动性准备，就能向所有的存款客户承诺兑付。而证券投资基金，虽然客户也能够赎回并获得现金，但是基金管理人不能承诺兑付，只能规规矩矩按照净值支付给投资者，投资者需要自行承担投资组合波动的风险。所以证券投资基金不是信贷机制，只是一个通道机制。

但是固定面值(CNAV, Constant NA V)的货币基金是承诺足额兑付的。虽然这些CNAV基金投资组合里并不全是现金或者国债等流动性和安全性最高的基础货币类资产，而是有大量CP/CD等在Flow Chart中扩张出的信贷派生资产，严格来说是不能足额提前兑付的，但是货币基金仍然拍着胸脯承诺足额面值兑付。而且，货币基金承诺的流动性是全额T+1甚至T+0（各国货币基金可能会有不同，但都是仅次于即时）。这比银行定存都厉害，跟银行的储蓄账户差不多了。所以，在美国的货币统计体系里，Retail Money Market Fund是被列入M2统计的。

这么牛的产品，自然必须有很多特权。最眼红的特权是CNAV基金可以用摊余成本法记账，货币基金经理基本上不用在乎投资组合净值波动，只需要算算每天的利息收入，然后分配给投资者就行了。这一点比银行都牛。银行存款也是摊余成本记账，但为了能够使用成本法，银行除了忍受繁杂的贷款风险管理监管，还必须保有相当的一级和二级资本。因为如果贷款出损失，或者投资没有办法兑付收益，银行是需要从资本里掏钱支付给投资者的。这些资本是为了能够让成本法正常运行必须提供的Buffer.，然而货币基金就不需要Buffer ，货币基金经理们表示投资者进进出出很常见，就算赎回导致投资组合已经资不抵债，过一阵子就回来了。所以，在货币基金里，为先赎回的投资者当Buffer的其实是后知后觉的投资者。如果货币基金挤兑，那么投资组合就会越跌越惨，一旦炮灰接济不上，Buffer们就会发现货币基金有赖账的风险。

可想而知，这种特权在监管者眼中多么扎眼。更要命的是，货币基金一旦发生挤兑几乎必然是全行业的挤兑，同时还会阻断大批企业和金融机构的短期融资渠道，后果不堪设想。所以，自从1971年货币基金诞生开始，一代代监管者们都试图与货币基金作斗争。然而，监管者们撼不动成本法，只能在投资风险上多做文章。以美国为例，旨在监管货币基金投资风险的2a-7法则从1983年发布开始，历经多次艰难的修订，限制Duration\Credit Risk\Liquidity等等等等，但还是防不住历史最悠久的Reserve Primary Fund在2008年因为踩中Lehman的坑而不得不跌破净值，先冻结后清盘。一大批投资者眼睁睁看着号称安全性流动性堪比现金的货币基金投资不仅不能兑付，还一路下跌，最惨的时候跌去5%. 为了阻止可能出现的全行业挤兑和和对整个金融市场的冲击，FED迅速宣布Money Market Investor Funding Facility，对货币基金行业提供高达6000亿美元的流动性支持。这个Facility虽然一分钱都没有实际动用，但是被认为是金融危机中最正确的决定之一。

缓过气来后，监管者们痛定思痛，进一步限制CNAV型货币基金的空间。欧洲由于VNAV型基金（Variable NAV，浮动净值型）占据半壁江山，CNAV无法独占话语权，所以改革阻力稍小，进展快一些。而美国SEC则遭遇了CNAV型基金群体的激烈抵抗，最终铁娘子Mary Jo White击败了阻挠者，于2014年7月23日宣布，从2016年10月14日起实施新的2a-7监管规定。虽然铁娘子仍然无法取消CNAV，但是她极大压缩了CNAV的空间。相关细则读者可以自行寻找相关资料阅读，只需要了解一点，有大概1万亿美元的资金，不得不转型为VNAV，也就是由信贷机制转换为通道机制。通道机制的效果是集结零散资金投资，虽然也可以投资，但是效率比起能轻松翻番的信贷机制要差一些。所以对这些VNAV，虽然他们仍然可以投资CP和CD之类的产品，但是胃口将极大受限。

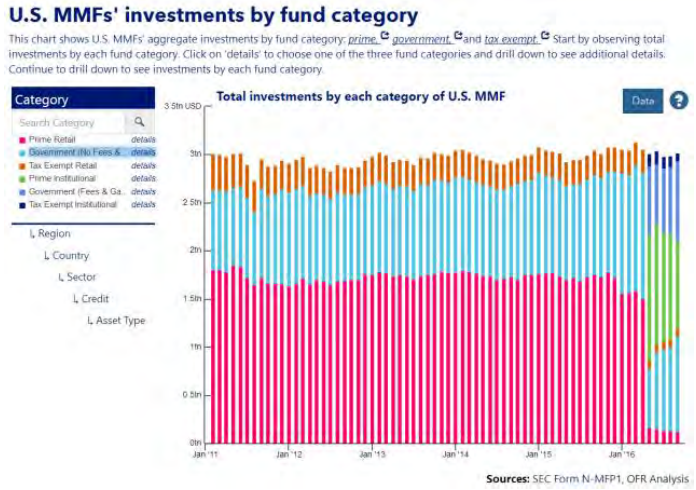
其他改革措施，比如市场上还有很多人在讨论的国债投资期限和流动性准备方面的限制，我并不准备特别考虑，因为《Monetary Economics》篇提到，国债也有基础货币的职能，所以，一个平均期限30天的国债、现金和回购投资组合，和平均期限300天的国债投资组合，对Flow Chart不会有实质性的差异。

考虑到读者范围，简单引用一个研究结论：大约有3000-5000亿美元的资金，以前一直依靠向货币基金发行CP/CD市场融资的，将不得不寻找新的融资路径。毫无疑问，市场融资利率将会升高。但是作为Macro Trader，我们更感兴趣这些资金承载的资产是什么。利率上行和融资利率的升高将压制这些承载资产的吸引力，这时候还持有这些资产可要当心。更重要的是，如果投资者缺乏新融资路径或者难以忍受高昂的融资成本，他们不得不卖掉资产，偿还债务，缩表走人。这时候Macro Trader将有一个绝佳的Front Run这些头寸的机会。

2Onshore 平平淡淡

我们已经意识到，虽然2a-7改革纯粹是一个证券投资基金改革，但这是在Flow Chart 的一个重要节点上动刀，效果相当于收紧流动性。

2a-7改革对证券投资基金业的影响，主要是事务性的，没有太多实质性影响。货币基金投资管理方面，美国基金经理们历经多年磨难已经相当成熟，风险偏好不大。应对2a-7改革的工作，更多是体现在拆分或者成立新基金、修订基金合同、聘请新受托人等等法律事务工作，投资组合调整反而不大。货币基金投资人方面，QE和零利率以后，投资者已经习惯了货币基金收益长期徘徊在几个BPs，而CNAV货币基金方便记账、没有银行交易对手风险等好处对企业CFO和PM们还是挺有吸引力，所以即使2a-7可能降低CNAV基金的收益率，投资者仍然高高兴兴的把钱放到新的CNAV基金里。所以美国货币基金的AUM，其实没有发生太大的变化。结构转换方面，主要的货币基金早在2016年1季度就开始转换，在7月份开始进入转换高峰，10月初基本已完成。整个转换过程应该说是平稳有序的，毕竟整个行业为这次转换已经准备了两年多的时间。



对利率市场的影响，则体现在数千亿美元需要寻求新的融资途径，这样势必会造成融资利率的上行。由于3个月是美国CP市场的主要融资期限，所以进入7月份以后（在10月份截止前发行3个月CP的最后期限），3个月期限LIBOR/OIS Spread一路飙升，接近5年以来的最高点。但是这只是价的变化，并不是Macro Trader最关注的量的变化。量方面，美国银行业还堆积着超过2万亿美元的超额准备金，能够轻松抹平本土市场可能出现的流动性青黄不接。所以本土市场，除了融资利率上升十几个BPs让融资者颇为不爽以外，正常的融资能够继续进行。本土的波澜不惊还体现在美国本土机构的融资利率其实还算稳定，下图亮黄线显示金融机构CP融资利率上行幅度低于LIBOR，绿线显示非金融企业的CP（70%为美国企业）更是几乎没什么变化。（9月底的OIS和CP利率的走高是因为考虑了12月份加息预期）

3.Offshore 好戏连台

《Monetary Economics》篇提到，美元是真正的全球货币，主要金融机构都有大量的美元需求。体现在地域上，就是离岸美元市场极其庞大。仅以CP市场为例，2016年6月底外国金融机构在美国发行了2700亿美元的CP来融资，此外还有体量更庞大的欧洲美元CP市场，2016年6月底规模超过5500亿美元，其中有一半是美元。美国金融机构也喜欢庞大的离岸市场，Prime Money Market Fund的投资中，在美国本土的经常不到一半，大部分投资投在欧洲、亚洲和其他离岸美元市场。

虽然货币基金改革影响全部的离岸美元市场，但欧洲银行业压力比日本银行业小。这是因为欧洲的离岸美元市场发展已经相当成熟，资金来源多样化，玩家多样化，货币体系抗冲击能力强。而日本银行业明显玩的太High了。据估算，日本银行业在美国的分支机构通过CP、CD等市场融入的美元在2016年1季度末超过6000亿美元。毫无疑问，当Prime Money Market Fund不得不关闭CP/CD窗口的时候，日本银行业的重要美元融资渠道被严重压缩了。

外国金融机构用CP/CD市场搞那么多美元拿来做什么？必须搞明白这个问题才能理出Flow Chart中最值得攻击的环节。Credit Suisse的Zoltan Pozsar列举了4项，分别是进行IOER套利、补充监管要求的高流动性资产、在美国本土放贷和调回本国使用。Zoltan Pozsar是一个严肃的Analyst，对几种情况做了深入的分析。但我们是Macro Trader，我们只对最脆弱的环节感兴趣。我们发现，在Flow Chart 扩表出来的资产比如股票、商品等，市场对其波动有极大的容忍度。另外一个有意思的发现是实盘投资者，例如外国投资者以及套牢了就不撒手的散户，市场对他们的P&L波动也表现的很淡漠。相比之下，金融机构随便一个负面新闻，都能让所有玩家的心跳陡然加速。这是因为这些资产或者参与者的Liability很轻，而金融机构Liability很重。特别强调一下，Liability驱动的交易往往是Convergence最强的交易。如果Asset一方出现波动但是Liability相对稳定，玩家往往还有套牢死磕的空间。但是一旦Liability出现问题，比如投资者赎回、银行削减Credit Line、Margin Call等等，玩家将陷入极端被动的境地。所以，熊猫在做PM的时候即使再目中无人，但见到客户，甚至见到自家Sales，也是要客客气气的。

那么Liability 驱动的交易可能出现在哪里？我们希望找到同时具备如下特点的环节。一是Liability 出现大幅收缩的可能性很大，最好是《Monetary Economics》中描述的基础资产比如备付金等出现收缩，或者银行金融机构扩表意愿出现大幅收缩，这样Liability的收缩将以成倍的效果，扩张到Flow Chart各个环节。二是流动性自我纠偏的能力很弱，容易出现羊群效应，这时候我们不喜欢强力的央行，而离岸市场正好没有央行。所以，Zoltan Pozsar列举的四项中，或者是在Flow Chart同一层级腾挪资产，难以形成Liability收缩，或者是有很强力的替代机制，自修复能力强，不容易攻击。唯独日本银行业的美元市场，用Prime Money Market Fund融入的美元起到备付金的作用。我们注意到这个市场的备付金正面临威胁，又是一个缺乏强力流动性提供者的离岸市场，再加上日本银行业的美元市场参与者行为高度同质化，很容易形成一致效应，所以这就是我们最喜欢的地方。最终，我们见到LIBOR/OIS逐渐走高的时候，JPY Basis狂跌不止，接近2011年欧债危机时候的水平，可见日本市场USD Funding紧张到什么状态。BoJ在7、8月份虽然喊话要提供美元流动性，但是嘴炮太多真金白银太少。可怜的日本银行业高兴了两天，回头发现美元还是紧的不行。

至此，读者可以把这个故事理解为“美国货币基金改革关闭日本融资渠道所以日本跪求美元”。这个解释其实已经可以拿来指导策略设计了。但是这个解释跳过了很多地方，真实的情况其实更加复杂，因为日本银行业的美元市场本身就是一个结构完整的离岸货币银行系统，包括实体经济、银行和证券系统，同样蕴含了许多有意思的结构，值得仔细推敲。首先是日本银行业的Funding 结构，一些读者已经指出，日本银行业的美元Funding渠道并非CP/CD一条路，FX Swap的融资量就同样可观，此外还有资本流入等等。BoJ前不久曾发布过一片工作论文Recent Trends in Cross-currency Basis曾讨论到这个题目。我需要说明，CP/CD的融资被压缩，并不是一口气把日本银行业的美元融资砍断了，而是起到一个急剧提高边际成本的作用。其次，美元Liability 压缩不意味着Funding 就必须紧，因为美元在这个货币体系里起到备付金的作用。《Monetary Economics》篇指出，只有在交易量增加又伴随着货币体系传导效率下降或者货币体系收缩时，才会出现显著的流动性收紧。这正是7月份以后日本银行业面临的情况，由于对货币政策和美元成本升高感到懵逼，日本金融机构不得不多囤积美元以应对不确定性。事实上BoJ7月份宣布要提供USD流动性的时候，市场情绪有一定恢复，JPY Basis出现明显的回调，但很快玩家们就发现BoJ只是嘴炮，又开始争先恐后的抢美元。第三个因素是，日本市场的离岸美元货币银行体系确实出现了雪上加霜的传导效率下降。这是个独立事件，起源于金融危机之后美国和欧洲监管机构普遍进一步加强了对银行的监管，因此银行提供流动性的能力越来越差。以CCY Basis为例，美资金融机构每笔CCY Basis都需要计算CVA，把本来就紧张的JPY Basis成本再加高十几个BPs。第四个因素，恰巧也在7月份前后出现的Brexit 和德银等风险事件，也影响了全球金融机构信贷偏好，Flight to Quality必然会导致JPY Basis跌跌不休，USD紧的不行。（关于JPY Basis和Flight to Quality的讨论可以见随后的《Case Study：日元避险货币之宝宝心里苦但是宝宝不说》）

现在我们已经整理出了最脆弱的Liability环节。作为Macro Trader，我们可以选择两个方法来Front Run日本人的钱。第一个方法是在Liability 方向，研判出LIBOR走高的机制后事先多囤一些USD，趁着日本人火烧屁股的时候高价借给日本人，赚高利贷。这当然不是什么高招，所以我们要在Asset方向做更狠的招数，就是研判这些Liability承载的Asset，Liability越高日本人越撑不住，很可能会斩一些Asset平仓出局。抢在日本人之前斩这些Asset，把伤口撕得更大一点，然后利用日本人的斩仓Flow平空头，能获得更丰厚的收益。

Liability 驱动的Asset 策略，往往是利润最丰厚的机会。因为玩家既然要用Liability承载Asset，那么Liability 一般期限短、流动性好、Risk Premium 低，Asset 一般期限长、流动性差、Risk Premium 高。这样才能赚到Asset/Liability 之间的价差。典型的如银行存款期限短而贷款期限长，又比如金融危机之前大批对冲基金发行ABCP投资Subprime甚至CDO。市场波澜不惊时，这些玩家可以赚到利差。但是一旦市场反转，Asset跌起来比Liability猛多了。如果能成功Front Run这些遭遇Squeeze的Asset斩仓盘，Macro Trader们就能赚到前所未有的丰厚利润。

在海外投资的日本金融机构主要是寿险公司和银行。他们都是投资活动受到严格监管的机构，因此主要投资在美国和欧洲的国债、MBS和高等级公司债等市场。寿险公司通常对冲大部分外汇敞口，银行更是几乎100%对冲外汇敞口。因此，扣除对冲成本之后的10年期美国国债实际收益率具有基准意义。进入7月以后，美国国债收益率受Brexit和德银风险事件等影响不断下行，10年期美国国债收益率在7月8日达到1.38的低点。几乎在同时，Prime Money Market Fund改革的效应开始在LIBOR/OIS和JPY Basis市场显现。下图可见2016年7月8日，扣除日元对冲成本之后的10年期美国国债收益率第一次触及0，这意味着美国国债对日本人已经毫无价值。接下来如果美元继续趋紧，日元对冲成本进一步上升，日本人卖美债只是个时间问题。可以推测，大批投机者默默观察着JPY Basis（中间肯定有人落井下石加码Squeeze了一把JPY Basis），同时建好了美债看跌头寸。JGB在8月底9月初由于揣测BoJ态度而出现的收益率上行也成为空头友军。所以这一天并没有太久，9月初日本投资者开始卖美债，很快就把美债收益率推高了20个BPs，将近一块五毛多钱。

这个故事似乎在2016年10月14日正式结束，实际上还要结束的早一些。因为Basel III要求银行把30天以内应付债务计入LCR，所以银行除非太任性，否则不会去发行30天的CP/CD来消耗LCR。考虑到10月14日是最终生效日，银行寻求融资的最晚时间其实是9月14日。因此，这个Trade最佳窗口是9月初。所以，Macro Trader们需要注意，银行的每一个微小细节，都需要放到你的策略中考虑。

戳蓝字，查看

往期“宏观交易员的自我修养”系列：

宏观交易101第一课丨货币银行体系的资金和信贷

宏观交易101第二课丨跨境资金流动

宏观交易101第三课丨宏观交易入门

参考资料
1. Investment Company Institute. “Money Market Fund Resource Center".
2. Office of Financial Research, "U.S. Money Market Fund Monitor"
3. Federal Reserve. "FRB: Commercial Paper Outstanding", Weekly Statistical Release
4. The Collaborative Market Data Network,
5. Credit Suisse Global Research, "Global Money Notes #7: Japanese Banks, LIBOR and the FX Swap Lines", Aug 3rd, 2016
6. 熊猫笔记，20150828
7. Fumihiko Arai, Yoshibumi Makabe, Yasunori Okawara and Teppei Nagano, ”Recent Trends in Cross-currency Basis“, Bank of Japan Review, 2016

宏观交易101第五课|Trading Skills

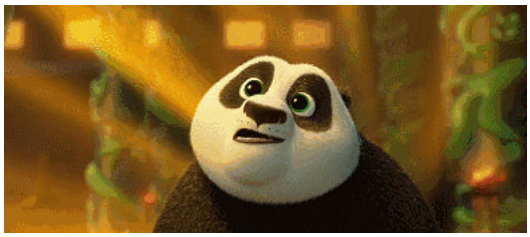
在进行了前面四节课的锤炼之后，相信大家都有些累了，这一次，本??准备让大家放松一下，和大家玩一个有意思的游戏——Trading 。

说是游戏，但还是有点小难度的，毕竟Trading 是一个复杂的环节，要把一锅煮得咕嘟咕嘟冒泡的浑浊液体，熬制成澄清的魔法药剂，也是需要时间，耐心与多多练习的。

不过别担心，让??带着你一起熬制Trading 这锅药，且看“宏观交易员的自我修养之五：《Trading Skills 》！

宏观交易员的自我修养之
Trading Skills

Trading Skills



Trading是最令人迷醉的环节。我们谈到的Monetary Economics、Macroeconomics乃至后面还将提及的Derivative Securities等部分，都来自于严谨的学术研究，所以有丰富的资料可以供Macro Trader修炼。然而虽然书店里堆满了《击败K线图》《21天致富》等东西，却没有一个商学院金融系有Trading这个研究方向或者出版Journal of Trading。究其原因，Trading其实是投资决策过程的一个执行环节，几乎所有影响投资决策的因素，从最广阔的宏观到最细微的微观，都会在Trading环节混杂起来。考虑到Trading环节需要面对一大锅成分复杂的可怕混合物，那么Trading环节的最大的困难，在我看来还真不是低买高卖，而是尽量把问题明确化清晰化，以把握自己需要主攻的方向。毕竟Trader有一个独特的好处是弱水三千我只取一瓢饮，我们并不需要在所有的市场上击败对手，只需要在某些方向上保持优势，就足以拿出很好的P&L了。

所以，本篇准备讨论的只是Trading环节的最常见的一部分小问题。希望通过这些讨论，协助Trader们找准自己的方向，避免被拖到不熟悉的方向上被迫作战。

1.“剑”“气”之争

在揣摩市场方向时，基本面派和技术派是最常见的两个流派。基本面交易员试图以宏观、估值、资金流等看得见摸得着的因素揣摩市场方向。这样的交易员往往有很好的教育背景，经受过正统的训练，在大型金融机构等重视价值投资的地方做交易。基本面交易员时常挂在嘴边的话是“这个分析可以这么证实或者证伪”，如果他们是剑客，必须比划出看得见摸得着的剑招才能讨论，可以称为剑宗。技术派则往往只看价格走势图，连纯粹的Brownian Motion他们都会去揣摩其中的原力。技术派认为价格走势本身就是一个强大的独立存在，蕴含丰富规律，好像看不见摸得着的“气”。他们坚信练好了“气”，就能降伏市场，可以称为气宗。

在话语权上，气宗往往不敌剑宗。因为气宗的沟通语言，比如头肩顶\背离\金叉等等，很容易神秘化，不像剑宗总是可以把问题分解到增长\通胀观点等可以讨论交流的状态。更何况气宗有一个先天劣势，就是如果气宗的方法真的有效，那么全市场参与者一起模仿时会轻松抹掉其有效性。气宗另一个劣势在于鱼龙混杂，名声容易被毁，尤其是在市场不成熟、投机盛行的市场，比如中国市场。在这样的市场，一些不愿意下功夫分析市场的参与者，往往希望有个快捷的暴富之路，气宗的语言偏偏很适合这种心理，影响了气宗的形象。

但是气宗有其独特的价值。第一，气宗其实是试图抓住人类的投机心理，只要做投资决策和执行的还是人类，气宗在应对投机情绪方面的重要性就毋庸置疑。市场参与者层面越广、投机心理越重、交易方法越初级，气宗越有效。第二，市场是一个超级复杂的群体，处在其中的人很多时候连推测到底发生了什么都是很奢侈的。这时候坚信市场可知的剑宗弟子很可能会涉险而不知，而气宗弟子，至少还能凭借一些条件反射一样的直觉，逃离险境。武林泰山北斗Paul Tudor Jones曾经这么批评过一些剑宗年轻弟子。

"...I see the younger generation hampered by the need to understand and rationalize why something should go up or down. Usually, by the time that becomes self-evident, the move is already over. When I got into the business, there was so little information on fundamentals, and what little information one could get was largely imperfect. We learned just to go with the chart. Why work when Mr. Market can do it for you? These days, there are many more deep intellectuals in the business, and that, coupled with the explosion of information on the Internet, creates the illusion that there is an explanation for everything and that the primary task is simply to find that explanation. As a result, technical analysis is at the bottom of the study list for many of the younger generation, particularly since the skill often requires them to close their eyes and trust the price action. The pain of gain is just too overwhelming for all of us to bear..."

Paul Tudor Jones, July 2008

所以总结一下，剑宗是基本功，剑宗修炼不扎实会被戳成筛子。但只学剑宗最多只能糊口。如果想要站在Flow之巅，就需要跟市场量价变化一起呼吸，揣摩细微之处，感受投机和恐惧心理的涨落。甚至很多时候要以气宗方法跨过复杂的决策流程，先出招再分析再续招。市场那么大，收钱的机会永远不缺，只要风险来临的时候总能占据安全的地方。

2.市场呼吸的节奏

作为一个Trader，对总体量价运行的节奏，比如有流动性窗口、策略容量、承受冲击的能力、交易成本等，必须有极其扎实的认识，才能在市场中布局。

为了盈利最大化，Trader要在最有利的策略上建立尽可能大的头寸。当Trader掌握的资金到达一定规模后，他的举动将对市场形成不可忽略的影响。不论是巨型资产管理机构的资产配置，还是需要在小众市场如某些商品，或者策略过于复杂而受制于某些流动性短板，都需要考虑"策略容量"问题。确定策略容量处于安全区间后，Trader需要以尽可能小的成本，尽快建立目标头寸，这时候需要尽可能利用流动性最好的时间、市场和渠道。

这个问题说来容易，但是现代金融市场的特色是时区、产品和交易方式都空前多样，很多菜鸟都误入流动性很差的窗口而不自知。例如外汇市场，亚洲由于开得最早，经常有冒冒失失的Trader在很差的流动性建错误的头寸，等到伦敦进来之后就被轻松收割。2015年希腊两次闹退欧，由于时间发生在欧洲晚上，亚洲时区有交易员自认为抢了先机，急吼吼的冒着薄弱的流动性高价开EUR空单，殊不知希腊的问题已经在欧洲银行业消化了数年，预案都不知道写了多少个，该跑的信贷早已跑光，退欧的冲击已经不大，反而是有可能导致欧洲银行业继续收缩跨境信贷，这时候对EUR是一个支撑而不是打压。等伦敦一进来，可怜的亚洲头寸就既输流动性又输钱，更可怜的是这帮亚洲交易员不长记性，第二次闹退欧还是干类似的Trade。幸好希腊闹了两次之后就从了，要是多闹几次，这些吃打不长教训的亚洲交易员们估计要把裤子都输光。

以上说的是二级，债券标售、新股发行等一级市场同样重要。一级市场的发行，既是流动性的集中释放渠道，也会大幅改变各个玩家手中的筹码对比。其发行结果，往往能影响二级市场很长时间。2011年6月PIMCO和Morgan Stanley在TIPS标售市场曾有一局精彩对决。当时，金融危机之后的债券牛市让Goldman Sachs赚的盆满钵满，而Morgan Stanley金融危机之后手足无措，错过很多利润，为此从挖来前Goldman Sachs明星交易员Edward Glenn Hadden主持美国国债债券自营。Hadden认为债市对通胀过于担忧，于是从2011年1月份开始，一边大手做空30年期TIPS一边向整个市场放风说TIPS多头要吃大亏。PIMCO当时是TIPS市场的大多头，他们没有被这种Bluff吓住，不仅全部接住Morgan Stanley砸的盘，还额外买了大量TIPS. 很快，二级市场所有玩家的关注点都集中到多空两大巨头上，这也意味着两巨头的交易都越来越难做，因为两家量都太大无法轻松脱身，而且，任何一方示弱，都可能恶化为一场大追杀。但是还有一个流动性窗口，就是2011年6月份的30年期TIPS标售。如果Morgan Stanley想撤退，这是几乎唯一能够不惊扰市场而大量买到TIPS以回补空头的机会。当时Morgan Stanley已经意识到情况比较危险，虽然嘴上继续Bluff，但是静悄悄的在TIPS标售中放了很多单。这时候就显示出PIMCO这些牛Trader揣测交易对手和市场情绪的本事了，他们以更凶悍的价格放了更多的标售单，最终成功抢到了数十亿的美元TIPS，不仅包抄了Morgan Stanley，还又抬高了TIPS的市价，在Morgan Stanley身上又戳了一刀。PIMCO付出的成本不低（标售价结果比二级市场贵5BPs，相当于好几毛钱），但是Morgan Stanley更窘迫。因为在接下来的三个月之内再无30年期TIPS供应，Morgan Stanley手头的空单将面临PIMCO肆无忌惮的Squeeze. 所以Morgan Stanley发现后路被包抄之后，唯一的选择是不带任何犹豫的斩仓，不计成本的回补空头，否则还要惨。估算PIMCO当天获利超过5千万美元，而Morgan Stanley的亏损则大大超过这个数字。

3.情景分析

了解了整个市场的呼吸节奏只是市场的第一步，接下来需要了解玩家。

一类特别的玩家是交易渠道本身，也就是银行等玩家。其重要性，首先体现在银行是重要的价格确定方。在《Monetary Economics》，我们反复强调银行这类特殊的玩家在Flow Chart中的重要性，还在《Case Study：隔山打牛的货币基金改革》揭示了一个银行资产负债出现扭曲时形成的奇幻例子。而且，银行通过收表扩表影响整个市场的流动性，进而影响套利机制等市场自我修复机制的效率，同时也决定整个市场对波动性的容忍度。对银行类玩家，最基础的分析是资产负债盈亏平衡分析。这里所说的负债，不仅仅包括资金负债，也包括监管资本规则等。

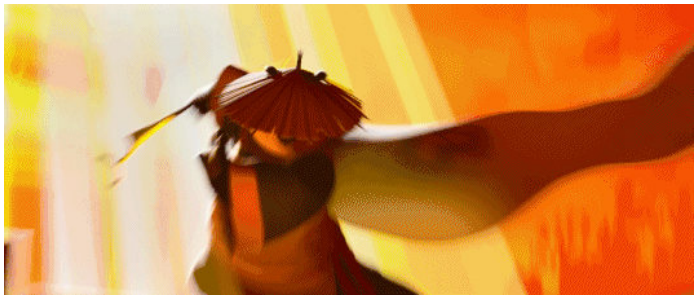
一个常见的误区是把玩家的重要性等同于其体量。但是边际上最激进的玩家，即使其资金量只有一根稻草，有时候也可以压垮整头骆驼。所以交易量的重要性远大于资金量，会利用流动性窗口的老手的交易单的重要性远大于整个市场的常见交易模式，新玩家和最脆弱玩家的交易单的重要性很多时候又远大

于老手们的。而如果玩家放弃了定价权甘心做一个被动玩家，资金量再大、交易量再大，对市场也是路人甲甚至鱼腩。

分析其他玩家，应该是Trading游戏中最有意思的部分。Trader们都爱打德州扑克，因为这些Check/Call/Raise/Fold的游戏跟Trading中间各式各样的Bluff实在太相似了。

4.信息分析

Trading的一些方面，主要是Market Microstructure方向，近年来学术界已经有很丰富的研究，极大提升了行业对Order和Price Action等因素的理解。HFT和算法交易的盛行，已经把很多人类交易员扫地出门。那么人类在执行交易方面还有优势吗？我认为还是有的，因为信息搜集和协同



作战，是人类交易员的独特的优势。老道的Trader们都会尽力拉拢中立玩家加入自己的队伍（其实也是Front Run他们），同时Bluff试图站在对手盘一侧的玩家。监管抓的再狠，Trader们也会在曼哈顿中城的高档餐厅、Mayfair的Club和六本木的夜店里，交换打听到的资金流信息，讨论哪里有机会可以结伴干一票大的。在突袭Lehman和围猎伦敦鲸这样的大战役中，都可以看到对冲基金群体联手做局的影子。这是Trading，作为一个人与人的游戏，最有意思也是最有价值的地方。

戳蓝字，查看

往期“宏观交易员的自我修养”系列：

宏观交易101第一课丨货币银行体系的资金和信贷match

宏观交易101第二课丨跨境资金流动

宏观交易101第三课丨宏观交易入门

宏观交易101第四课丨Cash Study：货币基金改革如何隔山打牛

参考资料

1. Wall Street Journal, "Pimco Wins in Bet on Inflation", July 5, 2011
2. 熊猫笔记，20150707，20150708，20150729，20150731，20151230

宏观交易101第六课丨Monetary Economics　考试题参考答案

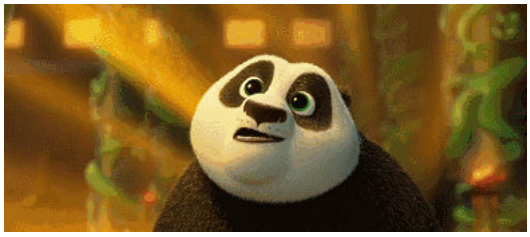
熊猫笔记惊现江湖之后，不断有大侠前来参阅其间奥妙。虽略有些难懂，令人阅而止步，可诸位也没有弃之而去，反而迎难而上，越战越勇。几番过后，纷纷表示颇为受益。于是，熊猫大侠留下了“十道题”，以敬各位大侠拜读的诚意。

这十道只有问题而没有答案的试题，想必是令各位焦灼困惑了一段日子，这种“猜中了开头，却看不清结尾”的感觉，确实...啧啧啧。

幸而，熊猫大侠云游归来，在书房中大笔一挥，写下这份“烤题答案”，愿与各位侠客研究一番。“世界上本没有答案，答得的人多了，自然就成型了。”

下面??，??为大侠们倾情奉上期待已久的“十道题之详解”！

答案详解



1 发问：

文中提到“在没有信贷机制的交易市场，比如居民住房市场，有多少房子就只能买卖多少房子。在有信贷机制的市场，例如货币银行领域，银行即使现在只有比较少的现金，也可以向银行的存款客户承诺大量的兑付。”其实除了货币信贷市场，还有很多市场有类似的信贷机制，使得参与者可以以较少的基础资产，维持较大的偿付或者交割Claim。例如美国国债市场，由于债券回购市场很便利，国债交易员即使库存券很少，也可以向客户提供大量的Offer盘。国债市场之外，请再举出至少三个存在类似信贷机制的市场（其信贷机制必须是成熟的市场惯例而不是私下约定）。中国银行间国债市场是这样的市场吗？

作答 ??

允许借资产、而且借入资产时发生真实交割的交易机制，例如出借、回购、租赁和Swap等机制的交易市场，就是信贷市场。比如存在融券做空机制的股票市场和存在租赁机制的黄金交易市场。有一个很有意思的例子是美国MBS的Dollar Roll市场，对Dollar Roll市场是否为信贷机制，我一开始认为No但是最终认为是Yes。

信贷机制的最大优势是使得OTC做市成为可能。假设一个有一万个参与者的债券市场，如果没有金融中介，一个买家如果想找一张债券的最优报价，他将不得不联系所有一万个参与者要求报价。这是个可怕的工作量，所以必须有金融中介帮忙。金融中介的初级形式是Broker，亦即大家都把信息报到Broker处，然后对照Broker汇总编制的信息，请求Broker为自己寻找另一方的实际需求者，亦即撮合交易。但这还只是价，玩家还需要考虑量是否足够、交易对手是否存在违约风险等等因素。即使有Broker协助，这个谈判过程也是耗时耗力的，而且谈不成交易的可能性很大，所以在Broker协助下的金融交易仍然效率不高。

比较高级的形式是做市商，亦即实际需求者不需要主动报价，而是金融中介，亦即做市商根据市场状况随时提供买卖双边报价。其效率体现在三个方面，一是玩家不再需要跟市场所有其他一万个参与者打交道，只需要跟少数几个，比如说20个做市商打交道就行了，这样就省下了交易对手风险管理的麻烦。二是做市商看到的信息比较全面，价格发现机制强。全市场成交价、流动性、资金流这些重要的Price Action在做市这个环节体现比较充分，所以做市商的报价合理性比两个懵懂的玩家坐在一起谈一个懵懂价要好太多。三是监管和自律往往要求做市商必须Honor自己的报价，所以交易执行效率会比较高。相比之下，两个懵懂的玩家坐一起，谈不成交易是很常见的事情（我知道你们要谈中国债券市场的做市商，但我就是不接话）。

银行其实是基础货币的做市商，可以联系以上这些特征，用银行模式对比P2P这种非做市模式，理解一下做市的特点。

但是做市商的活动必须以能够大量融入标的证券为前提。市场涨涨跌跌是很正常的，做市商等金融中介的合理位置是净头寸敞口保持相对中性，专心维护流动性。但如果没有信贷机制还要做市，尤其是接买单，那么为了降低交割风险，做市商必须天然保持一个巨大的库存，这显然是把做市商置于不必要的价格波动冲击之下。而且，如果没有借券机制，做市商不得不把大量本来能互相轧平的交易全部放到市场上平仓，增添不必要的交易，流动性创造也会收到影响。

所以有了借券机制后，做市商的库存风险大大降低，可以专心维护流动性。流动性得到稳固，交割逼空风险大幅降低之后，套利活动的风险也大幅降低，有利于迅速抹平可能出现的市场扭曲，形成可信赖的定价曲线，例如国债收益率曲线（作为对比，可以看看美国国债市场老券新券利差和中国国债市场老券新券利差），或者远期升贴水曲线。一类特别的套利活动是借助标的证券融资，赚取Financing Advantage，例如美国国债市场On The Run券需求很旺盛，因此持有On The Run券的投资者出借债券获得现金，然后将现金投资于货币市场，是非常成熟的交易，息差非常稳固，称为On The Run券的Repo Specialness。

本题的陷阱是有人会回答某些衍生品市场，例如期货市场。但我不认为期货市场是信贷机制，虽然期货市场和信贷机制极为相似，只存在极为细微的差异。

其一是影响范围，信贷机制主要是服务现货的交割或者支付，影响范围极广。而期货市场主要是提供一个不需要考虑即时交割的交易市场，即使是近月、实物交割的期货合同，最终进入交割的量相比平均持仓量也是很小的。

其二是定价机制，信贷机制往往形成一个完整的，从隔夜到长期限的利率曲线。其核心往往是隔夜或者最短期的利率，参与者对长期利率定价时，最主要是结合供需考虑隔夜利率的变化趋势。由于模式稳定、影响范围广，信贷机制驱动的Flow往往比较强，形成的经济故事确定性较高。比如利率曲线出现倒挂时，银行扩张信贷会面临损失，从而导致经济衰退，这是一个很强的故事。但期货市场的近远月价差曲线受影响的因素要多得多。

其三，做市商做市机制完全不同。信贷机制下，做市商的可以随心所欲的借入借出证券，用基础资产构建整条基础资产信贷利率曲线，为客户的需求创造出一个衍生品，甚至通过基差交易为相似的基础资产做市（比如日本银行业有TIBOR和LIBOR基差交易），做市空间很大。但是标准化的期货市场，做市商做市的其实是一张张合约，不是整条基础资产信贷利率曲线，买卖双方客户的交易单必须完全Match才能完成做市，灵活性较信贷机制大大受限。

最有意思的是MBS的Dollar Roll市场。读者可自行寻找关于MBS TBA和Dollar Roll的背景材料阅读。我开始认为MBS TBA看起来不过是每月交割一次的场外衍生品而已，类似衍生品市场，所以不是信贷机制。但是后来考虑到MBS的交割中，每月一次的TBA集中交割是主流交割形式，随时交割反而不是主流交割形式，所以Dollar Roll完全符合信贷机制服务于交割、影响范围广、基准利率（隔月）地位明显等特征，唯一的不同，在于这个信贷机制的时间Tenor最短是月度，而不是隔夜。

我收到的一些其他正确答案如下：

（1）. CNAV（固定面值）货币基金的货币信贷机制。这个答案很讨巧，但是非常正确。20160928《Case Study：隔山打牛的货币基金改革》正好有一段详细说明，提到CNAV货币基金是“信贷机制”，而VNAV基金是“归集机制”。最有意思的是，这个货币信贷机制不是在金融机构的信贷收支表上产生的，而是在基金产生的。其最大弱点，就是基金没有Capital Buffer以维护信贷机制的潜在风险。

（2）. FX Swap是FX Spot市场的信贷机制。这个答案也应该判定为正确，但是我认为这个信贷机制是由金融机构的货币信贷机制衍生出来的，所以不算是独立的信贷机制。

至此读者应该已经得到最后一问的答案。美国国债市场的回购主要是三方回购和买断式回购，参与者做回购的目的是借到自己做空的债券，所以是完整的信贷机制。中国国债市场主要是质押式回购，参与者往往不指定交割券，不是以借券到手为目的，只是一个简单的抵押融资，所以不能认为是信贷机制。由此对国债交易的影响，读者可以自行推演。

2.发问：

如果把央行主动向市场投放基础货币的利率定义为基础利率下限，那么正常的货币银行体系，是否可能出现信贷利率低于基础利率下限的情况？如果出现了这种情况，意味着什么？作为一个Macro Trader，你打算如何布局头寸？

作答 ??

同样收到一个意料之外但是完全正确的答案，表明我出题时想的还是不够全面，所以分两种情况说明如下。

正常的、价调控的货币银行体系，信贷利率是不应该低于基础利率下限的，因为银行以基础利率从央行获得基础货币，再成倍扩张自己的信贷收支表。如果信贷利率低于基础利率下限，意味着银行贴钱贷款，这显然是不对劲的。如果观察到这种现象，肯定是货币银行系统发生了扭曲。可能的情况如下：

（1）.因为抵押品稀缺造成信贷利率超低，甚至低于基础利率下限的情况。

参考第1个问题中提到的Financing Advantage Trade，以债券市场的回购交易为例，其交易形式是持券者与借券者的一笔回购交易，可以描述为持券者以券做抵押，向借券者借一笔现金。借券者需要支付Financing Advantage，一般不会单独支付，而是直接在市场回购基准利率上扣减掉这个Financing Advantage，其结果就是回购利率。所以这类回购的利率往往显著低于市场回购基准利率，低于央行基础利率的时候也很常见。

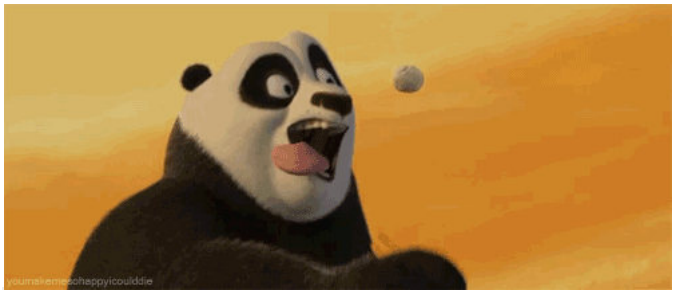
这个情形比较简单，做头寸也很简单，Trader只需要尽量出借自己的库存，赚人家白送的钱。

（2）. 因为流动性陷阱形成的信贷利率低于基础利率的情况。

当利率已经接近0的时候，金融机构持有现金的机会成本很低。如果实体经济相当差，金融机构对经济下行风险的担忧远远高于持有现金的机会成本，所以央行再继续增加流动性供应，金融机构也不愿放贷，宁愿放在央行账户形成超额准备金，就形成流动性陷阱。此时，如果有一些特殊的实体，基本不存在违约或流动性风险，但又无法从央行直接获得信贷（两个典型的例子：货币基金，外国央行的储备管理部门），但又有流动性需求。某些时候，金融机构可能会愿意把央行硬赛过来而且完全不打算回收的超额准备金贷给这些实体，这时候只要利率高于超额准备金利率，贷款就合算。再注意到央行主动投放的利率往往高于超额准备金利率（否则金融机构拿到钱根本不用放贷，摆在央行账户上就赚到套利了），所以会可能出现信贷利率低于基础利率、高于超额准备金利率的情况。

Bernanke在其回忆录《The Courage to Act》第15章中提到了2009年美国银行体系就出现了这种情况。当时联储通过QE向银行体系注入了大量现金，而且对超额准备金不付息，所以某些地方出现了银行拆借利率比Fed Funds Rate还低的现象，影响了联储对基准利率的控制力。Bernanke解决的办法是动用TARP的钱向超额准备金付息，相当于不动声色的紧缩，算是把基准利率又按回轨道上。

如何根据这个现象做Trade?也没有统一的答案，需要根据当时的情况而定。例如股市，美国流动性陷阱的后果是大大提升了市场对联储托底的信心，提振风险偏好，有利于股市。但是日本流动性陷阱，银行和企业主要的做法是利用流动性消化资产负债表的窟窿，不利于股市。对债市，如果观察到流动性陷阱的初期症状，而市场其他人还没有意识到，应该立即进场去增加国债的头寸，因为无处可去的资金会意识到国债是很好的去处，尤其是短端国债。Curve Trade也很有意思，最开始大家会哄抢短端国债，所以如果不想冒Duration Exposure，可以做Curve Steepen。哄抢短端国债的高峰期过后，资金的兴趣往往会逐渐延伸到长端，此时就要做Curve Flatten。



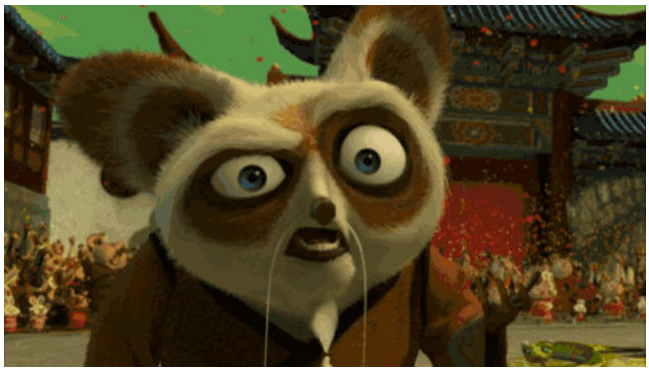
3.发问：

你是FDIC主席。闺蜜FED主席说货币政策工具中有很多工具能影响银行风险管理，而银行风险监管工具没有能影响宏观通胀的，所以才有“宏观审慎管理”，不可能有“微观通胀管理”。你很不爽这个自大的碧池，那么你打算拿什么驳倒她?

作答 ??

在Monetary Economics篇多次提到，实体经济往往是获得Credit而不是Money。因此，货币宽松只有在形成信贷宽松的时候，才能形成通胀压力。所以，固然货币政策放宽是通胀因素，银行信贷风险管理政策的变动，例如监管资本政策的变化，同样也会影响到通胀。比如FDIC如果宣布增加存款保险费率25个BPs，对货币信贷的影响和Fed加25个BPs准备金是等同的。

其实在信贷方面，宏观和微观的分界并不明确。例如存款准备金，现在看来是一个宏观调控措施，但是其本意完全是一个微观流动性风险管理措施。



4.发问：

同样是Unsecured Borrowing，为什么LIBOR o/n和Fed Funds Rate之间存在可观的Spread？作为一个Macro Trader，你认为LIBOR和FFR之间有套利机制吗？如果有，你打算如何做这种套利?

作答 ??

Fed Funds交易都是隔夜的，其目的往往不是维持信贷收支表，而是服务于结算中的现金需求。一家银行可能上午需要融入一些Fed Funds以完成一些结算，下午就发现自己的账户上钱多了，又想要融出Fed Funds。如果确实发现信贷收支表太庞大，需要更多的Fed Funds，银行不会靠隔夜Roll Fed Funds维持，这样太危险，只能拆入长期资金甚至直接发长期债券。所以，Fed Funds是真正的基础货币利率。

LIBOR是一个离岸利率。《Monetary Economics》和《熊猫笔记：20160118》篇提到，离岸市场是以结算行的在岸账户为基础货币，扩张出来的货币银行系统。所以画离岸货币的Flow Chart时，我习惯于把离岸货币银行系统放在与企业等同的层级。所以，可以认为LIBOR是由Fed Funds扩张出来的利率。既然如此，LIBOR和Fed Funds之间的利差，可以比照基础货币利率与贷款利率之间的利差来理解。其套利机制，说起来其实再平淡不过，就是简单的商业银行模式，亦即从央行获得基础货币再贷给企业或者非银金融机构。LIBOR和Fed Funds的套利也是一样，从在岸获得美元，在离岸扩张以LIBOR为基准利率的信贷收支表。

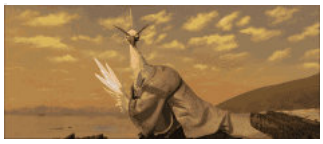
5.发问：

为什么金融危机期间Eurodollar的流动性会非常紧，逼的联储大量投放Swap，但Onshore流动性并未紧到迫使联储直接投放流动性的地步（QE是2009年才做的）？那么可观的Onshore/Offshore Spread为什么无法吸引商业银行从Onshore向Eurodollar搬运USD套利？

作答 ??

《Monetary Economics》提到，Offshore货币银行系统由于没有央行作为流动性的最终维护者，其流动性踩踏往往比Onshore更加凶猛。一旦Offshore市场有流动性紧缩的征兆，理性的参与者千万不可被高企的拆借利率所打动而拆出资金。流动性的坍塌可能远远超出你的估计，如果不知深浅就把宝贵的流动性消耗了，很可能会被Liability下家逼死。但凡你有Liability，就一定要守住流动性。

因为这是整个货币体系的收缩，所以如果单靠商业银行在Onshore/Offshore搬运基础货币，除非能搬过去相当于整个Offshore市场的资金量，否则就是杯水车薪。为缓解金融危机期间Eurodollar的紧缩，联储与各国央行进行了大量的Swap操作，量级是天文数字，巅峰时期接近3000亿美元。此外，由于各国央行天然是流动性维护者，加之央行不像商业机构那样有美元Liability，所以才能起到安抚商业机构信心，恢复流动性的作用。



6.发问：

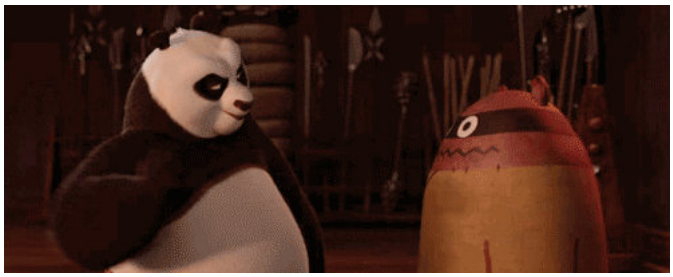
为什么离岸的LIBOR对在岸的美元Swap、公司债等资产有基准利率意义？在岸美元也有Prime Rate，为什么重要性无法超过离岸的LIBOR?

作答 ??

参考第4题，LIBOR本身就是由基础利率派生出的信贷利率。这么说不仅仅是因为看着像，而且有信贷机制和套利机制确保（作为Trader一定要记住这一点，光看着像还不行，必须要有一个机制确保其状态）。所以Offshore的LIBOR可以对Onshore美元的信贷类资产如Swap、衍生品和公司债起到基准利率作用。Onshore的其他利率包括Prime Rate之所以无法挑战LIBOR，有好些原因。

其一，历史沿革，LIBOR形成很早，想改一代代延续下来的Market Practice，尤其是法律、交易文档，非常费事。如果不是有LIBOR操纵丑闻之类的影响LIBOR地位，没人愿意花钱花力气改。日本银行业试图以TIBOR对抗LIBOR的努力就是例子。

其二，技术层面上，伦敦比纽约早5个时区。如果你是一个Swap Trader，你知道LIBOR和Prime Rate是一个故事，那么一大早进办公室的时候，你是愿意干等着旁边做Prime Rate的Trader跟人扭打撕咬，在他们打出结果之前不对任何交易报价，还是愿意看一眼5个小时之前已经扭打撕咬出来的LIBOR，然后轻松给交易报价?

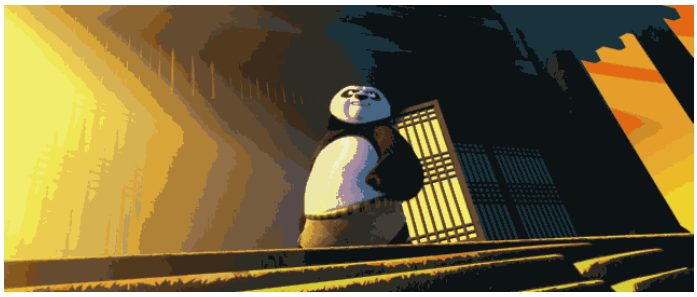


7. 发问：

一些研究表明EM的资本管制措施，比如资本流动税，往往在面临资本流入压力的时候比较有效，在资本流出压力时无效，结合Monetary Economics解释这个现象。

作答 ??

在《Monetary Economics》的Cross Border Capital Flow部分中，我们提到DM对EM的资本流动很多时候是套利方式。既然是套利，资本流入时增加管制措施包括资本流动税，就是降低了套利收益，效果很扎实。但如果是资本流出，往往会伴随流动性紧缩和政治不确定性，这时候套利者们担忧的就不是套利收益问题了，而是本金安全，这时候无论给他们怎么添堵，都挡不住套利者夺路而逃的冲动。



8.发问：

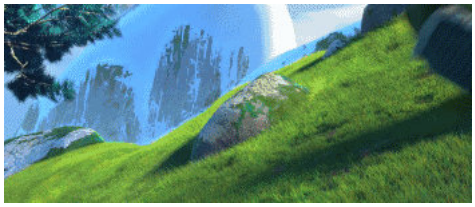
数字货币的狂热粉丝们试图游说政府彻底以数字货币取代现金，实现John Cochrane的《没有现金的世界》（A world without cash）里描绘的美好前景。商业银行强力反击，声称数字化货币将是另一种形式的金本位，按照Monetary Economics篇中的描述，突如其来的流动性紧缩将时不时威胁经济和货币体系。你同意商业银行的意见吗？如果商业银行说的有道理，数字货币体系有没有办法克服这个缺陷？

附《没有现金的世界》链接地址：http://johnhcochrane.blogspot.com/2016/08/a-world-without-cash.html

作答 ??

曾经有人作人类社会信贷增长图，显示史前社会一直到19世纪，信贷都是线性增长，而且速度极低，但进入20世纪以后出现了爆发性的指数型大增长。作图者试图说明债务增长是现代经济的威胁，但是我看到的图景正好相反。这个图显示，在19世纪以前，不知道出现过多少次经济增长被突然来临的流动性紧缩所打断。进入现代Fiat Money体系以后，央行可以调节基础货币的供应，金本位这种基础货币无法根据经济活动调节的货币体系退出经济体系，持续的经济增长才成为可能。

所以即使出现了《Monetary Economics》结尾中援引John Cochrane的文章所描述的未来货币体系，其基础货币也必须能够调节。需要特别指出的是，虽然现代Fiat Money体系的流动性紧缩往往是因为商业银行惜贷而产生，但即使没有商业银行，也可能出现流动性紧缩，因为只要货币同时具有储值和支付两大功能，那么持有者就必然面临是支付还是储值的抉择。如果太多持有者只愿意储值不愿意支付，就会形成流动性紧缩，现代Fiat Money体系建立前的流动性紧缩就是这么产生的。所以，未来货币体系，央行必须具备印数字货币的职能，否则相当危险。这也是我不看好比特币的原因，因为没有一个中央方。



9.发问：

2016年12月，中国银行间债券市场因为代持违约事件引起了巨大的交易对手风险恐慌，一些参与者建议中国债券市场全面由OTC形式转型为交易对手风险容易管理的交易所形式，你同意这个建议吗？

作答 ??

如果货币信贷银行体系主要采用交易所集中清算模式来结算债券交割，那么这个货币经济体将无法避免突如其来的流动性冲击。

由于交易所不可能承担会员的结算风险，所有的交易都必须在提供了足够的资金和债券之后才能进行。考虑到正常的货币信贷银行体系中，债券尤其是国债是重要的利率传导工具，因此交易量巨大，那么完全依赖交易所集中结算国债交易将意味着如果债券市场交易量突然增加，将会有大量的资金和债券被冻结以应对交割，这种突然性的流动性紧缩对货币经济的正常运行几乎是灾难性的。相比之下，OTC市场很多参与者扮演了信贷做市商的角色（参考第1题答案），自身承担结算风险，从而有效的向整个市场提供结算流动性，缓解这种冲击。即使简单加一个允许轧差交易的结算代理人，都能够极大提高结算效率。读者可以进一步比较“保证金杠杆交易”，“日内透支额度”，“轧差清算”等概念，理解形形色色的中间商是如何通过信贷机制降低结算风险和流动性风险的。

《Monetary Economics》中的大部分内容，尤其是以及本篇的第8题和第9题，描述的是一个浅显的事实，亦即信贷机制是承担企业和居民的未来不能履约风险而换取整个系统的流动性运转自如。微观层面对结算风险要求越严格，信贷机制调节能力越差，宏观层面越容易陷入整体流动性紧缩。

10.发问：判断题

如果认为“否”，需要简短说明理由。

（1）. 中国M2/GDP远高于美国，说明中国经济杠杆率远高于美国。

（2）. 受益于Flight to Quality的避险资产或者是货币银行系统的基础货币，或者是有基础货币职能的资产如国债、黄金等。

（3）. 国际化货币必须伴随稳定的经常项目逆差。

作答 ??

（1）否

关于M2/GDP的论述已有很多，奇怪的是还有好些人前仆后续的犯错误，有些还号称是大拿。其实说穿了很简单，在美国获得杠杆，大部分时候是直接发债，不是找银行借钱，所以M2统计不能描述美国杠杆率。美国企业和居民的总信贷数字，可以参阅联储的Z1表（Financial Accounts of the United States），2015年年底的数字是相当于GDP 245%，请读者自行计算中国的数字对比中美经济体的杠杆率。

（2）否

避险资产不一定是安全资产。安全资产指在主流货币体系崩溃后仍能有希望承担货币职能的资产，比如瑞士法郎和黄金。避险资产的含义则要广得多，只要受益于Flight to Quality都是避险资产。可能是来自于货币银行和影子货币银行体系的流动性收缩，这时候现金等基础货币和国债等承担部分基础货币职能的资产，可以成为避险资产。也可能是来自于Flight to Quality时空头无法借助信贷体系扩张，反而被逼仓，典型的例子比如日元。

关于避险资产和安全资产，在接下来的《Case Study：日元避险货币之宝宝心里苦但是宝宝不说》会有进一步的说明。

（3）是

国际贸易和货币银行学，貌似是两个题目，但是其实有紧密联系。国际化货币意味着有一个机制按照经济活动的需求向离岸输出基础货币，否则离岸货币将时常被货币紧缩所困扰，无法发展。这个机制只能是经常项目。历史上确实有通过资本项目输出货币的情形，比如美元的马歇尔计划和日元的黑字环流计划，但是资本项目很难稳定，而且往往要依靠政府拔苗助长，不利于市场培育。只有经常项目和经济活动结合紧密，能起到稳定输出货币的作用。

国际化货币必须是一个逆差国，其实这就是国际金融学中的“特里芬悖论”。

往期文章回顾

宏观交易101第一课丨货币银行体系的资金和信贷match

宏观交易101第二课丨跨境资金流动

宏观交易101第三课丨宏观交易入门（附试题）

宏观交易101第四课丨Cash Study：货币基金改革如何隔山打牛

宏观交易101第五课丨Trading Skills

宏观交易101第七课丨Derivative Securities & Other Skills for Macro T rading

上一期的内容是Trading Skills, 今天则再介绍一下Derivative Securities & Other Skills for Macro Trading，以及两个case sturdy。大家食用愉快！

宏观交易101第一课丨货币银行体系的资金和信贷match

宏观交易101第二课丨跨境资金流动

宏观交易101第三课丨宏观交易入门（附试题）

宏观交易101第四课丨Cash Study：货币基金改革如何隔山打牛

宏观交易101第五课丨Trading Skills

宏观交易101第六课丨Monetary Economics 考试题参考答案

宏观交易入门之二

Derivative Securities & Other Skills for Macro T rading & Case Study

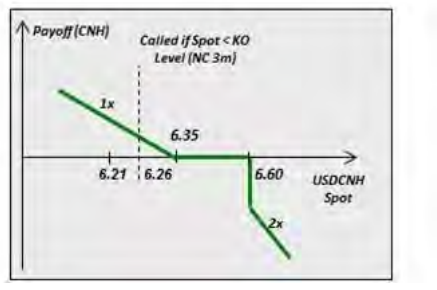
1.Derivative Securities

在前面数篇中，我们反复强调Trader最重要的工作应该是观察资金流。这样就有一个很自然的推论：任何可能显著推动资金流的因素，都是需要Trader密切关注的。所以，由于认为负债对资产的驱动是影响力最大的资金流，我们将Monetary Economics放在最重要的Skill Sets之列。此外，还专门开《Trading Skills》篇讨论观察资金流和布局Front Run的技巧。如果还有其他因素可能对资金流形成类似的影响，Trader也都必须一丝不苟的观察和推演。在Macro Finance 领域，Derivatives 同样也可能引发重大的资金流转换。因为Macro Finance最常见的工具，如利率、汇率和股指，都是Derivatives呼风唤雨的领域，许多震撼市场的资金流是Derivatives产生的。而且，由于Derivatives引起的资金流规律性更强一些，如果能把握好这类型的资金流，就有能力做出非常漂亮的Trade. 所以，称职的Macro Trader必须有能力做出扎实的Derivatives分析。

Derivatives同样也是很折磨人、门槛极高的一个行当，好在Macro Trader们学习Derivatives并不是要成为Financial Engineer，只是希望推演出市场上的Derivatives 头寸在市场波动时其Hedging 方式可能产生的资金流。所以Macro Trader并不需要精通Stochastic Calculus，但必须对如下问题有透彻的了解。第一，市场结构问题，包括谁在买Derivatives?为什么买Derivatives?投行怎么卖Derivatives？这其实也是分析如下问题：作为一个金融产品，Derivatives的负债是如何驱动其资产的。第二，Derivative Flow 如何产生，尤其是Gamma和Vega的变化是否可能产生大量的Delta Hedge 需求或者Volatility Hedge 需求。这些问题的答案只能从金融市场客户特性中寻找，而且每个市场的客户习惯都不一样。比如爱投机的华人客户喜欢FICC领域的杠杆Exotic Option类产品，而韩国客户喜欢大手大脚做欧美股指类衍生品。所以，Macro Trader最好是请教一位Derivative Sales或者Structurer，不一定需要请教Quant.

《熊猫笔记：20150824》记述了一个奇特的案例。我们都记得811汇改之后，CNH缓步贬值了大约两个星期，然后突然在8月24日出现巨量的USD/CNH买盘。与此同时，CNH的Vol大幅飙升，又影响到其他CNH资产。如果从CNY汇率机制上找这个现象的解释，肯定一无所获，因为这个奇特现象的发源地，不是CNY市场，而是CNH Derivatives市场。

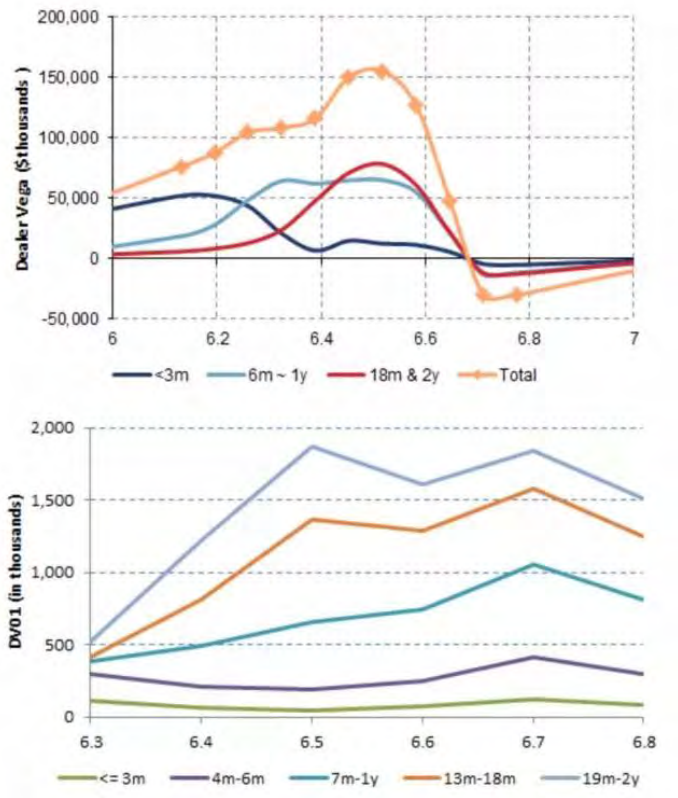
我们提到华人特别喜欢汇率方面的杠杆投机Exotic Option. 从2013年开始，由于PBoC长期压了汇率波动性，维持高Carry，许多香港和台湾资金涌入CNH市场，玩形形色色的杠杆CNH投机产品，以TARF最为流行。下图描述了一个典型的TARF结构。这个TARF期限是12个月，期间每月如果USD/CNH汇率低于6.35，客户就按6.35购入10mio USD/CNH，如果在6.35到6.6之间，就按当时市场价格购入10mio USD/CNH，如果USD/CNH高于6.6，则客户必须按6.35的价格购入20mio USD/CNH. 可以看出这是一个Exotic Option，其复杂的结构完全是为了适应华人投机资金又贪又抠门，既酷爱投机交易，又舍不得支付初始费用的习惯。



我们提到Macro Trader研究Derivatives 的视角是Flow观察者而不是Quant，所以不需要掌握Exotic Option中那些折磨人的数学工具，只需要了解其结构后，知道如何使用Quant们做好的Derivatives软件做分析就可以。对接受过Derivatives基本训练的Trader，使用这些软件计算这些TARF的Delta、Gamma和Vega并不难，计算结果见下图。再结合Sales和Structurer提供的估算，可知2015年8月前后市场上大约有400亿美元的CNH TARF. 出售这些TARF的投行需要时刻对冲额外的Vega和Gamma. 从Vega图可见，由于TARF结构复杂，对冲需求并不是线性的，在6.4以上，也就是TARF的Strike附近，Vega会突然增加，产生大量的USD/CNH Vol买盘和USD/CNH Spot买盘。

811以后将近两周时间，CNH慢慢贬值，市场波动看似缓和，其实已经暗流涌动接近Vega爆发点了。8月22日是个周五，当天纽约时间，USD/CNH汇率悄悄突破了爆发点。收市之后，许多投行的拿收市价格一算，就发现自己的TARF Book各项风险指标普遍大幅超标，必须加量对冲了。于是周一8月24日，CNH市场突然被汹涌的USD/CNH Vol买盘和USD/CNH Spot买盘淹没。再按照《Case Study: HIBOR and CNH FX Puzzle》中推演的机制，大量的USD/CNH买盘将剧烈消耗CNH流动性，同时催动所有CNH资产的全面下跌，又进一步产生大量的USD/CNH平仓需求，形成螺旋式的自我强化。幸好金管局及时向市场提供了CNH流动性，否则香港金融机构可能会面临意外的金融稳定风险。

所以，了解CNH Derivatives市场之后，才能解释为什么CNH市场的最大冲击会拖了两个星期而姗姗来迟，但一露面就凶悍残忍，几乎重手废掉了整个

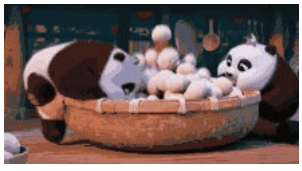


CNH市场，甚至威胁金融稳定。

2.Politics

政治分析是Macro Trader最容易迷失的方向。一方面，政治因素往往对整个市场都形成震撼。对政治环境和金融环境欠发达的EM，政治变乱往往能让EM金融市场不死也只剩半条命，给空头们巨大的机会。但另一方面，政治分析不是Trader们所擅长，但是狂妄自大的Trader们（这个行当极易偏执，很难谦卑），往往读几篇街头报告就信心十足，轻率的犯下错误。典型的例子比如公投之类的，特点是参与人群无序、信息杂乱、方向混沌，这往往成为神人最集中的地方，隔壁老王跟基友撸个串回来都能讲一大堆故事做一个高明的判断，但其实都是盲人摸象。

那么Trader们是否需要恶补自己的Politics知识？有条件的话补一点当然最好，但我也不赞成把这当成Trader的主攻方向。Trader们最擅长的，应该还是对资金流Flow Chart的把握。所以，我建议Trader们研读政治分析时不以判断结果为目的，而是以设计尽可能全面的Scenario，仔细推演。对Trader们，最合理的位置是不主动在政治事件上下注，但是对Political Scenario有扎扎实实的推演。所以当Flow Chart面临冲击时，Trader们总能有足够的时间和空间，识别Flow Chart出现的传导失误或者低效转换，从而轻松的Front Run其他资金。



3.Portfolio Management Skills

以上我们花了大量的篇幅谈论Macro Trader/PM需要的技能，对Trader/PM需要的一般性技能较少涉及。这主要是因为一般性技能的资料已经足够丰富了，例如本人最喜欢的Efficiently Inefficient: How Smart Money Invests and Market Prices Are Determined，对Smart Money应该具备的技巧就有很丰富的阐述。

强调一下，Portfolio Managment方面的一般性技巧，例如Portfolio Analysis，是PM/Trader的基本功。如果基本功不合格，即使Monetary Economics/Macroeconomics 等方面非常出色，做一个PM/Trader也是有隐患的。这也是很多才华出众的人能做很好的研究，但是在PM/Trader职位上败退

的原因。作为研究员，你只需要练好一部分Skill Sets ，但如果是PM/Trader，那就需要掌握尽可能全的Skill Sets ，以避免犯错。如果掌握的Skill Sets有缺陷，建议PM/Trader应该有自知之明，宁可放弃一些很诱人的策略，以避免暴露短板。

越全面、越深刻的Macro Trader，对市场的认识往往会越来越市侩，尤其不应该把做史诗级Trade当成自己的职业理想。只要人还在市场中，该抓住的资金流基本抓到，愚蠢的错误尽量少犯，多年以后，一个不败传奇就自然而然的形成了。

4.Case Study： 日元避险货币之

宝宝心里苦但是宝宝不说

在前面数篇中，我一直有意无意暗示这么一个观点，对宏观金融而言，金融比宏观要重要的多。所以如果忽视金融体系尤其是信贷体系本身的消长，忽视形形色色的参与者扭打撕咬这一价格形成过程，光从经济基本面、通胀等宏观因素入手做头寸，会吃很多闷亏。然而交易员的训练，尤其是强调逻辑和价值分析的机构交易员，第一课往往是从宏观基本面开始的。所以很多交易员会犯把基本面分析当成是投资分析的错误。更有甚者，好些交易员犯错后会试图用基本面解释基本面以外的因素，这就是错上加错了。《引言》中提到的日元交易，曾经虐过一代代交易员，总结出的经验教训几箩筐都装不完。这里我们只谈其中一条，就是如何理解避险资产，如何跟随避险资金流做交易。

从基本面上看，避险资产和安全资产理解起来应该是非常简单直白的概念。然而爱抬杠的人士总能找出反例，比如有人会认为波动性小的资产是安全的（黄金怒了，DEEP OTM OPTION在旁边窃笑），有人会认为本金亏损风险小的资产是安全的（30Yr High Coupon Treasury Bond一脸懵逼），有人会认为赤字比例低的国家发行的国债是安全资产（美国国债表示不屑）...即使是同一类资产，安全资产和避险资产的概念也很糊涂，比如外汇和贵金属领域，一般认为黄金和瑞士法郎是避险资产，所谓乱世买黄金，但是日元这么一个债台高驻的币种，却一直是无人敢质疑的避险资产，另外一些币种的故事就更有意思，比如欧元，从2013年到2015年，一直听到有人念念有词，“Risk On有利于欧元这样的在险资产...不对，Risk Off有利于欧元这样的避险资产...又不对...欧元表现出奇特的特性，有时候避险有时候在险...”

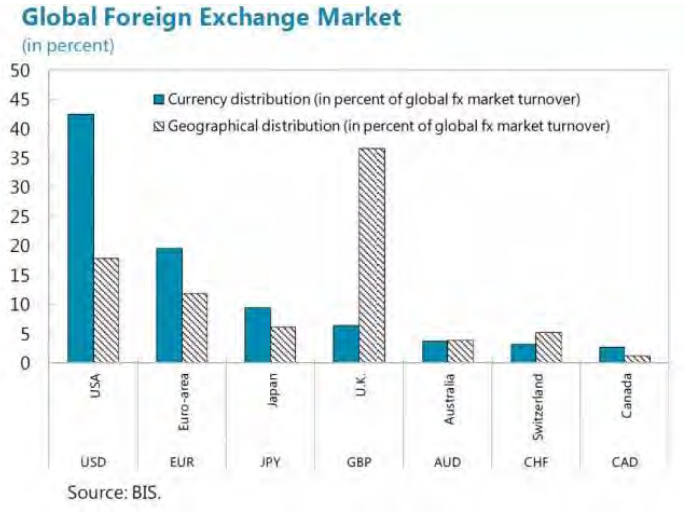
很多分析试图从基本面解释日元的避险属性，为此找出大量的理由，比如日元的外汇储备和国际贸易顺差等等，但如果顺着基本面，抬杠人士总能找出反例，比如挪威的人均储备和顺差更加亮眼，但挪威克朗在风险事件面前就比日元镇静的多。而且，即使能找到一个很强的基本面因素解释避险属性，如果跟随这个基本面因子的变化来做避险交易，几乎肯定是要交学费的。因为决定Price Action的是形形色色的参与者手中的头寸，只有通过观察市场参与者的行为，才能找到避险属性形成的原因。在这个方面，Botman, Filho & Lam (2013)，提供了一个很好的视角，解释日元避险属性的形成。

概括一点说，日元避险属性，是日本跨境证券投资头寸天然形成的。自从上世纪90年代日元进入零利率时代以后，日本积累起了大量的日元流动性，然而日本经济进入了漫长的资产负债表衰退，企业没有举债意愿，大批资金无处可去，加之资金确实低廉，降低了投机成本，所以日本从金融机构到个人投资者（鼎鼎大名的渡边太太们），都开始大规模投资海外证券市场。这样对银行交易渠道和流动性形成了很大依赖性，也形成了银行的风险敞口和资本占用。其一，在外汇风险管理方面，金融机构如银行和保险等普遍管理的比较严格，因此往往需要大量使用FX Swap/Forward 等工具对冲外汇敞口。这些工具主要是银行提供的，因此巨额的对外投资形成了巨大的FX Swap/Forward市场。衍生品交易随时需要市值重估，在衍生品盈亏较大时，还可能面临客户违约风险，因此衍生品交易，即使完全对冲，也经常形成对客户的敞口，所以多少都会占用银行风险资本。如果衍生品交易做市时承担了一些比较有吸引力Basis Risk或者Gamma Risk，占用的风险资本就更多了。其二，对个人投资者如渡边太太们，不管她们是否愿意承担外汇风险，由于她们往往不是在外国金融机构开证券投资户，而是购买日本银行业设计的理财产品例如Quanto等，所以银行也需要利用衍生品对冲。再考虑到这类理财产品往往附带杠杆或者期权，对冲盘就更有趣。其三，还有一些更激进的投资者，采用Prime Brokerage等方式杠杆借日元博外币，这时候银行其实是提供了一笔日元贷款，风险敞口就更高了。日元Carry Trade是这类跨境交易中最典型的，但是可以看出，这类跨境交易的范围远远不止日元Carry Trade，而是一个巨大的整体跨境证券投资头寸。日元货币政策越宽松，这个头寸越大。其通道，就是日元FX Swap/Forward等外汇衍生品市场。

《Monetary Economics 》篇提到，银行担忧流动性风险时，往往会收缩风险敞口，压一切需要消耗流动性的头寸，采取的手段包括降低信贷额度，提高衍生品担保金要求，或者提高信贷和衍生品的报价等等。作为日元跨境投资通道的FX Swap/Forward 市场，很不幸的处在对流动性最敏感的地方。所以每次Flight to Quality，日本投资者都会面临FX Swap成本升高，投资收益率下降，甚至追加保证金的窘境。如果是一个高杠杆的Carry Trade投机者，有可能不得不平仓，那么他出售外币头寸并买回日元以偿还信贷的行为，将推动日元升值。

这是最流行的解释，但作为一个认真的Trader，这个解释过于粗疏了。为什么这个现象独见于日元（美元和欧元都经历了0利率和负利率，丹麦克朗甚至早在2012年就进入负利率了）？为什么交易量巨大的日元外汇市场，竟然吸收不了一小批Carry Trade资金流？为什么Carry Trade建仓的时候不见日元大幅度贬值？....前篇多次提到，Trader是要做细活的，尤其是在观察市场参与者方面。解答这些问题，仍然需要从市场参与者，尤其是日本金融机构和欧美金融机构的行为入手。

虽然同为重要的国际化货币，但是日元的国际化，其实是做的相当差的。体现在金融机构上，就是日本金融机构的海外扩展和国际市场交易能力，相比欧美资金融机构，不在一个数量级上。比如日本金融机构境外扩展，经常有忽而大跃进忽而大回撤的现象。但是日元又是一个相当大的市场，在国际金融和国际贸易中相当重要。日本金融机构放弃的市场，欧美金融机构闷声不响的拿去了。以汇率为例，下图显示，约有1/3的日元交易，是在日本之外完成的。如果是日本资金投资海外债券和股票，欧美金融机构的能量还要大。如果欧美金融机构对日本资金流发言权太大，就会形成两个现象。



其一，日本投资者资金量大、过于集中而且单一的僵化行为模式，使之成为欧美资金融机构理想的Front Run 对象。而一旦Front Run日本投资者的资金流成为一个所有玩家口口相传的模式之后，反身性会促使所有玩家（包括认定日元基本面上没有看涨的理由的）都或多或少的调整自己的头寸，试图分一杯羹，或者避免被误伤。这种反身性形成了强悍的驱动因素，连日本金融机构都深陷其中，有时候最先Front Run日本韭菜的甚至是日本自己的金融机构。在2010年Fannie Mae和Freddie Mac改革中，日本投资者纷纷转投更安全的Ginnie Mae债券。于是各家金融机构的MBS交易员们都喜欢在早盘先哄抬一下Ginnie Mae债券价格，等到日本资金入场之后将自己手头的库存高价出给日本人。有趣的是，干这个活最积极的，竟然是一家日本金融机构，真是肥水不落外人田。

其二，究竟什么是风险事件，欧美金融机构说了算。中东出事了，肯定有损欧美金融机构，所以是风险事件。意大利银行出问题了，矮马太吓唬人了，毫无疑问风险事件。那么真正跟日本相关的风险事件呢？反而不好说了。比如泰国的政治经济风险，日本企业在泰国的投资其实非常大，泰国出麻烦会影响到日本企业和银行，但是如果欧美金融机构不认为泰国是风险，日本金融机构也不好意思太激动。如果一激动就买日元，搞不好会跑的太远。所谓拿头寸忌讳太出格，因为如果同桌玩家不跟，你就没有流动性，如果被同桌玩家发现，联手治你，那就更倒霉了。

所以，日元的避险特性，应该说是反身性在交易中的自我实现，与基本面并没有特别关系。由于这是一个交易现象，而非基本面现象，那么再按宏观交易员们习惯的增长\通胀\财政等基本面套路去交易，多半要吃闷亏。其一，由于其避险属性特异，相关资金流模式也不一样。比如同属避险资产的瑞士法郎，在Flight to Quality中往往能观察到显著的资金流入瑞士金融市场尤其是债券市场，但日本金融市场却没有这种资金流动，只在衍生品市场翻天覆地。所以不可按瑞士债券避险交易经验做日本债券避险交易。其二，反身性一旦形成就有很强的韧性，除非原有市场结构被破坏，否则将持续很久。比如2009年以后美元也进入了零利率，USDJPY Carry Trade的能量已经大幅度下降，但是日元的避险货币特性并未弱化，反而在安倍经济中进一步加强，因为玩家们判断日本投资者对外证券投资将只增不减，对银行渠道的依赖更强（至于是不是继续做Carry Trade反而没关系）。可能有读者会问进出口企业也是外汇市场重要玩家，为什么高昂的汇率不可以通过国际贸易顺逆差的方式矫正，答案是国际贸易资金流毕竟是一个有相当时滞的因素（往往以季为单位，因为订货周期通常为季度），而Macroeconomics 篇中提到，短期因素与中期因素冲突时，Trader是必须服从短期因素的。更何况国际贸易资金流是基于企业经营利润和现金流，相比金融机构的P&L和资金流，对汇率的偏离有一定容忍度，所以矫正效应有限。

应该说，日元避险货币特性，对日本金融系统是一个沉重的负担。其一，日本投资者为这个畸形的资金流付出了太多的成本。许多欧美金融机构都从Front Run日本资金中赚的盆满钵满。P家的一位MBS PM曾经洋洋得意的说，他在Ginnie Mae债券上的赚到的Alpha基本上都是日本人孝敬的。这种畸形的资金流，甚至能催生出一类很著名的交易，即所谓Yen Basis Trade。我估算，每年日本投资者仅仅在Yen Basis Trade上就孝敬给欧美金融机构数十亿美元买路钱。其二，避险货币特性，以及其他的Front Run 现象，严重干扰了日本金融政策的实施。比如避险特性时不时造成日元汇率的飙升，在日本的通缩上再踏上一只脚。又比如BoJ做QE试图扩张金融机构资产负债表的时候，几乎所有的JGB交易员都在Front Run BoJ，完全无视基本面，让BoJ的扩张性货币政策一股大力打在棉花上。所以，国际化货币是必须有强大的金融市场和强力的金融机构作为后盾的。曾经有人鼓吹人民币成为避险货币，把我吓的不轻，忍不住恶狠狠的说你才避险货币，你全家都避险货币。

真正值得人民币追求的，应该是安全资产而不是避险资产。这是两个完全不同的概念！安全资产指在主流货币体系崩溃后仍能有希望承担货币职能的资产，比如瑞士法郎和黄金。避险资产的含义则要广得多，只要受益于Flight to Quality都是避险资产。可能是来自于货币银行和影子货币银行体系的流动性收缩，这时候现金等基础货币和国债等承担部分基础货币职能的资产，可以成为避险资产。也可能是来自于Flight to Quality时空头无法借助信贷体系扩张，反而被逼仓，甚至是由于市场反身性自我强化形成的，典型的例子比如日元。

我们已经认识到，避险交易是一个市场现象，所以学习避险交易的时候，还是应该做细活，从资金流动开始，不应死抱着一两个指标。很多文献习惯于把VIX当成风险指标，但我不喜欢，因为VIX代表的是股票市场风险偏好的传导，在FICC领域VIX并不能包揽。如果要学习FICC领域的风险因子，那就应该从

驱动FICC领域风险偏好的资金流入手，日元汇率就是一个很好的入手点。在目前离岸美元主导全球金融市场的情况下，有一类因子将越来越重要，就是各种货币相对美元的Currency Basis. 有兴趣的读者，可以阅读参考资料中BIS和BoJ的论文。

5.Case Study： Jan 2016 HKD Raid

2016年1月，当人民币金融市场经受A股熔断和汇率大跌双重冲击的时候，香港金融市场也遭受了一场磨难。港币大贬值脱钩美元的忧虑搅得人心惶惶，港币汇率迫近弱方干预保证，金管局不得不频繁托住汇市，港币流动性紧张，恒指大跌...

我们前面提到，政治因素往往是搅动整个金融市场，尤其是EM金融市场的风暴根源。政治风险担忧，经常迫使企业削减CAPEX，金融机构削减风险敞口，导致经济下滑和金融市场疲软。政治风险加剧的时候，居民和企业还可能抽逃资本，导致通货紧缩，进一步对经济和金融形成压力。从2014年起，香港泛民派极度活跃，在香港的政治前途上留下阴影。泛民派甚至直接喊出“占领中环”的口号，金融机构对香港再有信心，面对堵在办公室门口的怒汉们，也不得不仔细考虑。泛民派对内地旅游者的攻击，则直接影响到香港支柱产业旅游业的收入。此外，人民币汇率大跌威胁香港金融稳定，A股熔断影响香港中资红筹...所有这一切，对香港金融市场，都是严重的困扰。这时候，空头似乎应该操起家伙什跃跃欲试？

如果只凭借这些认识就匆匆上阵，那很可能成为一株韭菜。金融市场看似变幻莫测的表面下，其实有很多严密而冷酷的法则。如果不做细活，不琢磨金融市场的结构，不顺着金融市场的法则而动，只凭借一些含糊、片面、惊乍的结论甚至传言就匆匆入场，多半要被割。实际上，虽然的确有许多空头大规模抛售港币现汇和港币远期，比如标志性的港币一年远期就曾经突破7.85的现汇弱方干预保证而触及7.88的高位，并形成了所谓联系汇率制一年之内会被打破的市场担忧，但如果韭菜依据这些迹象和判断就急匆匆抛港币就会吃闷亏。因为港币现汇和远期此后迅速的回调，几乎完全收复实地，还迫近7.75的强干预保证。

一个严肃的分析师必须是做细活，解剖一只麻雀的。所以我们不厌其烦讨论MonetaryEconomics/Macroeconomics/Market Microstructure/Derivatives等题目，在相关领域有一定铺垫之前不敢讨论跨市场的交易策略设计。现在我们简单分析一下这个例子中间的空头策略。这项工作本应该由Economist、Equity Analyst、Derivative Specialist和FX & IRS Trader组成的Strategist团队利用各个领域的数据和模型通过严谨扎实的分析完成。好在本文只是一篇段子，所以我们大幅度简化了分析。

Strategists发现了香港金融市场的几个结构性问题，也指出了某些市场认识的错误之处。其一，香港的联系汇率制，决定了港币流动性是依附于美元流动性的，港币本身的基础货币流动性池子不大，金管局发布的Monetary Statistics也证实了这一点。其二，恒生指数中，金融和地产的比例合计超过56%，与此相比，标准普尔500金融地产合计还不到17%，即使是英国金融时报指数，金融地产合计也才22%。这意味着香港股市对金融环境超级敏感。其三，香港常年位居地价最高的城市之一，居民房贷负担极重，而且主流房贷利率是挂钩HIBOR的。其他的讨论，比如说人民币市场对香港金融市场的影响，Strategists认为A股熔断确实可能影响H股投资者的情绪，但是CNH汇率对香港股市的影响就很淡了，甚至CNH贬值、资本出逃的过程，还是有利于香港金融机构的。

对金融体系的结构有了进一步的了解之后，一个更加扎实的策略就形成了。这个策略将主要的资金布置在股指和金融、地产股的空头上，同时还准备一部分资金，在合适的时点做空港币现汇和港币远期。建立好股市空头后，一面传播港币联系汇率制可能崩溃的传言，一面大手砸港币现汇和港币远期。如果吃瓜群众们惊疑不定跟着抛售港币换美元，迫使港币汇率接近7.85的弱方干预保证，金管局将不得不入场买港币抛美元。金管局回收港币的行为，以及吃瓜群众卖了港股换美元汇到境外的行为，将减少港币M0，因此提高港币HIBOR利率。同时，推升港币远期的行动也会推升港币HIBOR利率。由于80%的居民房贷都是跟随HIBOR浮动，一旦利率上升，居民将发现房贷利率陡增，房地产销售鸭梨山大。如果房地产下跌，整个银行和地产都会面临严重的打击，因此股市将大幅下行。

这个策略选取的薄弱点看似是港币汇率交易，其实是港币远期交易。《熊猫笔记：20160606》和《Trading Skills》篇都提到，建仓要选择流动性最好的窗口和产品，以最小的冲击成本建立最大的仓位，而Bluff 自然要选取流动性最差窗口和产品，以最少的资金高效率撼动市场，配合Bluff. 我们从香港金管局的相关数据中了解到，港币的利率衍生品，包括远期和掉期等，每日交易量确实不大，才数十亿美元，甚至不如CNH的利率衍生品。相比之下，港币外汇交易量是港币远期交易量的十倍左右。这意味着打翻远期比打翻现汇要容易得多。如果更谨慎一些，还需要考虑到潜在的对手，比如银行套利部门可能被利差吸引把砸下去的远期接走。所以还需要对银行有一定的研究，比如估算银行套利盘可动用的资金和资本占用，计算击退这些对手、把港币远期打到目标位所需要的资金量。

Average daily turnover of the Hong Kong FX market (by currency pair)

US$ billion

	Average daily turnover				
Currency pair	April 2016	Share (%)	April 2013	Share (%)	Change (%)
HKD / USD	53.8	12.3	47.3	17.2	13.7
HKD / RMB	0.7	0.2	0.5	0.2	52.8
of which: HKD / CNH	0.6	0.1	0.4	0.1	58.9
HKD / CNY	0.1	0.0	0.1	0.0	26.8
HKD / EUR	0.5	0.1	0.8	0.3	-42.1
HKD / JPY	0.3	0.1	0.4	0.1	-16.8
HKD / GBP	0.2	0.1	0.2	0.1	12.2
HKD / AUD	0.2	0.0	0.5	0.2	-56.4
HKD / CAD	0.1	0.0	0.1	0.0	26.7
HKD / Other	0.2	0.0	0.2	0.1	-5.0
Subtotal	*56.0*	*12.8*	*49.9*	*18.2*	*12.3*

Average daily turnover of the Hong Kong OTC interest rate derivatives market (by currency)

US$ billion

	Average daily turnover				
Currency	April 2016	Share (%)	April 2013	Share (%)	Change (%)
USD	47.6	43.4	3.5	12.5	1266.6
AUD	22.5	20.5	7.3	26.4	207.1
NZD	9.9	9.0	0.6	2.0	1677.4
RMB	6.2	5.7	2.0	7.3	205.4
of which: CNH	0.1	0.0	0.1	0.2	-4.7
CNY	6.2	5.6	2.0	7.1	211.0
KRW	5.7	5.2	4.1	14.8	38.6
HKD	4.7	4.3	1.8	6.4	167.7
SGD	3.0	2.7	0.6	2.0	424.6
JPY	3.0	2.7	2.4	8.6	23.8
INR	1.8	1.6	2.0	7.2	-11.3
MYR	1.6	1.5	0.8	3.0	89.4
THB	1.4	1.3	1.5	5.4	-7.6
EUR	1.1	1.0	0.9	3.3	23.1
TWD	0.8	0.7	0.1	0.5	488.1
GBP	0.3	0.3	0.1	0.3	266.4
Other	0.0	0.0	0.0	0.0	373.0
All currencies	**109.7**	**100.0**	**27.8**	**100.0**	**295.0**
Other OTC products	**0.1**		**0.1**		
Total OTC interest rate derivatives transactions	**109.8**		**27.9**		**293.5**

我们的Strategist团队的工作很不错，现在似乎有一个比较可靠的空头策略了，但是这还不够。如果靠这个策略就闷头杀进股市，可能赚了两天钱就节节败退了。恒指和恒生国企指数的确在1月中旬一溃千里，但是喘息了一个多月之后就开始迅速回升。而策略推演中的金管局干预 - 港币流动性趋紧 - 房贷走高 - 地产下跌 - 银行下跌 - 全面下跌的传导是一个比较缓和的过程，最快也需要数月。届时空头们多半又是丢盔弃甲。

从Strategist到PM/Trader，是一个艰难的、脱胎换骨的转换，许多成名人士都曾经犯过错误。《熊猫笔记：20151117 》和《岁末鸡汤》数篇中，都提到PM/Trader的综合职业要求。在本例中，有两点需要重申，一是Strategist 只需要在自己的领域做到登峰造极，而PM/Trader则需要全面修行、少犯错误，二是Strategist 只需要把数据、模型等信息钻研透彻，而PM/Trader则需要观察人。确切的说，是观察同桌玩家，因为同桌玩家的错误，往往会成为PM/Trader最好的P&L来源。

在金融市场，对手的实力和所有可能的异动，都需要纳入考虑。首先看港币现汇市场的玩家，我们注意到金管局实力极其雄厚，拥有三千多亿美元的外汇储备，而且，由于与中国人民银行的良好关系，随时可能得到中国人民银行的支援。所以在港币现汇和港币远期市场掀一点波浪，是有可能的，但是如果想击败金管局，实在是痴心妄想。这意味着两点，其一，空头的策略必须是一击必杀，迅速在汇率和远期市场搅起波澜然后利用恐慌心理收割股市。如果战事拖延开来，空头那点资金根本不足以应对。其二，这个策略是基于对官方机构循规蹈矩，所以只会按照空头的推演卖美元收港币流动性，而不会做出主动投放港币流动性、干预远期或者托股市这种非常规操作的判断。一旦金管局主动迎战，空头能跑多远就必须跑多远。

再看香港股市的玩家。Equity Analyst指出这里固然有担忧政治风险的国际投资人、对香港失望而离开的本土居民等空头，但也有为数不少的多头。我们强调Strategist 团队必须有Equity Analyst ，就是要把主战场的分析做到尽可能全面客观，避免犯被自己强烈心理暗示的方向引导过头的片面错误。大陆居民和企业实际控制的资金就是一个多头力量。香港的诸多H股，很多都是大陆企业。投资者普遍都有Home Bias，喜欢自己熟悉的股票和债券。香港作为大陆资本外流的第一站，给这些投资人提供了不小的吸引力，加之H股实在是便宜，刚刚忍受了A股折磨的大陆投资者见到打6折7折的H股，很难有不动心的。谈玩家必然要谈资金规模和交易习惯。资金规模只能通过常见的资本外流途径和资产管理机构的调研估算。交易习惯方面，这些投资者往往建仓步调比较慢。想快也快不起来，因为这类投资者的资金受限于通道效率，又很难搞到高效的交易途径例如杠杆融资。

此外，Equity Analyst还指出，港股的Valuation 实在太便宜了，对价值投资者，吸引力真的不小。如果EM金融市场的情绪稳定下来，港股就会对亚洲价值投资者们显得极具诱惑力。

另外一类不那么显山露水的玩家，就更有意思了。《Derivatives》篇特意提到韩国客户特别喜欢股指方面的衍生品，香港股市也是他们最喜欢的领域之一。TARF类似的结构化股指产品，韩国人买了不少，大部分是以估值便宜的恒指和估值更便宜的恒生国企指数做的。在2016年年初的时候，这类产品大概有200亿美元的规模。交易习惯方面，《Trading Skills》篇曾特别提醒要注意几类玩家，比如衍生品对冲盘、止损清仓盘和融资爆仓盘，因为他们为了获得流动性可以不计较价格，所以往往会对市场形成突然性、不见底的冲击。200亿美元规模不算大，但是港股疲软了很久，交易量常年才数百亿港币。一旦Vega和Gamma对冲盘涌入，孱弱的盘面根本无法承受这些奔流的卖单。而恒生国企指数逐步跌向8000点的时候，这些TARF的Vega和Gamma，不声不响的上来了。

空头真正的目标，其实是这些恒生国企TARF. 砸港币远期，抬高HIBOR，Bluff港股投资者，其实主要目的是拖延潜在的多头如内地投资者。至于CNH崩盘对香港金融机构形成系统性风险之类的谣言疑兵，连资金都不需要动用（也用不上，因为CNH对香港金融环境传导效应太弱）。建好空头仓位后，马上利用各种方法Bluff股市，包括真刀真枪和嘴炮。一旦恒指跌破8000，就会出现大量的不计成本的卖盘。这时候空头只需要轻松用这些卖盘回补自己的空头就可以。为什么要回补？因为这个头寸需要退出，最好的退出窗口是最虚弱的交易对手双手奉上的流动性。如果恋战不肯退出，会平添很多风险。严谨的Trader，每次在布局策略之初就会考虑退出窗口，尽可能利用市场资金实现利润。还是一句老话，谁都会买，但只有高手会卖。高手不一定是买在山脚卖在山顶，只是知道山腰哪个地方最肥沃而已。Trader 赚钱是要赚到量，不是赚到价。

所以，这是一个极为巧妙的Macro Trade. 虽然香港金融市场体量庞大实力雄厚，拥有海量资金的金管局坐镇中军帐，还可能得到中国人民银行相助，但是空头利用香港金融市场结构上的破绽，疑兵四出，搅得多头战战兢兢，甚至对中军都产生怀疑。等到多头阵脚大乱的时候，空头轻松包围了多头最孤立无援的一个营地，一击得手，攫取了最丰厚的一块利润，然后扬长而去。

参考资料：

1. Dennis Botman, Irineu de Carvalho Filho, W. Raphael Lam,The Curious Case of the Yen as a Safe Haven Currency: A Forensic Analysis, IMF Working Paper (2013)

2. Stefan Avdjiev, Wenxin Du, Catherine Koch, Hyun Song Shin, The dollar, bank leverage and the deviation from covered interest parity, BIS Working Paper(2016)

3. Fumihiko Arai, Yoshibumi Makabe, Yasunori Okawara and Teppei Nagano, Recent Trends in Cross-currency Basis, Bank of Japan Review(Sep 2016)

4. HKMA, The foreign exchange and derivatives markets in Hong Kong, HKMA Quarterly Bullentin(December 2016)

5. 熊猫笔记，20160606，20151117，20151230

宏观交易入门101| 第八课 尾声

告别??再见

听说，喜欢熊猫的小伙伴们，也在盼望着、盼望着更文的日子里，相继返回工位了...从“学习使我快乐”的口号，换到了“工作使我快乐”的鸡血，但，只要你是快乐的，就一切都好。

呃.....可是今天熊猫交易员似乎有点儿浅浅的忧伤，因为宏观交易入门系列课程更新即将完结??

是啊，随着宏观交易入门系列课程断断续续的更新，一不留神还是“刺溜”进入了尾声，从推出第一课《货币银行体系的资金和信贷》时，后台铺天盖地的留言“好难，看不懂”，到推出第四课《Cash Study: 货币基金改革如何隔山打牛》时，好多人兴奋的留言“哇哦，今天我竟然看懂了”，再到推出第六课《考试题参考答案》时，好多人欢呼“答案终于来了”并且热切的留下了自己的答案、意见和想法，而现在呢，只有阅读量在蹭蹭的往上走，大家似乎都做起了安静的读书人。算上这一篇，整个系列一共推出了八篇文章，虽然平时没能一一回复各位留下的疑问，解答所有的困惑，但是你们的心情和学习体验，我都一一记在了心里。我们不求文章篇篇惊艳，但是希望每一个系列的文章都不负卿望，让你学有所得。

走一段路，从迷茫荒唐走向清晰踏实；学一份习，从“不知”通往“知之”。宏观交易入门系列结束之后，还会有新的系列推出来，学习使我快乐，希望你也是。

今天??会谈谈自己对Macro Trade的一些看法。诸君阅读愉快！

宏观交易入门

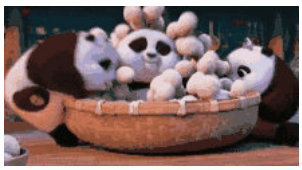
尾声

从2012年开始，虽然Marco Trader们一如既往的睥睨债券、外汇和商品界的Flow Trader同行，但是他们的光环正在飞速褪色。许多Macro Hedge Fund都出现了令人尴尬的亏损，而且一亏就是连续数年。大部分的Macro Hedge Fund都遇到了投资者赎回潮。许多极富盛名的Marco Hedge Fund，不愿意忍受投资人和监管的刁难，宁愿退回投资者的资金，转型为Family Office。与此同时，Macro Hedge Fund的领域突然出现了很多怪异的新人。 老Marcro Trader们应付完了难缠的投资人，回到办公室看到杂志上新出的对冲基金访谈，忍不住还要喷一下：“这都什么人，也来做Macro Fund?”

分析这些现象，同样应该结合Macro Hedge Fund 界的结构，还有宏观主题的变化。全球金融危机之后的Macro Hedge Fund大致可以分为三类，老炮、新枪和票友。最传统的，从分析经济体根本性的结构问题出发设计策略的，可以称为老炮。典型的老炮如Soros，策略深刻，出手狠辣。《Case Study: The Jan 2016 HKD Raid》篇就能看出老炮们的痕迹。老炮们往往三年不开张，开张吃三年，投资者如果没有对老炮们个人的高度信任，不容易跟老炮们长相厮守着不撤资。

新枪则往往出自银行交易部门。一些出类拔萃的交易员们从自己的领域向跨资产多策略发展时，发现Macro Trade非常适合自己的思维，于是转型Macro Hedge Fund. 新枪们一般会从自己的领域带一些固定的跨资产策略，能够做出与大市相关性很低的回报，因此颇得养老金等特别在乎分散化的投资者的青睐。可以通过《Case Study：隔山打牛的2a-7改革》和《Case Study：日元避险特性之宝宝心里苦但是宝宝不说》感受一下有代表性的新枪策略。

票友们则往往不是来自传统Macro Trade的领域，而是做证券选择类策略，比如Credit策略和Long/Short策略。他们挤到这个领域，主要是因为金融危机打破了很多常规思维，加之非常规货币政策造成的资产价格暴涨，让宏观环境是投资策略的基础这一观念在整个金融市场被强化到前所未有的高度。所以很多做行业策略的基金经理，也开始在策略中大量增加宏观内容，甚至干脆以宏观对冲基金的身份出现。不过，票友们的核心策略往往不是基于Macro Finance，只是加大了宏观作为一个变量的输入而已。毕竟，《Monetary Economics》开篇就提到，证券选择使用的工具与Macro Finance不同。



金融危机之后的非常规货币政策，以及投资者的变化，深刻的改变了Macro Hedge Fund 领域。在正常的货币环境下，《Monetary Economics》和《Macroeconomics》篇中提到的诸多货币传导、资金流动等因素，为宏观层面上各类资产提供了丰富交易策略，使得Macro Trader们借助熟悉的分析技巧，再应用套利、波动性等策略，能够做出与大类资产关联性很低的回报。但是，QE注入了大量流动性，为许多简单粗暴的策略比如Carry和Leverage提供了许多空间，也压缩了本应属于Macro Fund的跨资产套利空间。零利率和负利率引起的Risk Premium下降，把Volatility压制到罕有的地位，这又有把Macro Fund常用的Gamma策略抹掉了。在股票债券等蹭蹭蹭上涨的情况下，Macro Trader那些只比基础利率高个3%-5%收益率的策略，实在是寒碜。更倒霉的是金融危机之后监管和银行普遍收紧了对冲基金的杠杆率，这样想赚钱就更不容易了。

熊猫在前面多次强调“负债对资产的驱动”这个概念。在资产管理领域，投资者行为例如增资撤资、波动性要求、风格偏离容忍度等，熊猫认为是资产管理领域的负债（注意这个负债概念不是财务意义上的）。在对冲基金行业兴起之初，投资者多为富有的个人或者是颇具领先思维的机构，对金融市场有自己的理解，不受市场波动左右。这些投资人对老炮们的高度信任成就了Macro Hedge Fund的许多传奇。进入21世纪，对冲基金行业大爆发，大批新的机构投资者如养老金、大学捐赠基金甚至FoF都把对冲基金当成重要的投资工具。这批投资者最大的不同，是他们往往也有自己的负债（注意这个负债概念与上面相同，不是财务意义上的），加之资产管理研究的进展，提供了很多基金分析工具，所以这批投资人往往对波动性、回撤率等风险指标有一定的要求。而且，部分投资者还有很强的独立研究能力，有时候只是把对冲基金当成自己的宏观与资产配置观点的实现工具，对Macro Hedge Fund的风格偏离容忍度就更小。所以，虽然不少新投资人慕老炮之名而来，但是不能忍受老炮们桀骜不驯的P&L，加之非常规货币政策抹掉了很多经济体的结构性矛盾，让老炮们的传统策略打不到实处，所以新投资人走的也很快。新枪们略好一些，因为他们的传统策略往往强调稳定，但是他们这些稳定的策略，比如跨资产套利、Gamma策略等，也是受QE挤压最狠的。票友们业绩似乎最好看，但即使是票友们，也赶不上ETF等被动投资工具的光芒。如果所有的资产都在无差别上涨，为什么要交高额的2-20（对冲基金行规：2%固定管理费 + 20%业绩提成）给对冲基金，而不是买回报更高还几乎不收费的ETF？

Alpha 的消失，对整个主动资产管理行业都是很尴尬的事情，Macro Hedge Fund 尤其突出。

这些让Macro Hedge Fund不太舒服的变化，印证了金融市场的规律。亦即金融市场是所有行业中变化最快的，没有什么技巧和规律能够久混不衰。对Macro Trader，或者对所有的Trader，持续的寻找下一个宏观金矿，永远是职业中不可缺少的部分。

目前最具潜力的宏观金矿，应该是开放中的中国金融市场。虽然全球金融市场很早就极其重视中国经济对全球市场的影响，但往往只把中国当成一个经济变量而不是金融变量，因为过去数十年的金融抑制和资本管制，使得中国成为一个国际金融市场的被动价格承受者。这种被动，甚至能被动到一万亿外汇储备的时候嫌多、三万亿外汇储备的时候嫌少这种尴尬境界。随着利率市场化、汇率形成机制发展，中国应该有能力发展自己的定价机制，摆脱被动价格承受者的不利地位，以自己的利率、汇率和跨境资金流，为本国经济争取国际议价权，而不是陷于“我们的货币，你们的问题”这种状态。

这并不是一个宏大而不着边际的梦想，而是赤裸裸的现实利益问题。国际金融市场没有旁观者，如果不能得利，必然就是受损害的一方。在《Case Study：日元避险货币之宝宝心里苦但是宝宝不说》篇里，我们曾分析过日元的例子。由于日本金融机构在国际竞争中输掉了日元汇率和日本跨境资金流的话语权，导致日本经济和日本的海外投资面临一系列人傻钱多的囧境。如果中国金融市场没有一批实力强大的金融机构、没有一群勤奋刻苦基础扎实的Trader和Banker、人民币不能正常国际化，连日本这样的国际金融凯子状态中国都难以求到手，说不定还会沦落为EM常见的在暴饮暴食与奄奄一息之间游荡这种状态。

在国际金融市场走过一圈之后，我仍然相信中国是最有潜力的全球宏观金融市场。中国经济的韧性常在，而中国金融从业人员之勤奋聪明，在我从业过的所有金融市场中都是罕见的。许多在欧美金融市场功成名就的前辈，例如Capula的创始人霍焱，还有本人曾跟随过的一位传奇Trader，为中国Trader们树立了良好的学习榜样。相信我们不会等候太久，就能看到出身中国金融市场的全球宏观巨擘。

《熊猫笔记之宏观交易入门》全文完。

往期文章回顾

宏观交易101第一课丨货币银行体系的资金和信贷match

宏观交易101第二课丨跨境资金流动

宏观交易101第三课丨宏观交易入门（附试题）

宏观交易101第四课丨Cash Study：货币基金改革如何隔山打牛

宏观交易101第五课丨Trading Skills

宏观交易101第六课丨Monetary Economics 考试题参考答案

宏观交易101第七课丨Derivative Securities & Other Skills for Macro Trading

机器学习

机器学习系列：算法基础----Logistic 回归

logistic 分布

设X是连续随机变量，X服从logistic分布是指X具有下列分布函数和密度函数：

$$F(x) = P(x \le x) = \frac{1}{1+e^{-(x-\mu)/\gamma}}$$
$$f(x) = F'(x) = \frac{e^{-(x-\mu)/\gamma}}{\gamma(1+e^{-(x-\mu)/\gamma})^2}$$

其中，μμ为位置参数，γγ为形状参数。

f(x)f(x)与F(x)F(x)图像如下，其中分布函数是以(μ,12)(μ,12)为中心对阵，γγ越小曲线变化越快。

logistic 回归模型

二项logistic回归模型如下：

$$P(Y=1|x) = \frac{exp(w\cdot x+b)}{1+exp(w\cdot x+b)}$$
$$P(Y=0|x) = \frac{1}{1+exp(w\cdot x+b)}$$

其中，x∈Rnx∈Rn是输入，Y∈0,1Y∈0,1是输出，w称为权值向量，b称为偏置，w·xw·x为w和x的内积。

参数估计

假设：

P(Y=1|x)=π(x),P(Y=0|x)=1−π(x)P(Y=1|x)=π(x),P(Y=0|x)=1−π(x)

则似然函数为:

$$\prod_{i=1}^{N}[\pi(x_i)]^{y_i}[1-\pi(x_i)]^{1-y_i}$$

求对数似然函数：

$$\begin{aligned} L(w) &= \sum_{i=1}^{N}[y_i \log \pi(x_i) + (1-y_i)\log(1-\pi(x_i))] \\ &= \sum_{i=1}^{N}[y_i \log \frac{\pi(x_i)}{1-\pi(x_i)} + \log(1-\pi(x_i))] \end{aligned}$$

从而对L(w)L(w)求极大值，得到ww的估计值。

求极值的方法可以是梯度下降法，梯度上升法等。

概论¶

　在scikit-learn中，与逻辑回归有关的主要是这3个类。LogisticRegression， LogisticRegressionCV 和logistic_regression_path。其中LogisticRegression和LogisticRegressionCV的主要区别是LogisticRegressionCV使用了交叉验证来选择正则化系数C。而LogisticRegression需要自己每次指定一个正则化系数。除了交叉验证，以及选择正则化系数C以外， LogisticRegression和LogisticRegressionCV的使用方法基本相同。
　logistic_regression_path类则比较特殊，它拟合数据后，不能直接来做预测，只能为拟合数据选择合适逻辑回归的系数和正则化系数。主要是用在模型选择的时候。一般情况用不到这个类，所以后面不再讲述logistic_regression_path类。
　此外，scikit-learn里面有个容易让人误解的类RandomizedLogisticRegression,虽然名字里有逻辑回归的词，但是主要是用L1正则化的逻辑回归来做特征选择的，属于维度规约的算法类，不属于我们常说的分类算法的范畴。
　后面的讲解主要围绕LogisticRegression和LogisticRegressionCV中的重要参数的选择来来展开，这些参数的意义在这两个类中都是一样的。

```
sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,
```

```
sklearn.linear_model.LogisticRegressionCV(Cs=10, fit_intercept=True, cv=None, dual=False, penalty='l2', scoring=None, solver='lbfgs', tol=0.0001
```

```
logistic_regression_path(X, y, pos_class=None, Cs=10, fit_intercept=True, max_iter=100, tol=0.0001, verbose=0, solver='lbfgs', coef=None, copy=F
```

正则化选择参数：penalty ¶

　　LogisticRegression和LogisticRegressionCV默认就带了正则化项。penalty参数可选择的值为"l1"和"l2".分别对应L1的正则化和L2的正则化，默认是L2的正则化。

　　在调参时如果我们主要的目的只是为了解决过拟合，一般penalty选择L2正则化就够了。但是如果选择L2正则化发现还是过拟合，即预测效果差的时候，就可以考虑L1正则化。另外，如果模型的特征非常多，我们希望一些不重要的特征系数归零，从而让模型系数稀疏化的话，也可以使用L1正则化。

　　penalty参数的选择会影响我们损失函数优化算法的选择。即参数solver的选择，如果是L2正则化，那么4种可选的算法{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’}都可以选择。但是如果penalty是L1正则化的话，就只能选择‘liblinear’了。这是因为L1正则化的损失函数不是连续可导的，而{‘newton-cg’, ‘lbfgs’,‘sag’}这三种优化算法时都需要损失函数的一阶或者二阶连续导数。而‘liblinear’并没有这个依赖。

优化算法选择参数：solver ¶

　solver参数决定了我们对逻辑回归损失函数的优化方法，有4种算法可以选择，分别是：

a) liblinear：使用了开源的liblinear库实现，内部使用了坐标轴下降法来迭代优化损失函数。

b) lbfgs：拟牛顿法的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。

c) newton-cg：也是牛顿法家族的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。

d) sag：即随机平均梯度下降，是梯度下降法的变种，和普通梯度下降法的区别是每次迭代仅仅用一部分的样本来计算梯度，适合于样本数据多的时候，SAG是一种线性收敛算法，这个速度远比SGD快。关于SAG的理解，参考博文线性收敛的随机优化算法之 SAG、SVRG（随机梯度下降）

从上面的描述可以看出，newton-cg, lbfgs和sag这三种优化算法时都需要损失函数的一阶或者二阶连续导数，因此不能用于没有连续导数的L1正则化，只能用于L2正则化。而liblinear通吃L1正则化和L2正则化。

同时，sag每次仅仅使用了部分样本进行梯度迭代，所以当样本量少的时候不要选择它，而如果样本量非常大，比如大于10万，sag是第一选择。但是sag不能用于L1正则化，所以当你有大量的样本，又需要L1正则化的话就要自己做取舍了。要么通过对样本采样来降低样本量，要么回到L2正则化。

分类方式选择参数：multi_class ¶

　multi_class参数决定了我们分类方式的选择，有 ovr和multinomial两个值可以选择，默认是 ovr。

ovr即前面提到的one-vs-rest(OvR)，而multinomial即前面提到的many-vs-many(MvM)。如果是二元逻辑回归，ovr和multinomial并没有任何区别，区别主要在多元逻辑回归上。

OvR的思想很简单，无论你是多少元逻辑回归，我们都可以看做二元逻辑回归。具体做法是，对于第K类的分类决策，我们把所有第K类的样本作为正例，除了第K类样本以外的所有样本都作为负例，然后在上面做二元逻辑回归，得到第K类的分类模型。其他类的分类模型获得以此类推。

而MvM则相对复杂，这里举MvM的特例one-vs-one(OvO)作讲解。如果模型有T类，我们每次在所有的T类样本里面选择两类样本出来，不妨记为T1类和T2类，把所有的输出为T1和T2的样本放在一起，把T1作为正例，T2作为负例，进行二元逻辑回归，得到模型参数。我们一共需要T(T-1)/2次分类。

从上面的描述可以看出OvR相对简单，但分类效果相对略差（这里指大多数样本分布情况，某些样本分布下OvR可能更好）。而MvM分类相对精确，但是分类速度没有OvR快。

如果选择了ovr，则4种损失函数的优化方法liblinear，newton-cg, lbfgs和sag都可以选择。但是如果选择了multinomial,则只能选择newton-cg, lbfgs和sag了。

类型权重参数： class_weight ¶

　class_weight参数用于标示分类模型中各种类型的权重，可以不输入，即不考虑权重，或者说所有类型的权重一样。如果选择输入的话，可以选择balanced让类库自己计算类型权重，或者我们自己输入各个类型的权重，比如对于0,1的二元模型，我们可以定义class_weight={0:0.9, 1:0.1}，这样类型0的权重为90%，而类型1的权重为10%。

如果class_weight选择balanced，那么类库会根据训练样本量来计算权重。某种类型样本量越多，则权重越低，样本量越少，则权重越高。

sklearn的官方文档中，当class_weight为balanced时，类权重计算方法如下：

n_samples / (n_classes * np.bincount(y))，n_samples为样本数，n_classes为类别数量，np.bincount(y)会输出每个类的样本数，例如y=[1,0,0,1,1],则np.bincount(y)=[2,3]

那么class_weight有什么作用呢？在分类模型中，我们经常会遇到两类问题：

第一种是误分类的代价很高。比如对合法用户和非法用户进行分类，将非法用户分类为合法用户的代价很高，我们宁愿将合法用户分类为非法用户，这时可以人工再甄别，但是却不愿将非法用户分类为合法用户。这时，我们可以适当提高非法用户的权重。

第二种是样本是高度失衡的，比如我们有合法用户和非法用户的二元样本数据10000条，里面合法用户有9995条，非法用户只有5条，如果我们不考虑权重，则我们可以将所有的测试集都预测为合法用户，这样预测准确率理论上有99.95%，但是却没有任何意义。这时，我们可以选择balanced，让类库自动提高非法用户样本的权重。

提高了某种分类的权重，相比不考虑权重，会有更多的样本分类划分到高权重的类别，从而可以解决上面两类问题。

样本权重参数： sample_weight ¶

　调节样本权重的方法有两种，第一种是在class_weight使用balanced。第二种是在调用fit函数时，通过sample_weight来自己调节每个样本权重。

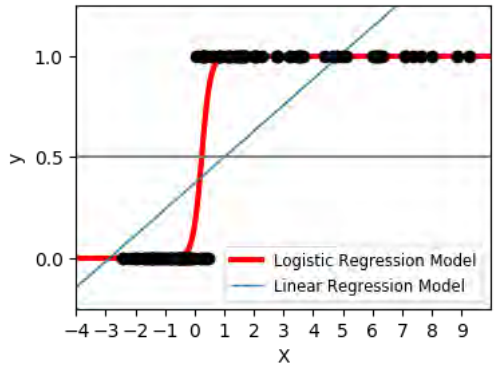
在scikit-learn做逻辑回归时，如果上面两种方法都用到了，那么样本的真正权重是class_weight*sample_weight.

实例（线性回归和Logistic 回归对比）¶

In [1]:

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

from sklearn import linear_model

# this is our test set, it's just a straight line with some
# Gaussian noise
xmin, xmax = -5, 5
```

```
n_samples = 100
np.random.seed(0)
X = np.random.normal(size=n_samples)
y = (X > 0).astype(np.float)
X[X > 0] *= 4
X += .3 * np.random.normal(size=n_samples)

X = X[:, np.newaxis]
# run the classifier
clf = linear_model.LogisticRegression(C=1e5)
clf.fit(X, y)

# and plot the result
plt.figure(1, figsize=(4, 3))
plt.clf()
plt.scatter(X.ravel(), y, color='black', zorder=20)
X_test = np.linspace(-5, 10, 300)


def model(x):
    return 1 / (1 + np.exp(-x))
loss = model(X_test * clf.coef_ + clf.intercept_).ravel()
plt.plot(X_test, loss, color='red', linewidth=3)

ols = linear_model.LinearRegression()
ols.fit(X, y)
plt.plot(X_test, ols.coef_ * X_test + ols.intercept_, linewidth=1)
plt.axhline(.5, color='.5')

plt.ylabel('y')
plt.xlabel('X')
plt.xticks(range(-5, 10))
plt.yticks([0, 0.5, 1])
plt.ylim(-.25, 1.25)
plt.xlim(-4, 10)
plt.legend(('Logistic Regression Model', 'Linear Regression Model'),
           loc="lower right", fontsize='small')
plt.show()
```



非常简单的一个实例，单纯比较了线性回归和logistic回归的一些效果。

小运用，多分类情况¶

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model, datasets
import numpy as np
```

In [3]:

```
# 导入数据，还是我们经典的iris
iris=pd.read_csv('../Iris.csv')
iris.head()
```

Out[3]:

	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa

	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	class
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [4]:

```
X = np.array(iris.iloc[:, :2])
Y = np.array(iris['class'])
Y[Y == 'Iris-setosa'] = 0
Y[Y == 'Iris-versicolor'] = 1
Y[Y == 'Iris-virginica'] = 2
Y = np.array(Y,dtype = np.int64)
Y
```

Out[4]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [5]:

```
h = .02  # step size in the mesh

logreg = linear_model.LogisticRegression(C=1e5)

# we create an instance of Neighbours Classifier and fit the data.
logreg.fit(X, Y)
```

Out[5]:

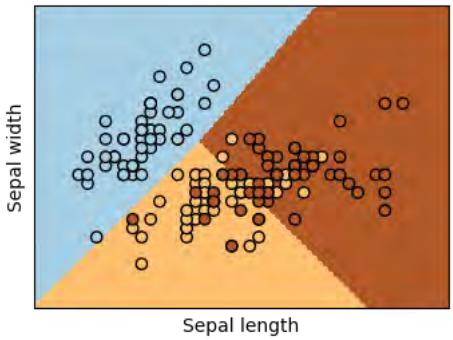
```
LogisticRegression(C=100000.0, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [6]:

```
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])
```

In [7]:

```
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```



机器学习系列：算法基础----K-Means

k-means 算法接受参数 k ； 然后将事先输入的n个数据对象划分为 k个聚类以便使得所获得的聚类满足：同一聚类中的对象相似度较高；而不同聚类中的对象相似度较小。

聚类相似度是利用各聚类中对象的均值所获得一个“中心对象”（引力中心）来进行计算的。 K-means算法是最为经典的基于划分的聚类方法，是十大经典数据挖掘算法之一。K-means算法的基本思想是：以空间中k个点为中心进行聚类，对最靠近他们的对象归类。通过迭代的方法，逐次更新各聚类中心的值，直至得到最好的聚类结果。

KNN与K-Means 对比

KNN	K-Means
目的是为了确定一个点的分类	目的是为了将一系列点集分成k类
KNN是分类算法	K-Means是聚类算法
监督学习，分类目标事先已知	非监督学习，将相似数据归到一起从而得到分类，没有外部分类
训练数据集有label，已经是完全正确的数据	训练数据集无label，是杂乱无章的，经过聚类后才变得有点顺序，先无序，后有序
没有明显的前期训练过程，属于memory-based learning	有明显的前期训练过程
K的含义：“k”是用来计算的相邻数据数。来了一个样本x，要给它分类，即求出它的y，就从数据集中，在x	K的含义：“k”是类的数目。K是人工固定好的数字，假设数据集合可以分为K个簇，由于是依靠人工定好，需要

附近找离它最近的K个数据点，这K个数据点，类别c占的个数最多，就把x的label设为c	一点先验知识
K值确定后每次结果固定	K值确定后每次结果可能不同，从 n 个数据对象任意选择 k 个对象作为初始聚类中心，随机性对结果影响较大
时间复杂度：O（n）	时间复杂度：O(n*k*t)，t为迭代次数

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set()
```

简介¶

K-Means是用于无监督聚类的算法：即，仅基于数据属性（而不是标签）在数据中找到各自的类。

K均值是一个相对容易理解的算法。它搜索作为其中的点的平均值的聚类中心，使得每个点最接近它被分配给的聚类中心。让我们看看KMeans如何在我们之前看过的简单集群上运行。

概述¶

在scikit-learn中，包括两个K-Means的算法，一个是传统的K-Means算法，对应的类是KMeans。另一个是基于采样的Mini Batch K- Means算法，对应的类是MiniBatchKMeans。一般来说，使用K-Means的算法调参是比较简单的。
用KMeans类的话，一般要注意的仅仅就是k值的选择，即参数n_clusters；如果是用MiniBatchKMeans的话，也仅仅多了需要注意调参的参数batch_size，即我们的Mini Batch的大小。

当然KMeans类和MiniBatchKMeans类可以选择的参数还有不少，但是大多不需要怎么去调参。下面我们就看看KMeans类和MiniBatchKMeans类的一些主要参数。

Keans 类的主要参数¶

1) n_clusters: 即我们的k值，一般需要多试一些值以获得较好的聚类效果。k值好坏的评估标准在下面会讲。

2）max_iter： 最大的迭代次数，一般如果是凸数据集的话可以不管这个值，如果数据集不是凸的，可能很难收敛，此时可以指定最大的迭代次数让算法可以及时退出循环。

3）n_init：用不同的初始化质心运行算法的次数。由于K-Means是结果受初始值影响的局部最优的迭代算法，因此需要多跑几次以选择一个较好的聚类效果，默认是10，一般不需要改。如果你的k值较大，则可以适当增大这个值。

4）init： 即初始值选择的方式，可以为完全随机选择'random',优化过的'k-means++'或者自己指定初始化的k个质心。一般建议使用默认的'k-means++'。

5）algorithm：有“auto”, “full” or “elkan”三种选择。"full"就是我们传统的K-Means算法， “elkan”是我们原理篇讲的elkan K-Means算法。默认的"auto"则会根据数据值是否是稀疏的，来决定如何选择"full"和“elkan”。一般数据是稠密的，那么就是 “elkan”，否则就是"full"。一般来说建议直接用默认的"auto"

MiniBatchKMeans 类主要参数¶

1) n_clusters: 即我们的k值，和KMeans类的n_clusters意义一样。

2）max_iter：最大的迭代次数， 和KMeans类的max_iter意义一样。

3）n_init：用不同的初始化质心运行算法的次数。这里和KMeans类意义稍有不同，KMeans类里的n_init是用同样的训练集数据来跑 不同的初始化质心从而运行算法。而MiniBatchKMeans类的n_init则是每次用不一样的采样数据集来跑不同的初始化质心运行算法。

4）batch_size：即用来跑Mini Batch KMeans算法的采样集的大小，默认是100.如果发现数据集的类别较多或者噪音点较多，需要增加这个值以达到较好的聚类效果。

5）init： 即初始值选择的方式，和KMeans类的init意义一样。

6）init_size: 用来做质心初始值候选的样本个数，默认是batch_size的3倍，一般用默认值就可以了。

7）reassignment_ratio: 某个类别质心被重新赋值的最大次数比例，这个和max_iter一样是为了控制算法运行时间的。这个比例是占样本总数的比例，乘以样本总数就得到了每个类别质心可以重新赋值的次数。如果取值较高的话算法收敛时间可能会增加，尤其是那些暂时拥有样本数较少的质心。默认是0.01。如果数据量不是超大的话，比如1w以下，建议使用默认值。如果数据量超过1w，类别又比较多，可能需要适当减少这个比例值。具体要根据训练集来决定。

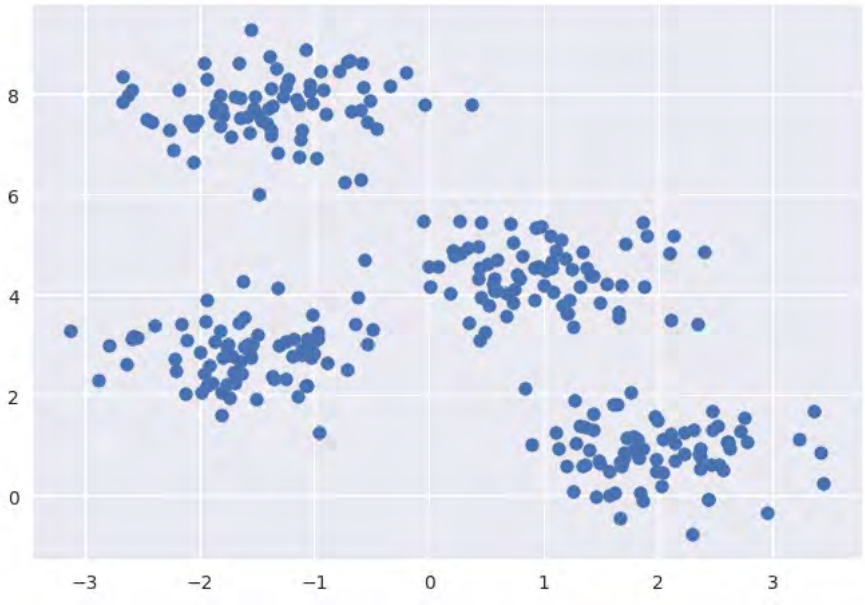
8）max_no_improvement：即连续多少个Mini Batch没有改善聚类效果的话，就停止算法， 和reassignment_ratio， max_iter一样是为了控制算法运行时间的。默认是10.一般用默认值就足够了。

范例¶

In [2]:

```
# 生成数据
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=300, centers=4,
                  random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], s=50);
```

```
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```

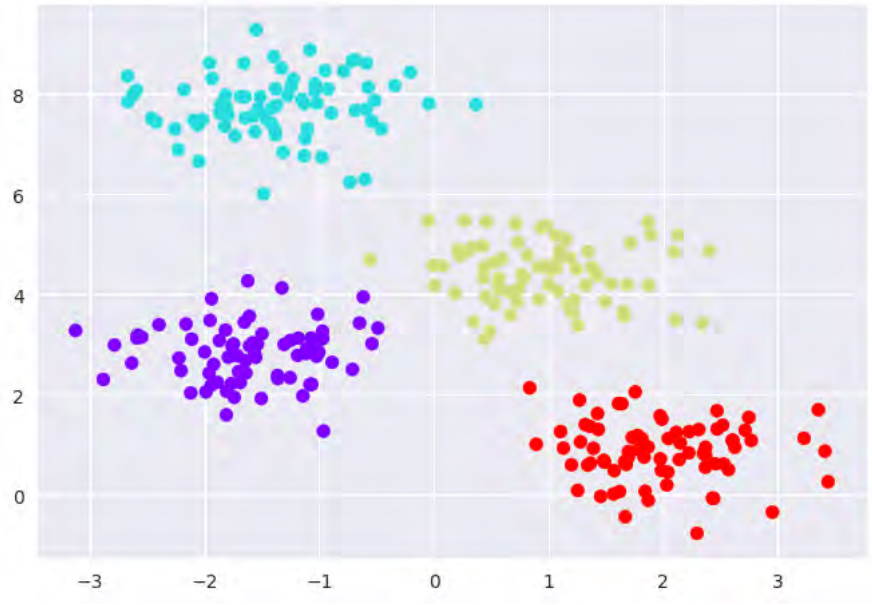


肉眼可见，容易挑出四个类。但是，如果要对数据进行详细的分类，则搜索空间将是点数的指数。幸运的是，有一个众所周知的期望最大化画算法（EM算法）来帮助我们解决问题。

In [3]:

```
from sklearn.cluster import KMeans
est = KMeans(4)  # 4 clusters
est.fit(X)
y_kmeans = est.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='rainbow');
```

```
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```



小运用（数字识别）¶

对于接近实际的例子，导入一些数字的数据。在这里，我们将使用KMeans自动聚集64个维度的数据，然后查看聚类中心来看看算法找到了什么。

In [6]:

```
from sklearn.datasets import load_digits
digits = load_digits()
```

In [7]:

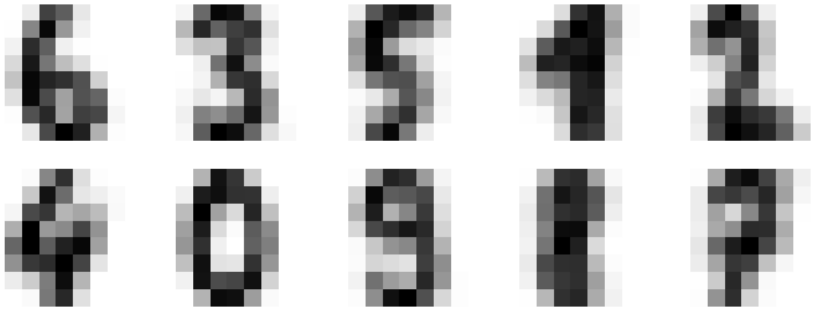
```
est = KMeans(n_clusters=10)
clusters = est.fit_predict(digits.data)
est.cluster_centers_.shape
```

Out[7]:

```
(10, 64)
```

In [8]:

```
fig = plt.figure(figsize=(8, 3))
for i in range(10):
    ax = fig.add_subplot(2, 5, 1 + i, xticks=[], yticks=[])
    ax.imshow(est.cluster_centers_[i].reshape((8, 8)), cmap=plt.cm.binary)
```



我们看到，即使没有标签，KMeans能够找到识别数字的手段

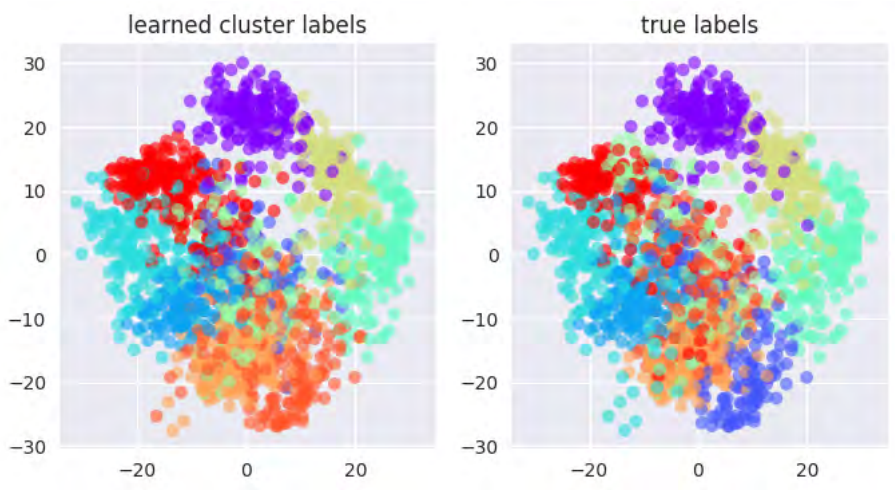
In [9]:

```
from scipy.stats import mode

labels = np.zeros_like(clusters)
for i in range(10):
    mask = (clusters == i)
    labels[mask] = mode(digits.target[mask])[0]
```

In [10]:

```
from sklearn.decomposition import PCA

X = PCA(2).fit_transform(digits.data)

kwargs = dict(cmap = plt.cm.get_cmap('rainbow', 10),
              edgecolor='none', alpha=0.6)
fig, ax = plt.subplots(1, 2, figsize=(8, 4))
ax[0].scatter(X[:, 0], X[:, 1], c=labels, **kwargs)
ax[0].set_title('learned cluster labels')

ax[1].scatter(X[:, 0], X[:, 1], c=digits.target, **kwargs)
ax[1].set_title('true labels');
```

```
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```



In [11]:

```
from sklearn.metrics import accuracy_score
accuracy_score(digits.target, labels)
```

Out[11]:

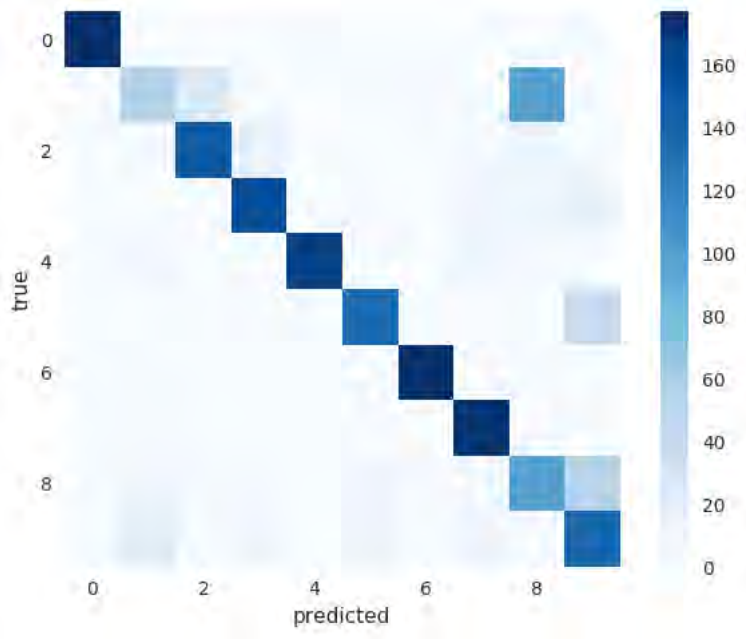
```
0.79298831385642743
```

In [12]:

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(digits.target, labels))

plt.imshow(confusion_matrix(digits.target, labels),
           cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.grid(False)
plt.ylabel('true')
plt.xlabel('predicted');
```

```
[[177   0   0   0   1   0   0   0   0   0]
 [  0  55  24   1   0   1   2   0  99   0]
 [  1   2 148  13   0   0   0   3   8   2]
 [  0   0   0 155   0   2   0   7   7  12]
 [  0   5   0   0 164   0   0   8   4   0]
 [  0   0   0   1   2 136   1   0   0  42]
 [  1   1   0   0   0   0 177   0   2   0]
 [  0   2   0   0   0   1   0 174   2   0]
 [  0   6   3   2   0   6   2   3 100  52]
 [  0  20   0   6   0   7   0   7   1 139]]
```

```
/srv/env/lib64/python3.4/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: Font family ['sans-serif'] not found.
  (prop.get_family(), self.defaultFamily[fontext]))
```



同样，对于一个完全无监督的估计器，能够达到80的精度。

In [ ]:

机器学习系列：算法基础----决策树

决策树参数

根据官网提供的信息，这里进行了简单的翻译。请点击：官方地址

criterion:string类型，可选（默认为"gini"）
　衡量分类的质量。支持的标准有"gini"代表的是Gini impurity(不纯度)与"entropy"代表的是information gain（信息增益）。

splitter:string类型，可选（默认为"best"）
　一种用来在节点中选择分类的策略。支持的策略有"best"，选择最好的分类，"random"选择最好的随机分类。

max_features:int,float,string or None 可选（默认为None）
　在进行分类时需要考虑的特征数。
　1.如果是int，在每次分类是都要考虑max_features个特征。
　2.如果是float,那么max_features是一个百分率并且分类时需要考虑的特征数是int(max_features*n_features,其中n_features是训练完成时发特征数)。
　3.如果是auto,max_features=sqrt(n_features)
　4.如果是sqrt,max_features=sqrt(n_features)
　5.如果是log2,max_features=log2(n_features)
　6.如果是None，max_features=n_features
　注意：至少找到一个样本点有效的被分类时，搜索分类才会停止。

max_depth:int or None,可选（默认为"None"）
　表示树的最大深度。如果是"None",则节点会一直扩展直到所有的叶子都是纯的或者所有的叶子节点都包含少于min_samples_split个样本点。忽视max_leaf_nodes是不是为None。

min_samples_split:int,float,可选（默认为2）

　区分一个内部节点需要的最少的样本数。
　1.如果是int，将其最为最小的样本数。
　2.如果是float，min_samples_split是一个百分率并且ceil(min_samples_split*n_samples)是每个分类需要的样本数。ceil是取大于或等于指定表达式的最小整数。

min_samples_leaf:int,float,可选（默认为1）
　一个叶节点所需要的最小样本数：
　1.如果是int，则其为最小样本数
　2.如果是float，则它是一个百分率并且ceil(min_samples_leaf*n_samples)是每个节点所需的样本数。

min_weight_fraction_leaf:float,可选（默认为0）
　一个叶节点的输入样本所需要的最小的加权分数。

max_leaf_nodes:int,None 可选（默认为None）
　在最优方法中使用max_leaf_nodes构建一个树。最好的节点是在杂质相对减少。如果是None则对叶节点的数目没有限制。如果不是None则不考虑max_depth.

class_weight:dict,list of dicts,"Banlanced" or None,可选（默认为None）
　表示在表{class_label:weight}中的类的关联权值。如果没有指定，所有类的权值都为1。对于多输出问题，一列字典的顺序可以与一列y的次序相同。
　"balanced"模型使用y的值去自动适应权值，并且是以输入数据中类的频率的反比例。如：n_samples/(n_classes*np.bincount(y))。
　对于多输出，每列y的权值都会想乘。
　如果sample_weight已经指定了，这些权值将于samples以合适的方法相乘。

random_state:int,RandomState instance or None
　如果是int,random_state 是随机数字发生器的种子；如果是RandomState，random_state是随机数字发生器，如果是None，随机数字发生器是np.random使用的RandomState instance.

persort:bool,可选（默认为False）
　是否预分类数据以加速训练时最好分类的查找。在有大数据集的决策树中，如果设为true可能会减慢训练的过程。当使用一个小数据集或者一个深度受限的决策树中，可以减速训练的过程。

范例¶

In [1]:

```
# 导入包和数据
import numpy as np
import pandas as pd
from pandas import DataFrame,Series
import datetime
from io import BytesIO
from sklearn import tree
```

In [2]:

```
iris=pd.read_csv('../Iris.csv')
iris.head()
```

Out[2]:

	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [3]:

```
# 创建数据
X_train = np.array(iris.iloc[:120,:4],dtype = np.float64)
X_test = np.array(iris.iloc[120:,:4],dtype = np.float64)
y_train = np.array(iris.iloc[:120,4:]).flatten()
y_test = np.array(iris.iloc[120:,4:]).flatten()
```

In [4]:

```
# 创建训练模型
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
```

In [5]:

```
clf.predict(X_test)
```

Out[5]:

```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica'], dtype=object)
```

In [6]:

```
# 计算评分
clf.score(X_test,y_test)
```

Out[6]:

```
0.73333333333333328
```

多分类AdaBoosted 决策树¶

这个例子再现了Zhu等人[1]的图1，并且示出了AdaBoosted算法可以多分类问题的精度。分类数据集通过取十维标准正态分布并且定义由嵌套的同心十维球分离的三个类别来构建，使得在每个类别中大致相等数量的样本。 比较SAMME和SAMME.R [1]算法的性能。 SAMME.R使用概率估计来更新加性模型，而SAMME仅使用分类。如示例所示，SAMME.R算法通常比SAMME算法收敛得更快，以较少的迭代实现较低的测试误差。在每次增强迭代之后，每个算法在测试集上的误差在左边示出，每个树的测试集上的分类误差在中间示出，并且每个树的增强权重在右边示出。所有树在SAMME.R算法中具有一个权重，因此未示出。

In [7]:

```
from sklearn.externals.six.moves import zip
import matplotlib.pyplot as plt
from sklearn.datasets import make_gaussian_quantiles
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

# 载入数据
X, y = make_gaussian_quantiles(n_samples=13000, n_features=10,n_classes=3, random_state=1)
n_split = 3000
X_train, X_test = X[:n_split], X[n_split:]
y_train, y_test = y[:n_split], y[n_split:]
X_train.shape
```

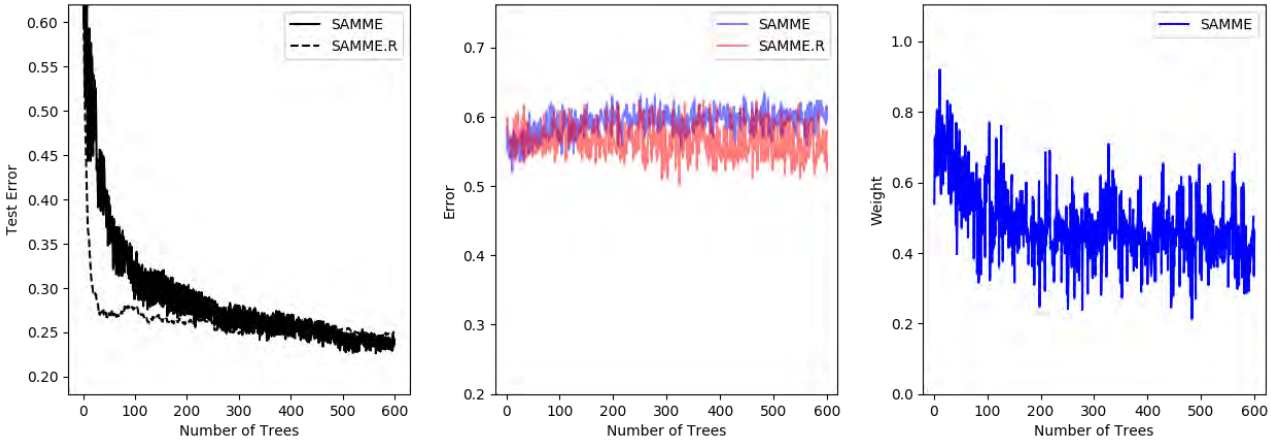
Out[7]:

```
(3000, 10)
```

In [8]:

```
bdt_real = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=2),
    n_estimators=600,
    learning_rate=1)

bdt_discrete = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=2),
    n_estimators=600,
    learning_rate=1.5,
    algorithm="SAMME")

bdt_real.fit(X_train, y_train)
bdt_discrete.fit(X_train, y_train)

real_test_errors = []
discrete_test_errors = []

for real_test_predict, discrete_train_predict in zip(
        bdt_real.staged_predict(X_test), bdt_discrete.staged_predict(X_test)):
    real_test_errors.append(
        1. - accuracy_score(real_test_predict, y_test))
    discrete_test_errors.append(
        1. - accuracy_score(discrete_train_predict, y_test))

n_trees_discrete = len(bdt_discrete)
n_trees_real = len(bdt_real)

# Boosting might terminate early, but the following arrays are always
```

```
# n_estimators long. We crop them to the actual number of trees here:
discrete_estimator_errors = bdt_discrete.estimator_errors_[:n_trees_discrete]
real_estimator_errors = bdt_real.estimator_errors_[:n_trees_real]
discrete_estimator_weights = bdt_discrete.estimator_weights_[:n_trees_discrete]

plt.figure(figsize=(15, 5))

plt.subplot(131)
plt.plot(range(1, n_trees_discrete + 1),
         discrete_test_errors, c='black', label='SAMME')
plt.plot(range(1, n_trees_real + 1),
         real_test_errors, c='black',
         linestyle='dashed', label='SAMME.R')
plt.legend()
plt.ylim(0.18, 0.62)
plt.ylabel('Test Error')
plt.xlabel('Number of Trees')

plt.subplot(132)
plt.plot(range(1, n_trees_discrete + 1), discrete_estimator_errors,
         "b", label='SAMME', alpha=.5)
plt.plot(range(1, n_trees_real + 1), real_estimator_errors,
         "r", label='SAMME.R', alpha=.5)
plt.legend()
plt.ylabel('Error')
plt.xlabel('Number of Trees')
plt.ylim((.2,
         max(real_estimator_errors.max(),
             discrete_estimator_errors.max()) * 1.2))
plt.xlim((-20, len(bdt_discrete) + 20))

plt.subplot(133)
plt.plot(range(1, n_trees_discrete + 1), discrete_estimator_weights,
         "b", label='SAMME')
plt.legend()
plt.ylabel('Weight')
plt.xlabel('Number of Trees')
plt.ylim((0, discrete_estimator_weights.max() * 1.2))
plt.xlim((-20, n_trees_discrete + 20))

# prevent overlapping y-axis labels
plt.subplots_adjust(wspace=0.25)
plt.show()
```



In [ ]:

```
```

机器学习系列：进阶算法----SVM

支持向量机，因其英文名为support vector machine，故一般简称SVM，它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，其学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。也等价于正则化的合页损失函数的最小化问题，支持向量机的学习算法是求解凸二次规划的最优化算法。

SVM学习方法包含构建由简至繁的模型：线性可分向量机、线性支持向量机及非线性支持向量机，简单模型是复杂模型的特殊情况。当训练数据可分时，通过硬间隔最大化学习一个线性分类器，即线性可分支持向量机；当数据近似线性可分时，通过软间隔最大化学习一个线性分类器，即线性支持向量机；当数据线性不可分时，通过核技巧及软间隔最大化学习非线性支持向量机。

线性可分

对于给定一些数据点，它们分别属于两个不同的类，可以找到一条直线（二维的）或超平面（多维的）将两类数据点完全分割开来，称之为线性可分。直线（超平面）的定义：

$$w^T x + b = 0$$

1 线性可分支持向量机：

对给定线性可分训练数据集，通过间隔最大化或等价地求解相应的凸二次规划问题学习得到的分离超平面为：

$$w^T x + b = 0$$

以及相应的决策函数 f(x) = sign(w* x + b*)，称为线性可分支持向量机。支持向量机学习的基本想法是求解能够正确划分训练数据集并且几何间隔最大的分离超平面，这里的间隔最大化又称为硬间隔最大化。 我们可以把这样的问题抽象称为如下的数学表达式：

$$\max_{w,b} \frac{\hat{\gamma}}{||w||}$$

$$s.t.\ y_i\left(w \cdot x_i + b\right) \geq \hat{\gamma}$$

然而，函数间隔$\hat{\gamma}$的取值并不影响最优化问题的解，我们可以取$\hat{\gamma} = 1$。则上述的优化问题就可以转化为：

$$\max_{w,b} \frac{1}{||w||}$$

$$s.t.\ y_i\left(w \cdot x_i + b\right) - 1 \geq 0$$

可以将上述的最大化问题转化为最小化问题：

$$\min_{w,b} \frac{1}{2}||w||^2$$

$$s.t.\ y_i\left(w \cdot x_i + b\right) - 1 \geq 0$$

这样的问题是一个凸二次规划的问题。在线性可分情况下，训练数据集的样本点中的分离超平面距离最近的样本点的事例称为支持向量，即满足：

$$y_i\left(w \cdot x_i + b\right) - 1 = 0$$

1.1 函数间隔和几何间隔

在超平面$w \cdot x + b = 0$确定的情况下，$|w \cdot x + b|$可以相对地表示点x距离超平面的远近。对于两类分类问题，如果$w \cdot x + b > 0$，则x的类别被判定为1；否则判定为-1。所以如果$y(w \cdot x + b) > 0$，则认为的x分类结果是正确的，否则是错误的。且$y(w \cdot x + b)$的值越大，分类结果的确信度越大。反之亦然。所以样本点(x_i, y_i)与超平面之间的函数间隔定义为$\gamma_i = y_i(w \cdot x_i + b)$。

但是该定义存在问题：即X和y同时缩小或放大M倍后，超平面并没有变化，但是函数间隔却变化了。所以，需要将w的大小固定，如$||w|| = 1$，使得函数间隔固定。这时的间隔也就是几何间隔 。几何间隔的定义如下

$$\gamma_i = y_i(\frac{w}{||w||} \cdot x_i + \frac{b}{||w||})$$

实际上，几何间隔就是点到超平面的距离。想像下中学学习的点(x_i, y_i)到直线$ax + by + c = 0$的距离公式：

$$d(x_i, y_i) = \frac{|ax_i + by_i + c|}{\sqrt{a^2 + b^2}}$$

所以在二维空间中，几何间隔就是点到直线的距离。在三维及以上空间中，就是点到超平面的距离。而函数距离，就是上述距离公式中的分子，即未归一化的距离。定义训练集到超平面的最小几何间隔是

$$\gamma = min_{i=1,\ldots,n}\gamma_i$$

SVM训练分类器的方法是寻找到超平面，使正负样本在超平面的两侧，且样本到超平面的几何间隔最大。所以SVM可以表述为求解下列优化问题

$$\underset{w,b}{max} \quad \gamma$$

$$s.t. \quad y_i(\frac{w}{||w||} \cdot x_i + \frac{b}{||w||}) \geq \gamma$$

1.2 间隔最大化

间隔最大化的直观解释是：对训练集找到几何间隔最大的超平面意味着以充分大的确信度对训练数据进行分析，在满足将不同类别数据点分开的情况下，而且对最难分的数据点（离超平面最近的点）也有足够大的确信度将他们分开。面最近的点）也有足够大的确信度将他们分开。

1.3 对偶问题

为了求解线性可分支持向量机的最优化问题，可以将它作为原始最优化问题，应用拉格朗日对偶性，通过求解对偶问题得到原始问题的最优解，这就是线性可分支持向量机的对偶算法。这样做的优点，一是对偶问题往往更加容易求解：而是自然引入核函数，进而推广到非线性分类问题。

对于上述的带约束的优化问题，我们可以引进拉格朗日函数来解决。这样，原始的问题就转化成一个极小极大问题：

$$\min_{w,b}\max_{\alpha} L(w,b,\alpha)$$

再通过拉格朗日函数的对偶性，将上述的极小极大问题转换成一个极大极小问题：

$$\max_{\alpha}\min_{w,b} L(w,b,\alpha)$$

此时，我们先求$\min_{w,b} L(w,b,\alpha)$。

将拉格朗日函数$L(w,b,\alpha)$分别对w和b求偏导，并令其为0，则为$\frac{\partial L(w,b,\alpha)}{\partial w} = w - \sum_{i=1}^{N}\alpha_i y_i x_i = 0$，$\frac{\partial L(w,b,\alpha)}{\partial b} = \sum_{i=1}^{N}\alpha_i y_i = 0$，可得：$w = \sum_{i=1}^{N}\alpha_i y_i x_i$，$\sum_{i=1}^{N}\alpha_i y_i = 0$。将上面两个等式带入拉格朗日函数$L(w,b,\alpha)$，得再求$\min_{w,b} L(w,b,\alpha)$，再求$\min_{w,b} L(w,b,\alpha)$对$\alpha$的极大，即：$s.t.\ \sum_{i=1}^{N}\alpha_i y_i = 0,\ \alpha_i \geq 0$。将这样的最大化问题转化为最小化问题，即为$s.t.\ \sum_{i=1}^{N}\alpha_i y_i = 0,\ \alpha_i \geq 0$。

根据拉格朗日对偶性，通过对偶函数的最优解即可以求出原始函数的最优解：$w^* = \sum_{i=1}^{N}\alpha_i^* y_i x_i$，$b^* = y_i - \sum_{i=1}^{N}\alpha_i^* y_i (x_i \cdot x_j)$。其中，下标$\alpha_j^* > 0$的样本。这里使得$\alpha_j^* > 0$的样本也称为支撑向量，与上述的满足$y_i(w \cdot x_i + b) - 1 = 0$的样本本质上是一样的。

1.4 线性可分支持向量机的步骤

1、构造带约束的优化问题：

$$s.t.\ \sum_{i=1}^{N}\alpha_i y_i = 0,\ \alpha_i \geq 0$$

2、计算原始问题的最优解：

$$w^* = \sum_{i=1}^{N}\alpha_i^* y_i x_i$$

$$b^* = y_j - \sum_{i=1}^{N}\alpha_i^* y_i (x_i \cdot x_j)$$

3、求分离超平面：

$w^* \cdot x + b^* = 0$，得到分类决策函数：$f(x) = sign(w^* \cdot x + b^*)$

2 线性支持向量机：

线性可分问题的支持向量机学习方法，对线性不可分训练数据不使用。我们需要修改硬间隔最大化，并使其成为软间隔最大化。线性支持向量机是针对线性不可分的数据集的，这样的数据集可以通过近似可分的方法实现分类。对于这样的数据集，类似线性可分支持向量机，通过求解对应的凸二次规划问题，也同样求得分离超平面

$w^* \cdot x + b^* = 0$，以及相应的分类决策函数$f(x) = sign(w^* \cdot x + b^*)$。

线性支持向量机与线性可分支持向量机最大的不同就是在处理的问题上，线性可分支持向量机处理的是严格线性可分的数据集，而线性支持向量机处理的是线性不可分的数据集，然而，在基本的原理上他们却有着想通之处。这里的线性不可分是指数据集中存在某些点不能满足线性可分支持向量机的约束条件：$y_i(w \cdot x_i + b) - 1 \geq 0$。

具体来讲，对于特征空间上的训练数据集T，且T不是线性可分的，即存在某些特异点不满足$y_i(w \cdot x_i + b) - 1 \geq 0$的约束条件，若将这些特异点去除，那么剩下的数据点是线性可分的，由此可见，线性可分支持向量机是线性支持向量机的特殊情况。为了解决这样的问题，对每个样本点(x_i, y_i)引入一个松弛变量ξ_i，且$\xi_i \geq 0$，则上述的约束条件被放宽，即：$y_i(w \cdot x_i + b) \geq 1 - \xi_i$。此时，此时目标函数变为：$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i$。其中$C$称为惩罚参数，且$C > 0$。在线性支持向量机中加入了惩罚项，与线性可分支持向量的应间隔最大化相对应，在线性支持向量机中称为软间隔最大化。

2.1 线性支持向量机的原理

由上所述，我们得到线性支持向量机的原始问题：$\min_{w,b,\xi}\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i$。接下来的问题就变成如何求解这样一个最优化问题(称为原始问题)。引入拉格朗日函数，其中$\alpha_i \geq 0,\ u \geq 0$。此时，原始问题即变成$\min_{w,b,\xi}\max_{\alpha,u} L(w,b,\xi,\alpha,u)$。利用拉格朗日函数的对偶性，将问题变成一个极大极小优化问题：$\max_{\alpha,u}\min_{w,b,\xi} L(w,b,\xi,\alpha,u)$。

首先求解$\min_{w,b,\xi} L(w,b,\xi,\alpha,u)$，将拉格朗日函数分别对$w, b, \xi$求偏导，并令其为0，即为：

$$\begin{cases} w = \sum_{i=1}^{N} \alpha_i y_i x_i \\ \sum_{i=1}^{N} \alpha_i y_i = 0 \\ C - \alpha_i - u_i = 0 \end{cases}$$

第二步，求 $\max\limits_{\alpha,u} \min\limits_{w,b,\xi} L(w,b,\xi,\alpha,u)$，即求：$\begin{matrix} \sum_{i=1}^{N} \alpha_i y_i = 0 \\ s.t. \quad C - \alpha_i - u_i = 0 \\ \alpha_i \geq 0 \\ u_i \geq 0 \end{matrix}$。由$C - \alpha_i - u_i = 0, \alpha_i \geq 0, u_i \geq 0$可得$0 \leq \alpha_i \leq C$，因为在第二步求极大值的过程中，函数只与$\alpha$有关。将上述的极大值为题转化为极小值问题：$\begin{matrix} \sum_{i=1}^{N} \alpha_i y_i = 0 \\ s.t. \quad 0 \leq \alpha_i \leq C \end{matrix}$，这就是原始问题的对偶问题。

2.2 线性支持向量机的求解过程

1、设置惩罚参数C，并求解对偶问题：$\begin{matrix} \sum_{i=1}^{N} \alpha_i y_i = 0 \\ s.t. \quad 0 \leq \alpha_i \leq C \end{matrix}$。假设求得最优解$\alpha^*$；

2、计算原始问题的最优解：$w^* = \sum_{i=1}^{N} \alpha^* y_i x_i$，选择选择$\alpha^*$中满足中满足$0 < \alpha_j^* < C$的分量，计算：$b^* = y_j - \sum_{i=1}^{N} y_i \alpha_j^* (x_i \cdot x_j)$

3、求分离超平面和分类决策函数。分离超平面为：$w^* \cdot x + b^* = 0$，分类决策函数为：$f(x) = sign(w^* \cdot x + b^*)$。

3 非线性支持向量机：

前面介绍了支持向量机的基本概念，线性可分支持向量机的原理以及线性支持向量机的原理，线性可分支持向量机是线性支持向量机的基础。对于线性支持向量机，选择一个合适的惩罚参数$C > 0$，并构造凸二次规划问题。求得原始问题的对偶问题的最优解α^*，由此可求出原始问题的最优解：

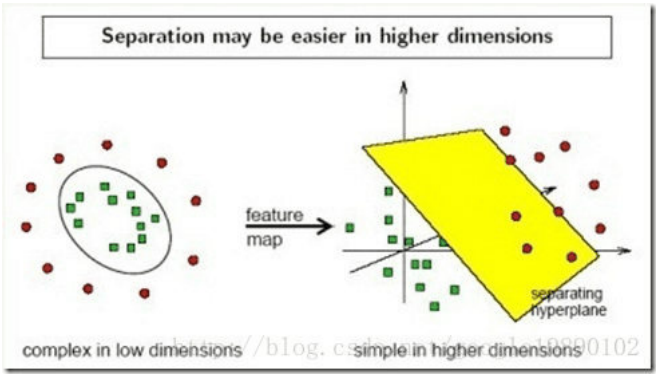
$$w^* = \sum_{i=1}^{N} \alpha_i^* y_i x_i$$

$$b^* = y_j - \sum_{i=1}^{N} y_i \alpha_i^* (x_i \cdot x_j)$$

其中(x_j, y_j)为α^*中满足$0 \leq \alpha_j \leq C$的分量。这样便可以求得分离超平面$w^* \cdot x + b^* = 0$以及分类决策函数：$f(x) = sign(w^* \cdot x + b^*)$线性可分支持向量机算法是线性支持向量机算法的特殊情况。

3.1 非线性问题的处理方法

在处理非线性问题时，可以通过将分线性问题转化成线性问题，并通过已经构建的线性支持向量机来处理。如下图所示：



(非线性转成线性问题)

通过一种映射可以将输入空间转换到对应的特征空间，体现在特征空间中的是对应的线性问题。核技巧就可以完成这样的映射工作。

3.2 核函数的定义及常见形式

设χ是输入空间(欧式空间$\mathbf{R}^n$的子集或离散集合)，又设H为特征空间(希尔伯特空间)，如果存在一个从χ到H的映射$\phi(x): \chi \to H$，使得对所有$x, y \in \chi$，函数$K(x,y)$满足条件$K(x,y) = \phi(x) \cdot \phi(y)$。则称$K(x,y)$为核函数，$\phi(x)$为映射函数。

在实际的问题中，通常使用已有的核函数。

多项式核函数(Polynomial Kernel Function)

$$K(x,y) = (x \cdot y + 1)^p$$

高斯核函数(Gaussian Kernel Function)

$$K(x,y) = exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$$

3.3 算法

1、 选取适当的核函数$K(x,y)$和适当的参数$C>0$，构造原始问题的对偶问题：$\begin{array}{l} s.t. \sum_{i=1}^{N} \alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C \end{array}$ ， 求得对应的最优解α^*

2、 选择α^*的一个满足$0 \leq \alpha_j \leq C$的分量，求:b^*：$b^* = y_j - \sum_{i=1}^{N} y_i \alpha_i^* \ K(x_i \cdot x_j)$

3、构造决策函数

$$f\left(x\right) = sign\left(\sum_{i=1}^{N} y_i \alpha_i^* \ K(x \cdot x_i) + b^*\right)$$

In [1]:

```
import plotly.plotly as py              # for sending things to plotly
import plotly.tools as tls              # for mpl, config, etc.
from plotly.graph_objs import *         # __all__ is safely defined
import plotly.graph_objs as go
import cufflinks as cf
import pandas as pd
from pandas import Series,DataFrame
import numpy as np
import datetime

py.sign_in('DemoAccount','2qdyfjyr7o')
```

核函数¶

In [2]:

```
def get_y1(x):
    return (25-x**2)**0.5
def get_y2(x):
    return (100-x**2)**0.5
```

In [3]:

```
symbol = np.random.random(100)
```

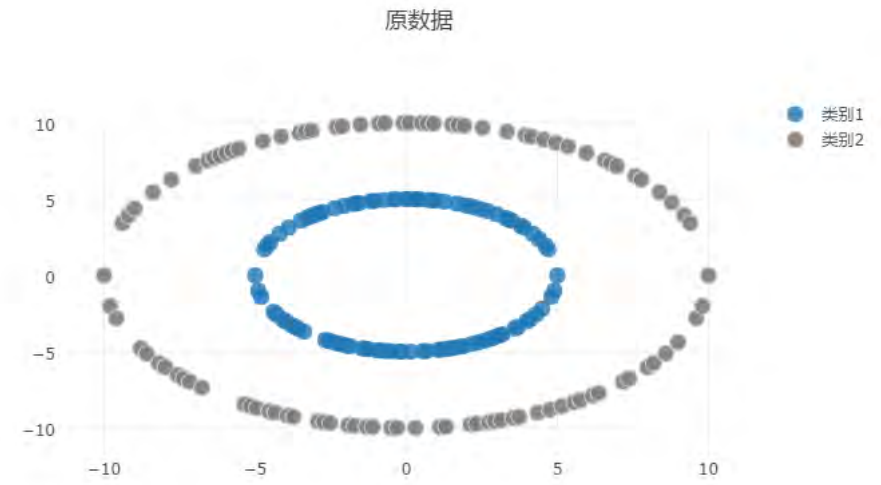
In [4]:

```
symbol[symbol>0.5] = 1
symbol[symbol<=0.5] = -1
# 添加一点小噪音也没多大的意思
x1 = np.linspace(-5, 5, 100)
x2 = np.linspace(-10, 10, 100)
y1 = []
y2 = []
for x in x1:
    y1.append(get_y1(x))
for x in x2:
    y2.append(get_y2(x))
y1 = np.array(y1)*symbol
y2 = np.array(y2)*symbol
```

In [ ]:

```
trace0 = go.Scatter(
    x = x1,
    y = y1,
    name = '类别1',
    mode='markers',
    marker=dict(
        size=12,
        line=dict(
            color='rgba(217, 217, 217, 0.14)',
            width=0.5
        ),
        opacity=0.8
    )
)

trace1 = go.Scatter(
    x = x2,
    y = y2,
    name = '类别2',
    mode='markers',
```

```
    marker=dict(
        color='rgb(127, 127, 127)',
        size=12,
        symbol='circle',
        line=dict(
            color='rgb(204, 204, 204)',
            width=1
        ),
        opacity=0.9
    )
)

data = [trace0, trace1]

layout = dict(title = '原数据',
              yaxis = dict(zeroline = False),
              xaxis = dict(zeroline = False)
             )

fig = dict(data=data, layout=layout)
py.iplot(fig, filename='原数据集')
```



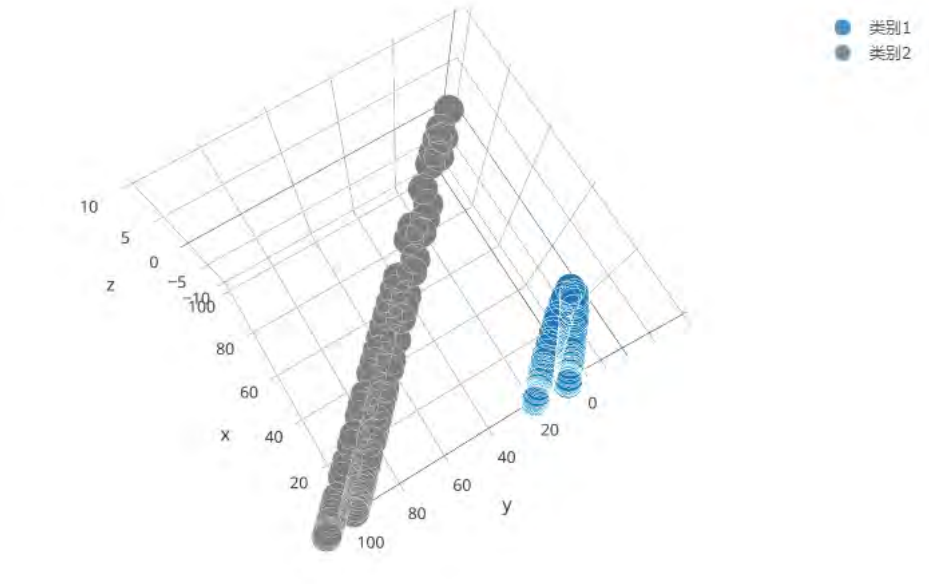
In [7]:

```
# 进行核变换
a1 = x1**2
a2 = y1**2
a3 = y1
b1 = x2**2
b2 = y2**2
b3 = y2
```

In [ ]:

```
import plotly.plotly as py
import plotly.graph_objs as go
trace1 = go.Scatter3d(
    x=a1,
    y=a2,
    z=a3,
    name = '类别1',
    mode='markers',
    marker=dict(
        size=12,
        line=dict(
            color='rgba(217, 217, 217, 0.14)',
            width=0.5
        ),
        opacity=0.8
    )
)


trace2 = go.Scatter3d(
    x=b1,
    y=b2,
    z=b3,
```

```
    name = '类别2',
    mode='markers',
    marker=dict(
        color='rgb(127, 127, 127)',
        size=12,
        symbol='circle',
        line=dict(
            color='rgb(204, 204, 204)',
            width=1
        ),
        opacity=0.9
    )
)
data = [trace1, trace2]
layout = go.Layout(
    margin=dict(
        l=0,
        r=0,
        b=0,
        t=0
    )
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='映射后的数据集')
```



可以点击克隆下来，自己转着看一下

核函数的作用我们做出来一个图，可以给大家清晰的展示出来他到底有什么作用。在二位空间内，看似线性不可分，但是转换为三位空间后，这就编程了一个非常简单的事情。

机器学习系列：进阶算法----随机森林

机器学习中，随机森林是一种组合方法，由许多的决策树组成，因为这些决策树的形成采用了随机的方法，因此也叫做随机决策树。随机森林中的树之间是没有关联的。

当测试数据进入随机森林时，其实就是让每一颗决策树进行分类，最后取所有决策树中分类结果最多的那类为最终的结果。因此随机森林是一个包含多个决策树的分类器，并且其输出的类别是由个别树输出的类别的众数而定。 随机森林由决策树组成，决策树实际上是将空间用超平面进行划分的一种方法，每次分割的时候，都将当前的空间一分为二，使得每一个叶子节点都是在空间中的一个不相交的区域，在进行决策的时候，会根据输入样本每一维feature的值，计算信息增益，或者Gini值，一步一步往下分裂，最后使得样本落入N个区域中的一个(假设有N个叶子节点，对于二元分类，N=2)。

随机森林可以既可以处理属性为离散值的量，比如ID3算法，也可以处理属性为连续值的量，比如C4.5算法。

随机森林的优点：

1.适合做多分类问题；当存在分类不平衡的情况时，随机森林能够提供平衡数据集误差的有效方法(通过属性评估?)； 2.训练和预测速度快； 3.对训练数据的容错能力，是一种有效估计missing值的方法，当数据集中有大比例的数据缺失时仍然可以保持精度不变； 4.能够有效地处理大的数据集； 5.它能够处理很高维度(feature很多)的数据，并且不用做特征选择 6.能够在分类的过程中可以生成一个泛化误差的内部无偏估计(OOB error可以作为泛化误差的一个估计)； 7.能够检测到特征之间的相互影响以及重要性程度(通过feature_importances_方法)； 8.不容易出现过度拟合；实现简单容易并行化(通过n_jobs)。 9.模型的上述性能可以被扩展运用到未标记的数据集中，用于引导无监督聚类、数据透视和异常检测；

随机森林的缺点：

1.对于有不同级别的属性的数据，级别划分较多的属性会对随机森林产生更大的影响，所以随机森林在这种数据上产出的属性权值是不可信的； 2.单棵决策树的预测效果很差：由于随机选择属性，使得单棵决策树的预测效果很差。 3.随机森林在解决回归问题时并没有像它在分类中表现的那么好，这是因为它并不能给出一个连续型的输出。当进行回归时，随机森林不能够作出超越训练集数据范围的预测，这可能导致在对某些还有特定噪声的数据进行建模时出现

过度拟合。 4.对于许多统计建模者来说，随机森林给人的感觉像是一个黑盒子，你几乎无法控制模型内部的运行，只能在不同的参数和随机种子之间进行尝试。

随机森林的定义¶

　　在机器学习中，随机森林是一个包含多个决策树的分类器， 并且其输出的类别是由个别树输出的类别的众数而定。 Leo Breiman和Adele Cutler发展出推论出随机森林的算法。 而 "Random Forests" 是他们的商标。 这个术语是1995年由贝尔实验室的Tin Kam Ho所提出的随机决策森林（random decision forests）而来的。这个方法则是结合 Breimans 的 "Bootstrap aggregating" 想法和 Ho 的"random subspace method"以建造决策树的集合。

　　随机森林是一种统计学习理论，其随机有两个方面：首先在训练的每一轮中，都是对原始样本集有放回的抽取固定数目的样本点，形成k 个互不相同的样本集。第二个点是：对于每一个决策树的建立是从总的属性中随机抽取一定量的属性作为分裂属性集，这样对于k个树分类器均是不相同的。由随机生成的k个决策树组成了随机森林。

sklearn 中的随机森林¶

　　在scikit-learn中，RF的分类类是RandomForestClassifier，回归类是RandomForestRegressor。当然RF的变种Extra Trees也有， 分类类ExtraTreesClassifier，回归类ExtraTreesRegressor。由于RF和Extra Trees的区别较小，调参方法基本相同，本文只关注于RF的调参。

　　和GBDT的调参类似，RF需要调参的参数也包括两部分，第一部分是Bagging框架的参数，第二部分是CART决策树的参数。下面我们就对这些参数做一个介绍。

　　随机森林主要有那种算法，一种是随机森林的分类算法，另外一个自然就是回归了。下面我们列出了两个类的不同。

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fract
```

```
class sklearn.ensemble.RandomForestRegressor(n_estimators=10, criterion='mse', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fractio
```

<font size=3>

　　上面已经介绍，随机森林是一个包含多个决策树的分类器，所以它的参数我们分两部分介绍：1、决策树的参数；2、随机森林特有的参数。

sklearn中决策树的参数：

　　1，criterion: ”gini” or “entropy”(default=”gini”)是计算属性的gini(基尼不纯度)还是entropy(信息增益)，来选择最合适的节点。

　　2，splitter: ”best” or “random”(default=”best”)随机选择属性还是选择不纯度最大的属性，建议用默认。

　　3，max_features: 选择最适属性时划分的特征不能超过此值。当为整数时，即最大特征数；如果是小数，训练集特征数百分比；如果是“auto”，那么max_features = sqrt（n_features）。如果“sqrt”，则max_features = sqrt（n_features）（与“auto”相同）。如果“log2”，则max_features = log2（n_features）。如果没有，那么max_features = n_features。

　　4，max_depth: (default=None)设置树的最大深度，默认为None，这样建树时，会使每一个叶节点只有一个类别，或是达到min_samples_split。

　　5，min_samples_split:根据属性划分节点时，每个划分最少的样本数。

　　6，min_samples_leaf:叶子节点最少的样本数。

　　7，max_leaf_nodes: (default=None)叶子树的最大样本数。

　　8，min_weight_fraction_leaf: (default=0) 叶子节点所需要的最小权值

　　9，verbose:(default=0) 是否显示任务进程

随机森林特有的参数：

　　1，n_estimators=10：决策树的个数，越多越好，但是性能就会越差，至少100左右（具体数字忘记从哪里来的了）可以达到可接受的性能和误差率。

　　2，bootstrap=True：是否有放回的采样。

　　3，oob_score=False：oob（out of band，带外）数据，即：在某次决策树训练中没有被bootstrap选中的数据。多单个模型的参数训练，我们知道可以用cross validation（cv）来进行，但是特别消耗时间，而且对于随机森林这种情况也没有大的必要，所以就用这个数据对决策树模型进行验证，算是一个简单的交叉验证。性能消耗小，但是效果不错。

　　4，n_jobs=1：并行job个数。这个在ensemble算法中非常重要，尤其是bagging（而非boosting，因为boosting的每次迭代之间有影响，所以很难进行并行化），因为可以并行从而提高性能。1=不并行；n：n个并行；-1：CPU有多少core，就启动多少job。

　　5，warm_start=False：热启动，决定是否使用上次调用该类的结果然后增加新的。

　　6，class_weight=None：各个label的权重。

　　对比上面两个类，唯一的不同就在于最后一个参数class_weight，给出各个标签的权重。

<font size=3>

进行预测可以有几种形式:

　　1，predict_proba(x)：给出带有概率值的结果。每个点在所有label的概率和为1.

　　2，predict(x)：直接给出预测结果。内部还是调用的predict_proba()，根据概率的结果看哪个类型的预测值最高就是哪个类型。

　　3，predict_log_proba(x)：和predict_proba基本上一样，只是把结果给做了log()处理。

<font size=3>　　通过总结模型调参常见的问题，我们可以把模型的参数分为4类：目标类、性能类、效率类和附加类。下表详细地展示了4个模型参数的意义

分类问题¶

In [1]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import numpy as np

from sklearn.datasets import load_iris
iris=load_iris()
#print iris#iris的4个属性是：萼片宽度  萼片长度  花瓣宽度  花瓣长度  标签是花的种类：setosa versicolour virginica
print(iris['target'].shape)
rf=RandomForestRegressor()#这里使用了默认的参数设置
rf.fit(iris.data[:150],iris.target[:150])#进行模型的训练
#
#随机挑选两个预测不相同的样本
instance=iris.data[[100,109]]
print(instance)
print('instance 0 prediction：',rf.predict(instance[0]))
print('instance 1 prediction：',rf.predict(instance[1]))
print(iris.target[100],iris.target[109])
```

```
(150,)
[[ 6.3  3.3  6.   2.5]
 [ 7.2  3.6  6.1  2.5]]
instance 0 prediction：  [ 2.]
instance 1 prediction：  [ 2.]
2 2
```

```
/srv/env/lib64/python3.4/site-packages/sklearn/utils/validation.py:395: DeprecationWarning: Passing 1d arrays as data is deprecated i
  DeprecationWarning)
/srv/env/lib64/python3.4/site-packages/sklearn/utils/validation.py:395: DeprecationWarning: Passing 1d arrays as data is deprecated i
  DeprecationWarning)
```

可以对比看到我们的分类是完全正确

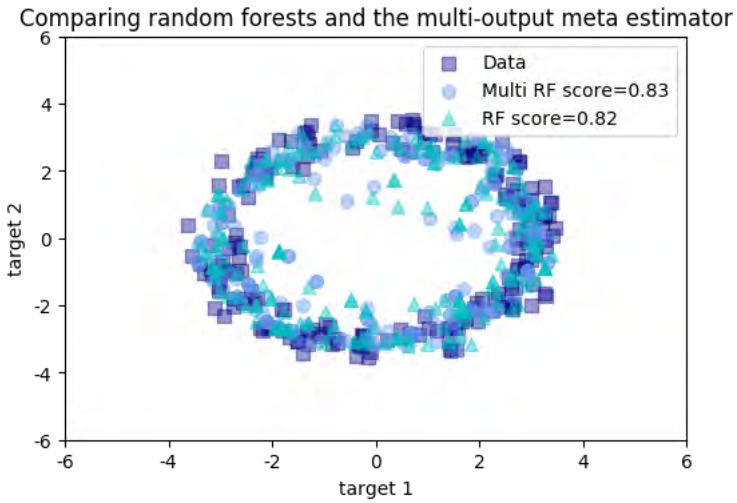
回归问题¶

<font size=3>　　我们对比了随机森林和多输出的回归模型，比较两个模型的效果。

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputRegressor


# Create a random dataset
rng = np.random.RandomState(1)
X = np.sort(200 * rng.rand(600, 1) - 100, axis=0)
y = np.array([np.pi * np.sin(X).ravel(), np.pi * np.cos(X).ravel()]).T
y += (0.5 - rng.rand(*y.shape))

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=400,
                                                    random_state=4)

max_depth = 30
regr_multirf = MultiOutputRegressor(RandomForestRegressor(max_depth=max_depth,
                                                          random_state=0))
regr_multirf.fit(X_train, y_train)

regr_rf = RandomForestRegressor(max_depth=max_depth, random_state=2)
regr_rf.fit(X_train, y_train)

# Predict on new data
y_multirf = regr_multirf.predict(X_test)
y_rf = regr_rf.predict(X_test)

# Plot the results
plt.figure()
s = 50
a = 0.4
plt.scatter(y_test[:, 0], y_test[:, 1],
            c="navy", s=s, marker="s", alpha=a, label="Data")
plt.scatter(y_multirf[:, 0], y_multirf[:, 1],
            c="cornflowerblue", s=s, alpha=a,
            label="Multi RF score=%.2f" % regr_multirf.score(X_test, y_test))
plt.scatter(y_rf[:, 0], y_rf[:, 1],
```

```
            c="c", s=s, marker="^", alpha=a,
            label="RF score=%.2f" % regr_rf.score(X_test, y_test))
plt.xlim([-6, 6])
plt.ylim([-6, 6])
plt.xlabel("target 1")
plt.ylabel("target 2")
plt.title("Comparing random forests and the multi-output meta estimator")
plt.legend()
plt.show()
```



泰坦尼克号¶

<font size=3>　　泰坦尼克号的沉没是历史上最著名的沉船时间之一。 1912年4月15日，泰坦尼克号在首航期间，与冰山相撞后沉没，在2224名乘客和船员中造成1502人死亡。 这场悲剧震撼了国际社会，并为船舶制定了更好的安全规定。造成这种生命损失的原因之一是乘客和船员没有足够的救生艇。 虽然有幸遇难下沉，但一些群体比女性，儿童和上层阶级更有可能生存下去。
现在我们拿到了这些数据，并对这些数据进行分析，以探讨个体生命生存的可能性。

<font size=3>　　数据已分为两组，即“训练集”和“测试集”。 对于训练集，我们为每个乘客提供了标签（是否生还）。 对于测试集中的每个乘客，我们将根据得到的特征判断其是否生还（0人死亡，1人幸存）。

In [1]:

```
import pandas as pd
import numpy as np
import pylab as plt

# Set the global default size of matplotlib figures
plt.rc('figure', figsize=(10, 5))

# Size of matplotlib figures that contain subplots
fizsize_with_subplots = (10, 10)

# Size of matplotlib histogram bins
bin_size = 10
```

In [2]:

```
# 导入数据
df_train = pd.read_csv('train.csv')
df_train.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

In [3]:

```
# 统计性描述
df_train.describe()
```

Out[3]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

<font size=3>　　上面现在我们对数据集内容有一个总体的看法，观察一下具体的每个特征。 如果特征单独就具备很好的区分度这是一个不错的事情。

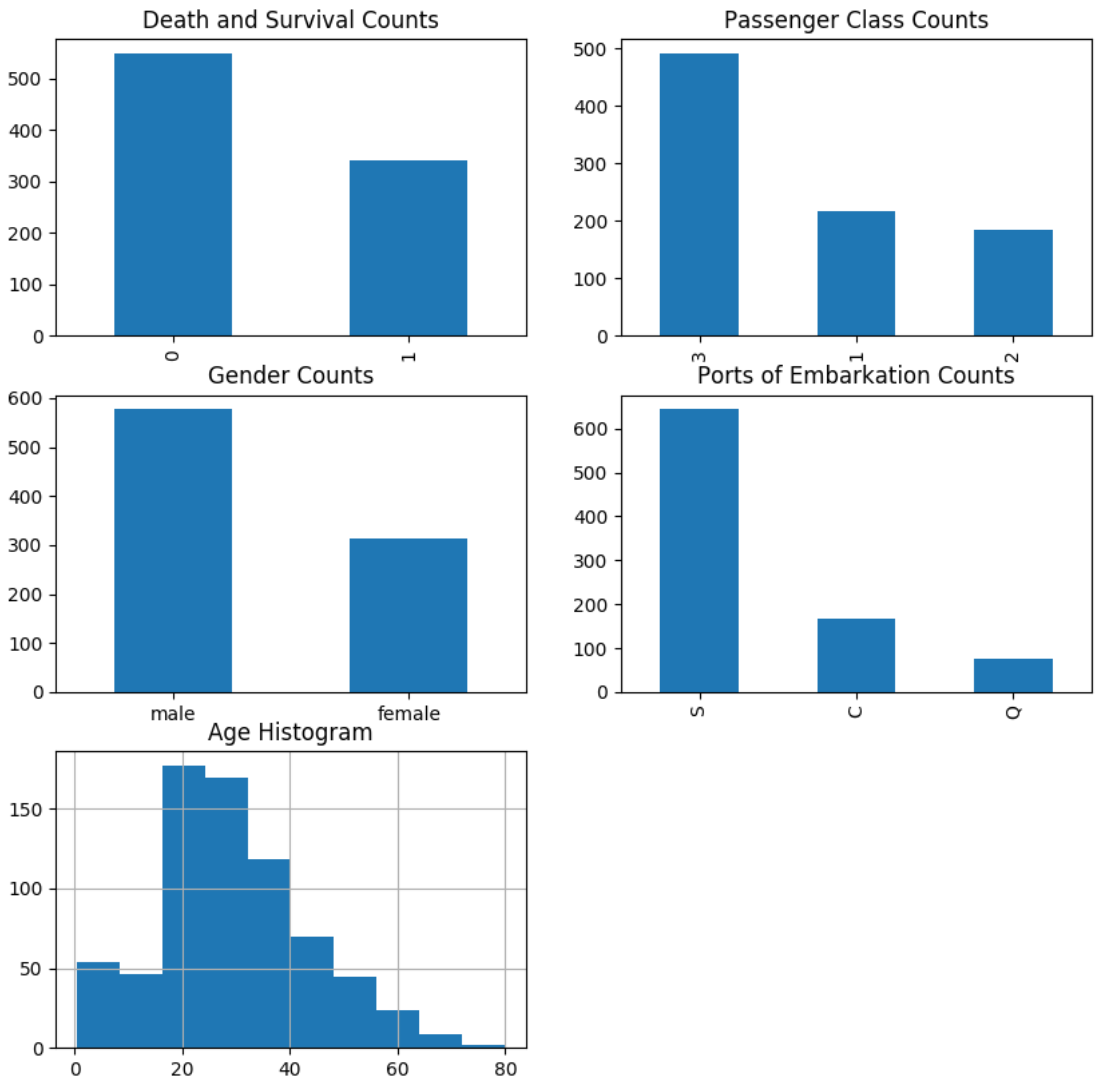
In [4]:

```
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (3, 2)

# 生死的区别
plt.subplot2grid(fig_dims, (0, 0))
df_train['Survived'].value_counts().plot(kind='bar',
                                         title='Death and Survival Counts')

# 乘客类型
plt.subplot2grid(fig_dims, (0, 1))
df_train['Pclass'].value_counts().plot(kind='bar',
                                       title='Passenger Class Counts')

# 性别
plt.subplot2grid(fig_dims, (1, 0))
df_train['Sex'].value_counts().plot(kind='bar',
                                    title='Gender Counts')
plt.xticks(rotation=0)

# 位置
plt.subplot2grid(fig_dims, (1, 1))
df_train['Embarked'].value_counts().plot(kind='bar',
                                         title='Ports of Embarkation Counts')

# 年龄段
```

```
plt.subplot2grid(fig_dims, (2, 0))
df_train['Age'].hist()
plt.title('Age Histogram')
```

Out[4]:

```
<matplotlib.text.Text at 0x7fd3ae47ffd0>
```



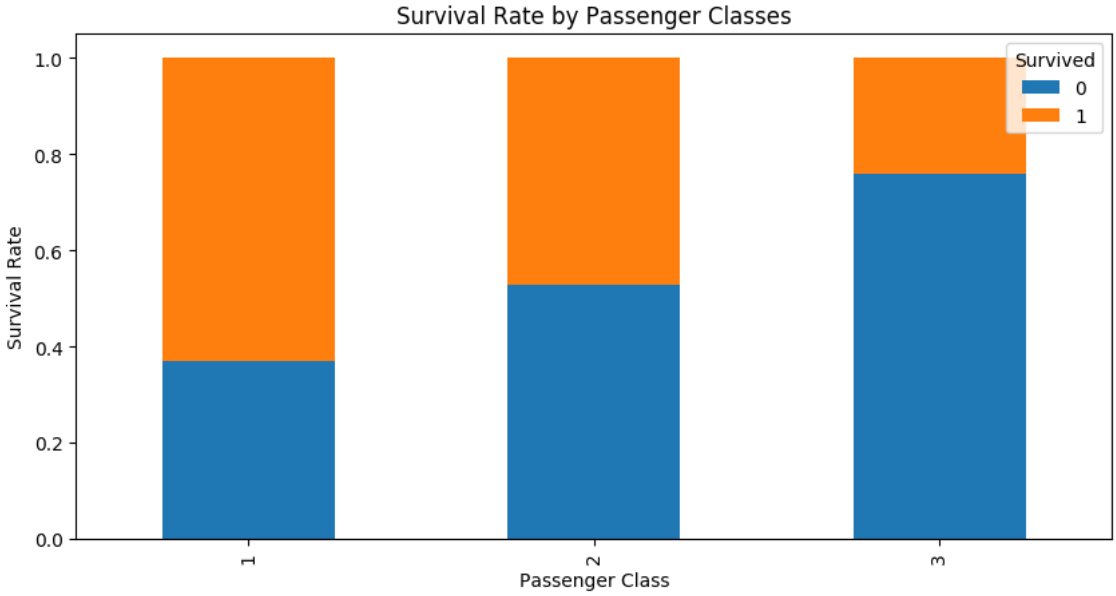
乘客类型¶

In [5]:

```
# 我们将观察各个类型程度的存活率
pclass_xt = pd.crosstab(df_train['Pclass'], df_train['Survived'])
pclass_xt_pct = pclass_xt.div(pclass_xt.sum(1).astype(float), axis=0)

pclass_xt_pct.plot(kind='bar',
                   stacked=True,
                   title='Survival Rate by Passenger Classes')
plt.xlabel('Passenger Class')
plt.ylabel('Survival Rate')
```

Out[5]:

```
<matplotlib.text.Text at 0x7fd3ae43afd0>
```



性别¶

In [6]:

```
sexes = sorted(df_train['Sex'].unique())
genders_mapping = dict(zip(sexes, range(0, len(sexes) + 1)))
genders_mapping
```

Out[6]:

```
{'female': 0, 'male': 1}
```

In [7]:

```
df_train['Sex_Val'] = df_train['Sex'].map(genders_mapping).astype(int)
df_train.head()
```

Out[7]:

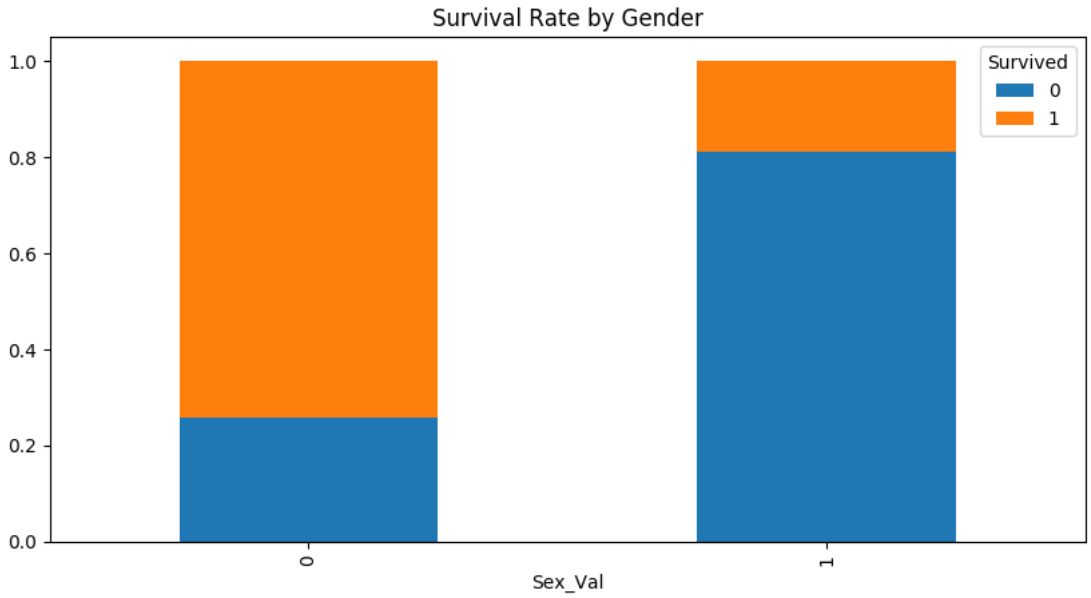
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

In [8]:

```
sex_val_xt = pd.crosstab(df_train['Sex_Val'], df_train['Survived'])
sex_val_xt_pct = sex_val_xt.div(sex_val_xt.sum(1).astype(float), axis=0)
sex_val_xt_pct.plot(kind='bar', stacked=True, title='Survival Rate by Gender')
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd3ae689588>
```



<font size=3>　　大多数女性幸存下来，而大多数男性没有幸免。接下来，我们将通过查看Sex和Pclass来确定我们是否可以获得关于生存率的任何突破。

In [9]:

```
passenger_classes = sorted(df_train['Pclass'].unique())

for p_class in passenger_classes:
    print('M: ', p_class, len(df_train[(df_train['Sex'] == 'male') & (df_train['Pclass'] == p_class)]))
    print('F: ', p_class, len(df_train[(df_train['Sex'] == 'female') & (df_train['Pclass'] == p_class)]))
```

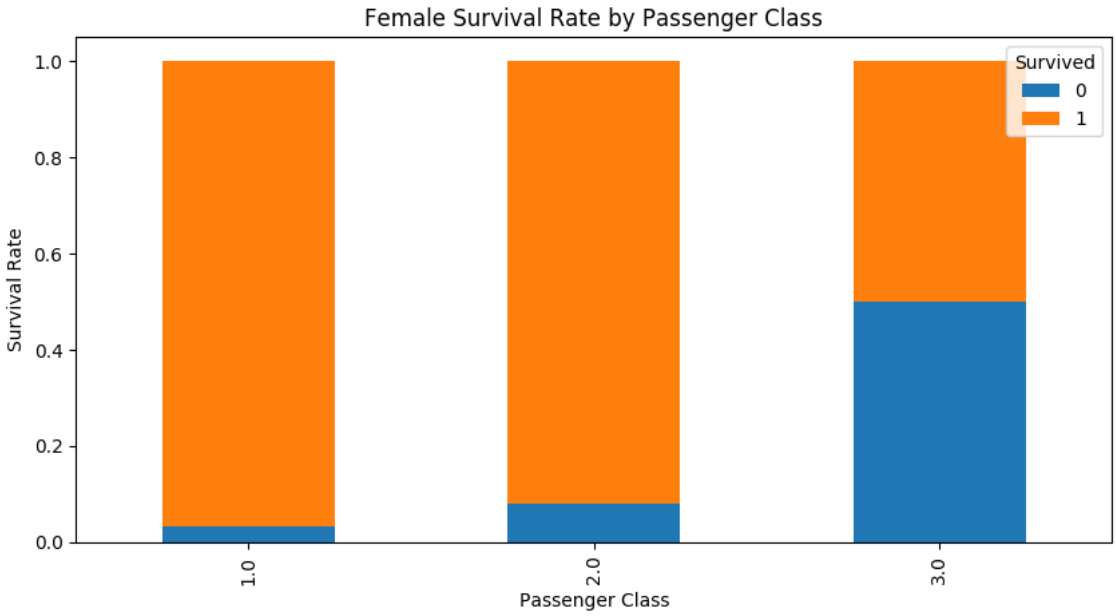
```
M:  1 122
F:  1 94
M:  2 108
F:  2 76
M:  3 347
F:  3 144
```

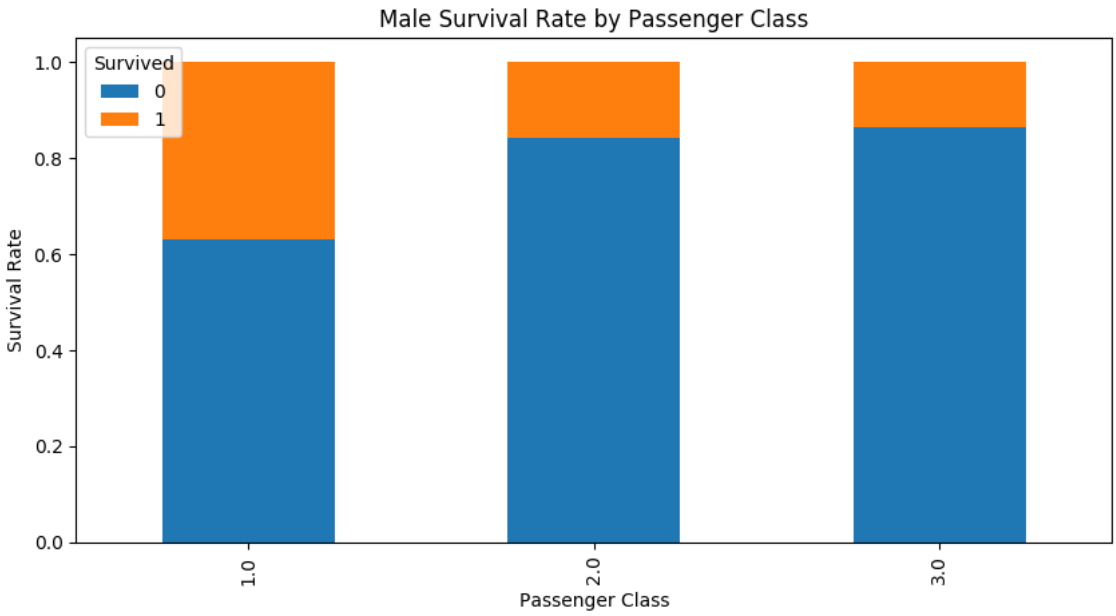
In [10]:

```
# 性别不同情况下类别不同乘客的生存率
females_df = df_train[df_train['Sex'] == 'female']
females_xt = pd.crosstab(females_df['Pclass'], df_train['Survived'])
females_xt_pct = females_xt.div(females_xt.sum(1).astype(float), axis=0)
females_xt_pct.plot(kind='bar',
                    stacked=True,
                    title='Female Survival Rate by Passenger Class')
plt.xlabel('Passenger Class')
plt.ylabel('Survival Rate')

# Plot survival rate by Pclass
males_df = df_train[df_train['Sex'] == 'male']
males_xt = pd.crosstab(males_df['Pclass'], df_train['Survived'])
males_xt_pct = males_xt.div(males_xt.sum(1).astype(float), axis=0)
males_xt_pct.plot(kind='bar',
                  stacked=True,
                  title='Male Survival Rate by Passenger Class')
plt.xlabel('Passenger Class')
plt.ylabel('Survival Rate')
```

Out[10]:

```
<matplotlib.text.Text at 0x7fd3ac1dbe10>
```





登船数量¶

In [11]:

```
df_train[df_train['Embarked'].isnull()]
```

Out[11]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
61	62	1	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0	B28	NaN
829	830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0	0	0	113572	80.0	B28	NaN

In [12]:

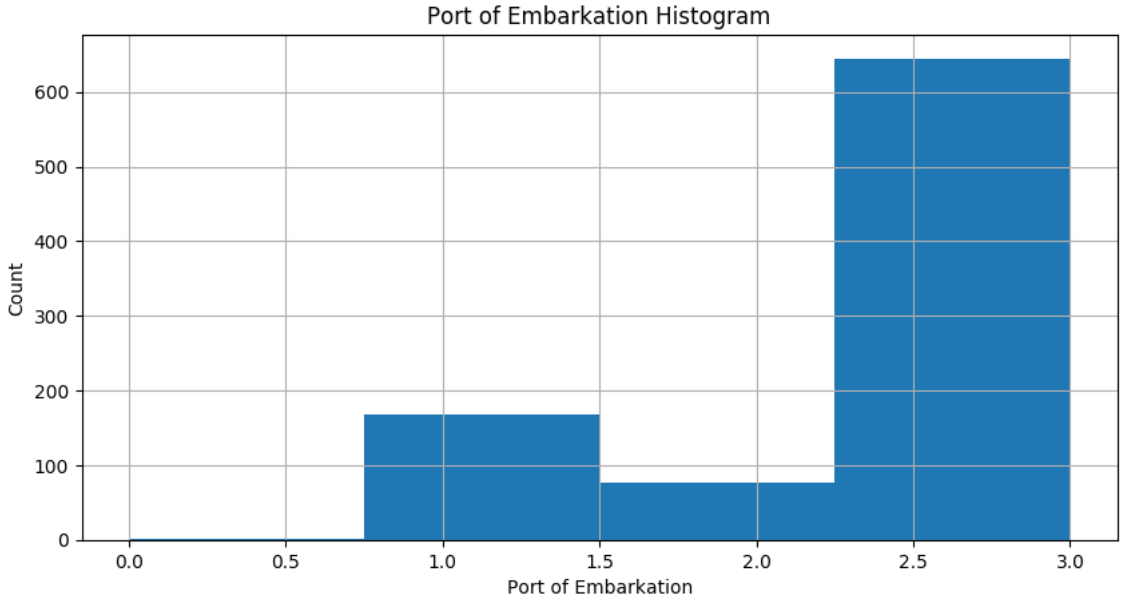
```
embarked_locs_mapping = {np.nan: 0, 'C': 1, 'Q': 2, 'S': 3}
df_train['Embarked_Val'] = df_train['Embarked'] \
                           .map(embarked_locs_mapping) \
                           .astype(int)
df_train.head()
```

Out[12]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

In [13]:

```
df_train['Embarked_Val'].hist(bins=4, range=(0, 3))
plt.title('Port of Embarkation Histogram')
plt.xlabel('Port of Embarkation')
plt.ylabel('Count')
plt.show()
```



In [14]:

```
if len(df_train[df_train['Embarked'].isnull()] > 0):
    df_train.replace({'Embarked_Val' :
                { embarked_locs_mapping[np.nan] : embarked_locs_mapping['S']
                }
            },
            inplace=True)
```

In [15]:

```
embarked_locs = sorted(df_train['Embarked_Val'].unique())
embarked_locs
```

Out[15]:

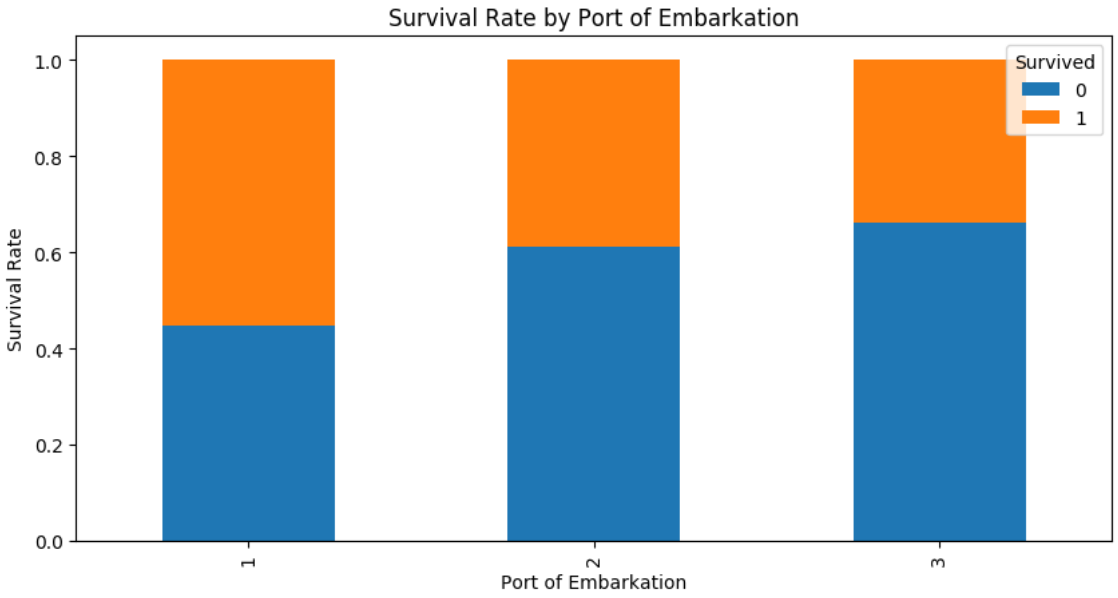
```
[1, 2, 3]
```

In [16]:

```
embarked_val_xt = pd.crosstab(df_train['Embarked_Val'], df_train['Survived'])
embarked_val_xt_pct = \
    embarked_val_xt.div(embarked_val_xt.sum(1).astype(float), axis=0)
embarked_val_xt_pct.plot(kind='bar', stacked=True)
plt.title('Survival Rate by Port of Embarkation')
plt.xlabel('Port of Embarkation')
plt.ylabel('Survival Rate')
```

Out[16]:

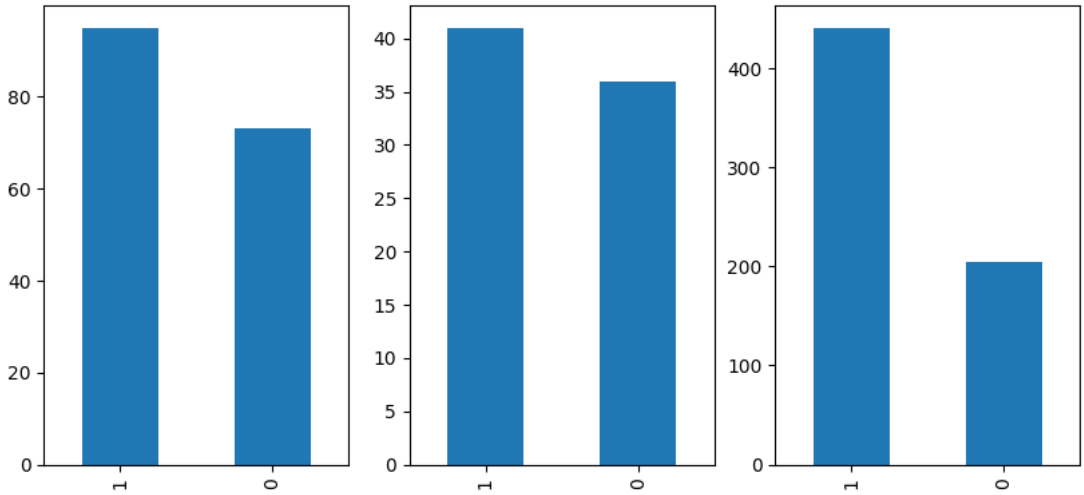
```
<matplotlib.text.Text at 0x7fd3ac13dcc0>
```

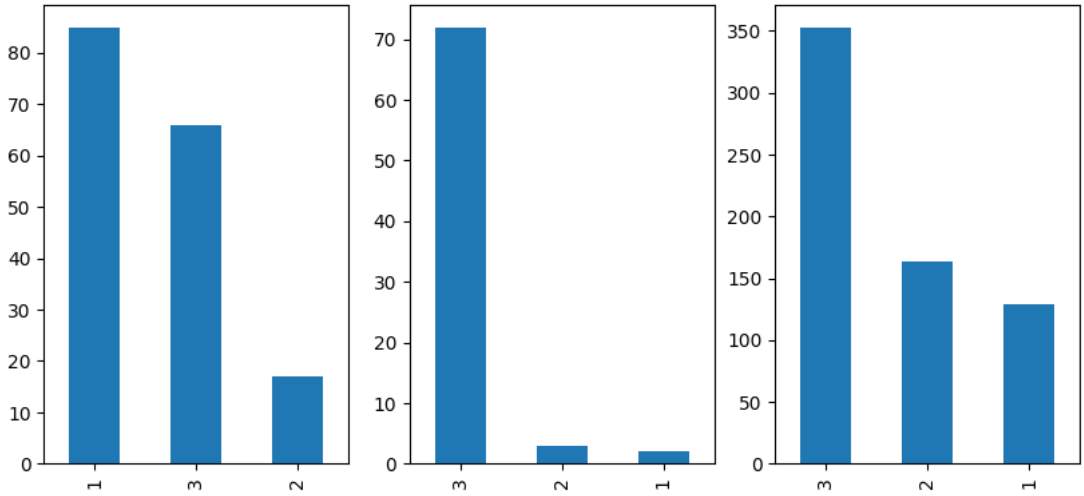


看来那些开始位置“C”的人是最高的生存率。 我们会再多挖一些，看看为什么会出现这种情况。 下面我们绘制一个图表，以确定每个港口的性别和乘客班级构成：

In [17]:

```
fig = plt.figure(figsize=fizsize_with_subplots)

rows = 2
cols = 3
col_names = ('Sex_Val', 'Pclass')

for portIdx in embarked_locs:
    for colIdx in range(0, len(col_names)):
        plt.subplot2grid((rows, cols), (colIdx, portIdx - 1))
        df_train[df_train['Embarked_Val'] == portIdx][col_names[colIdx]] \
            .value_counts().plot(kind='bar')
```





In [18]:

```
df_train = pd.concat([df_train, pd.get_dummies(df_train['Embarked_Val'], prefix='Embarked_Val')], axis=1)
```

年纪¶

In [19]:

```
df_train[df_train['Age'].isnull()][['Sex', 'Pclass', 'Age']].head()
```

Out[19]:

	Sex	Pclass	Age
5	male	3	NaN
17	male	2	NaN
19	female	3	NaN
26	male	3	NaN
28	female	3	NaN

In [20]:

```
df_train['AgeFill'] = df_train['Age']
df_train['AgeFill'] = df_train['AgeFill'] \
                      .groupby([df_train['Sex_Val'], df_train['Pclass']]) \
                      .apply(lambda x: x.fillna(x.median()))
```

In [21]:

```
len(df_train[df_train['AgeFill'].isnull()])
```

Out[21]:

```
0
```

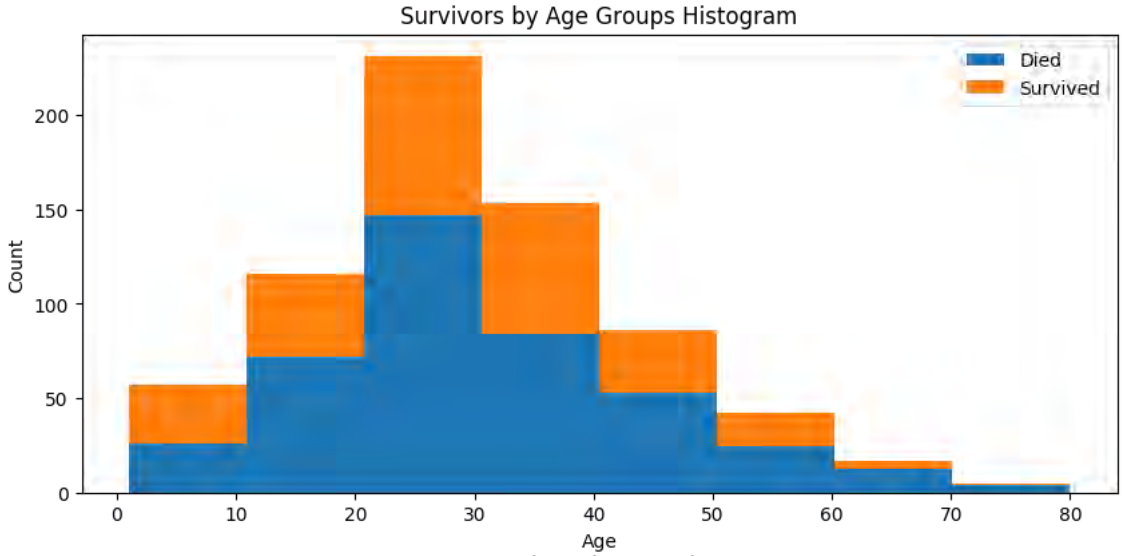
In [22]:

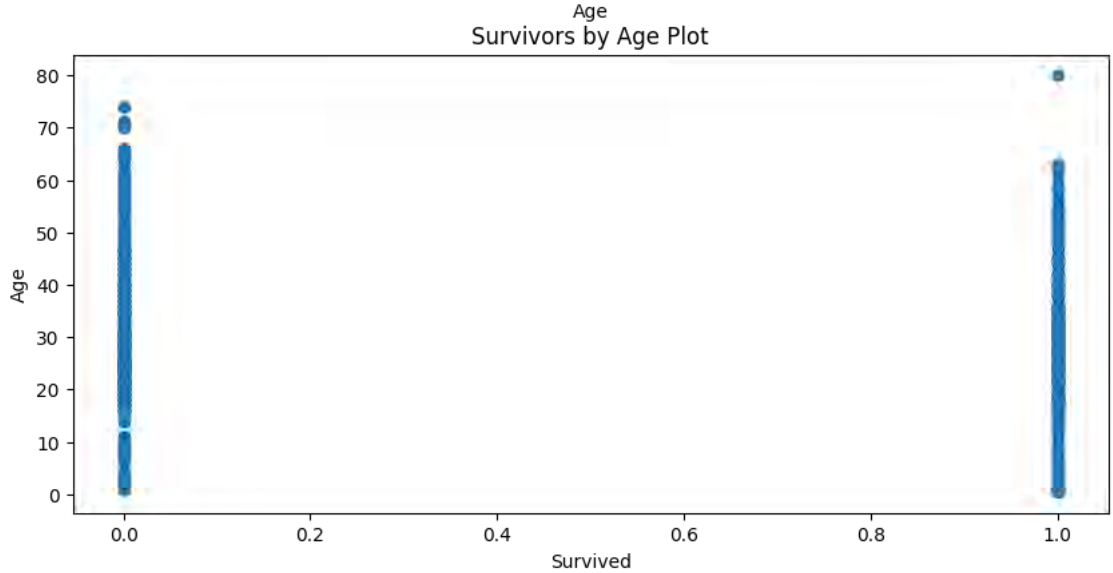
```
fig, axes = plt.subplots(2, 1, figsize=fizsize_with_subplots)

# Histogram of AgeFill segmented by Survived
df1 = df_train[df_train['Survived'] == 0]['Age']
df2 = df_train[df_train['Survived'] == 1]['Age']
max_age = max(df_train['AgeFill'])
axes[0].hist([df1, df2],
             bins=int(max_age / bin_size),
             range=(1, max_age),
             stacked=True)
axes[0].legend(('Died', 'Survived'), loc='best')
axes[0].set_title('Survivors by Age Groups Histogram')
axes[0].set_xlabel('Age')
axes[0].set_ylabel('Count')

# Scatter plot Survived and AgeFill
axes[1].scatter(df_train['Survived'], df_train['AgeFill'])
axes[1].set_title('Survivors by Age Plot')
axes[1].set_xlabel('Survived')
axes[1].set_ylabel('Age')
```

```
/srv/env/lib64/python3.4/site-packages/numpy/lib/function_base.py:747: RuntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= mn)
/srv/env/lib64/python3.4/site-packages/numpy/lib/function_base.py:748: RuntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= mx)
```

Out[22]:

```
<matplotlib.text.Text at 0x7fd3a46dc390>
```





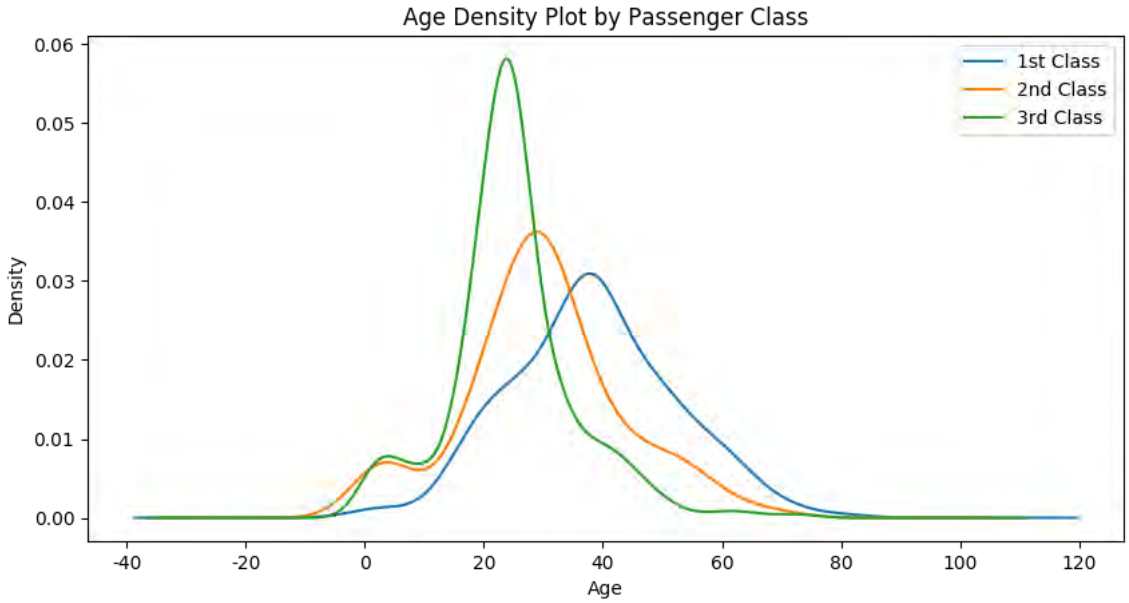
In [23]:

```
for pclass in passenger_classes:
    df_train.AgeFill[df_train.Pclass == pclass].plot(kind='kde')
```

```
plt.title('Age Density Plot by Passenger Class')
plt.xlabel('Age')
plt.legend(('1st Class', '2nd Class', '3rd Class'), loc='best')
```

Out[23]:

```
<matplotlib.legend.Legend at 0x7fd3ac0205f8>
```



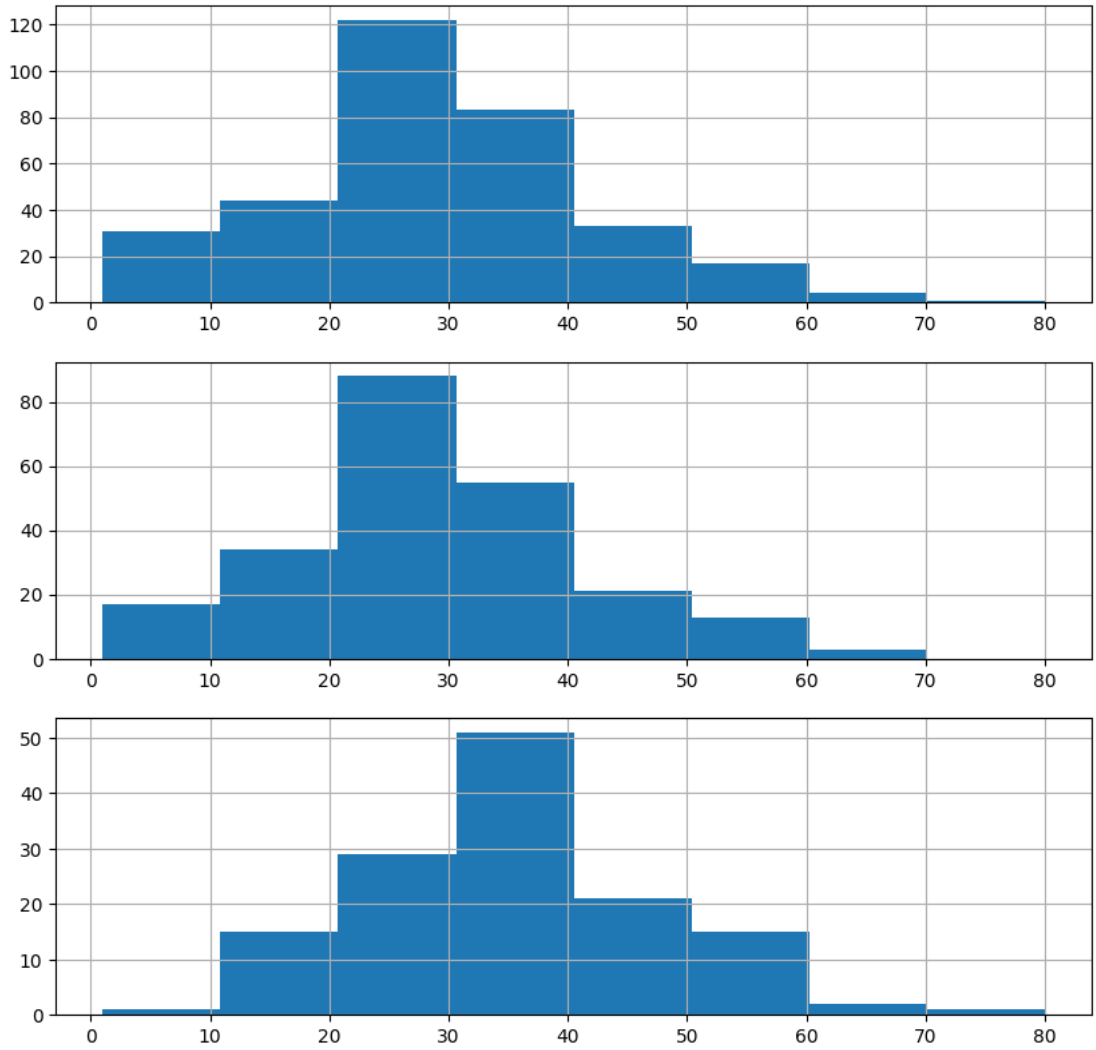
通过概率密度图，我们看到第一类乘客一般比二等乘客大，而二级乘客则大于三级乘客。 我们确定一等乘客的生存率要高于二级乘客，而乘客的生存率又高于三级乘客。

In [24]:

```
# Set up a grid of plots
fig = plt.figure(figsize=fizsize_with_subplots)
fig_dims = (3, 1)

# Plot the AgeFill histogram for Survivors
plt.subplot2grid(fig_dims, (0, 0))
survived_df = df_train[df_train['Survived'] == 1]
survived_df['AgeFill'].hist(bins=int(max_age / bin_size), range=(1, max_age))

# Plot the AgeFill histogram for Females
plt.subplot2grid(fig_dims, (1, 0))
females_df = df_train[(df_train['Sex_Val'] == 0) & (df_train['Survived'] == 1)]
females_df['AgeFill'].hist(bins=int(max_age / bin_size), range=(1, max_age))

# Plot the AgeFill histogram for first class passengers
plt.subplot2grid(fig_dims, (2, 0))
class1_df = df_train[(df_train['Pclass'] == 1) & (df_train['Survived'] == 1)]
class1_df['AgeFill'].hist(bins=int(max_age / bin_size), range=(1, max_age))
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd3a47ae7b8>
```



<font size=3>　　在第一张图中，我们看到大多数幸存者都是从20岁到30岁的年龄段，可以被以下两个图解释。 第二张图表显示，大多数女性都在20多岁。 第三张图表显示，大多数一流的乘客都在30多岁。

家庭人口数量¶

In [25]:

```
df_train['FamilySize'] = df_train['SibSp'] + df_train['Parch']
df_train.head()
```

Out[25]:

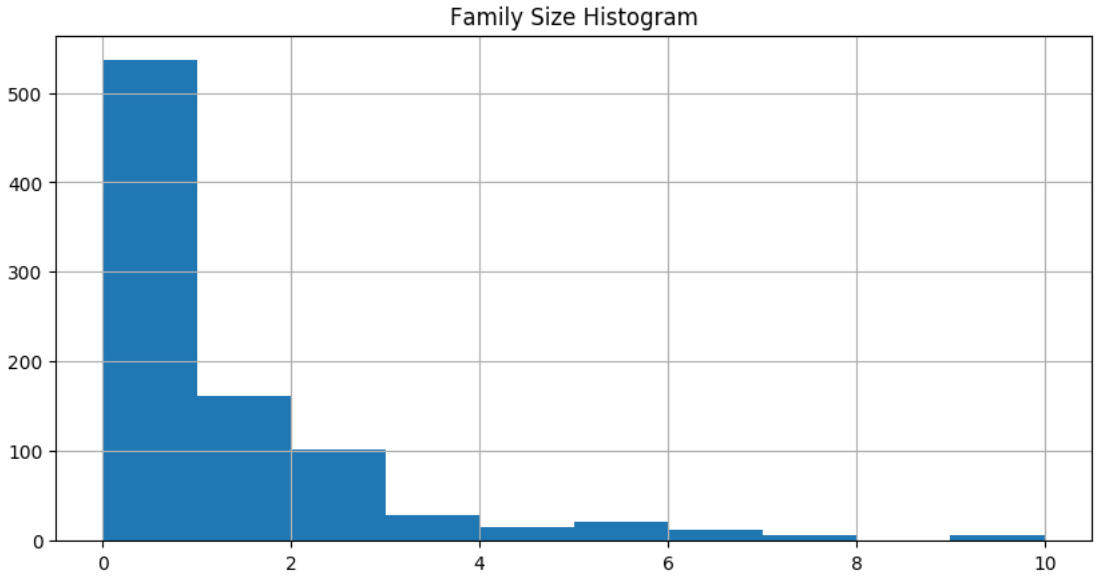
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

In [26]:

```
df_train['FamilySize'].hist()
plt.title('Family Size Histogram')
```

Out[26]:

```
<matplotlib.text.Text at 0x7fd3ac097080>
```

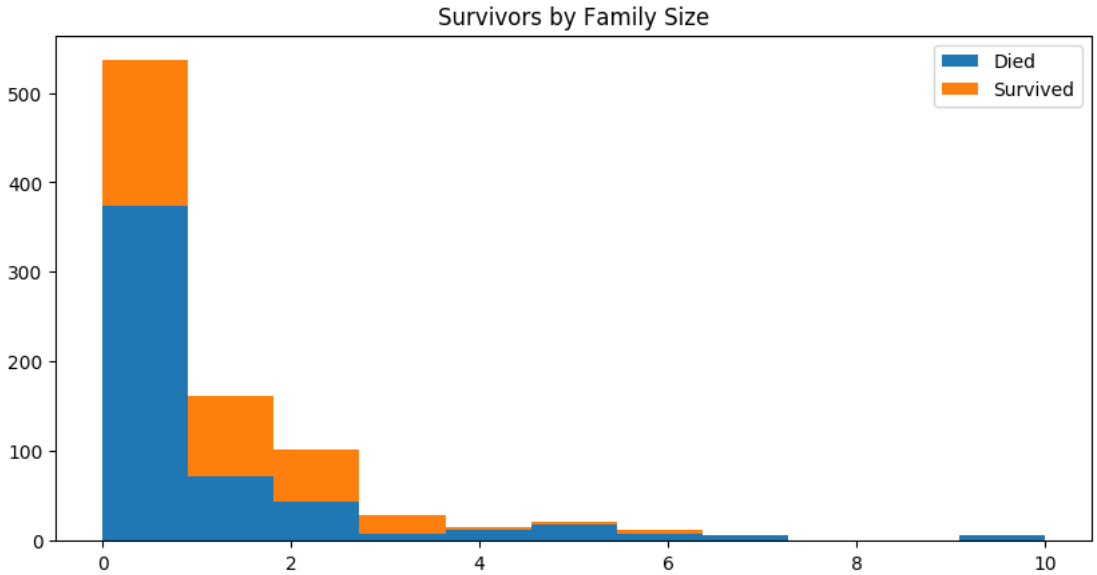


In [27]:

```
family_sizes = sorted(df_train['FamilySize'].unique())
family_size_max = max(family_sizes)

df1 = df_train[df_train['Survived'] == 0]['FamilySize']
df2 = df_train[df_train['Survived'] == 1]['FamilySize']
plt.hist([df1, df2],
         bins=family_size_max + 1,
         range=(0, family_size_max),
         stacked=True)
plt.legend(('Died', 'Survived'), loc='best')
plt.title('Survivors by Family Size')
```

Out[27]:

```
<matplotlib.text.Text at 0x7fd39821bbe0>
```



<font size=3>　　基于直方图，家庭人口数量对生存的影响并不明显。

随机森林¶

In [28]:

```
df_train.dtypes[df_train.dtypes.map(lambda x: x == 'object')]
df_train = df_train.drop(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'],
                         axis=1)
df_train = df_train.drop(['Age', 'SibSp', 'Parch', 'PassengerId', 'Embarked_Val'], axis=1)
train_data = df_train.values
```

In [29]:

```
def clean_data(df, drop_passenger_id):

    # Get the unique values of Sex
    sexes = sorted(df['Sex'].unique())

    # Generate a mapping of Sex from a string to a number representation
    genders_mapping = dict(zip(sexes, range(0, len(sexes) + 1)))

    # Transform Sex from a string to a number representation
    df['Sex_Val'] = df['Sex'].map(genders_mapping).astype(int)

    # Get the unique values of Embarked
    embarked_locs = sorted(df['Embarked'].unique())

    # Generate a mapping of Embarked from a string to a number representation
    embarked_locs_mapping = dict(zip(embarked_locs,
                                     range(0, len(embarked_locs) + 1)))

    # Transform Embarked from a string to dummy variables
    df = pd.concat([df, pd.get_dummies(df['Embarked'], prefix='Embarked_Val')], axis=1)

    # Fill in missing values of Embarked
    # Since the vast majority of passengers embarked in 'S': 3,
    # we assign the missing values in Embarked to 'S':
    if len(df[df['Embarked'].isnull()] > 0):
        df.replace({'Embarked_Val' :
                    { embarked_locs_mapping[nan] : embarked_locs_mapping['S']
                    }
                },
                inplace=True)

    # Fill in missing values of Fare with the average Fare
    if len(df[df['Fare'].isnull()] > 0):
        avg_fare = df['Fare'].mean()
        df.replace({ None: avg_fare }, inplace=True)

    # To keep Age in tact, make a copy of it called AgeFill
    # that we will use to fill in the missing ages:
    df['AgeFill'] = df['Age']

    # Determine the Age typical for each passenger class by Sex_Val.
    # We'll use the median instead of the mean because the Age
    # histogram seems to be right skewed.
    df['AgeFill'] = df['AgeFill'] \
                        .groupby([df['Sex_Val'], df['Pclass']]) \
                        .apply(lambda x: x.fillna(x.median()))

    # Define a new feature FamilySize that is the sum of
    # Parch (number of parents or children on board) and
    # SibSp (number of siblings or spouses):
    df['FamilySize'] = df['SibSp'] + df['Parch']

    # Drop the columns we won't use:
    df = df.drop(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], axis=1)

    # Drop the Age column since we will be using the AgeFill column instead.
    # Drop the SibSp and Parch columns since we will be using FamilySize.
    # Drop the PassengerId column since it won't be used as a feature.
    df = df.drop(['Age', 'SibSp', 'Parch'], axis=1)

    if drop_passenger_id:
        df = df.drop(['PassengerId'], axis=1)

    return df
```

In [30]:

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100)
```

In [31]:

```
# 构建模型的特征
train_features = train_data[:, 1:]

# '目标标签
train_target = train_data[:, 0]

# 训练模型
clf = clf.fit(train_features, train_target)
score = clf.score(train_features, train_target)
"Mean accuracy of Random Forest: {0}".format(score)
```

Out[31]:

```
'Mean accuracy of Random Forest: 0.9809203142536476'
```

In [33]:

```
# 检验模型
df_test = pd.read_csv('test.csv')
df_test.head()
```

Out[33]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

In [34]:

```
df_test = clean_data(df_test, drop_passenger_id=False)
test_data = df_test.values
```

In [35]:

```
test_x = test_data[:, 1:]

# 预测
test_y = clf.predict(test_x)
```

In [36]:

```
df_test['Survived'] = test_y
df_test[['PassengerId', 'Survived']] \
    .to_csv('results-rf.csv', index=False)
```

评估模型的准确性¶

In [37]:

```
from sklearn import metrics
from sklearn.cross_validation import train_test_split

# 分为8-2两部分
train_x, test_x, train_y, test_y = train_test_split(train_features,
                                                    train_target,
                                                    test_size=0.20,
                                                    random_state=0)
print (train_features.shape, train_target.shape)
print (train_x.shape, train_y.shape)
print (test_x.shape, test_y.shape)
```

```
(891, 8) (891,)
(712, 8) (712,)
(179, 8) (179,)
```

```
/srv/env/lib64/python3.4/site-packages/sklearn/cross_validation.py:44: DeprecationWarning: This module was deprecated in version 0.18
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [38]:

```
clf = clf.fit(train_x, train_y)
predict_y = clf.predict(test_x)

from sklearn.metrics import accuracy_score
print ("Accuracy = %.2f" % (accuracy_score(test_y, predict_y)))
```

```
Accuracy = 0.84
```

In [41]:

```
model_score = clf.score(test_x, test_y)
print ("Model Score %.2f \n" % (model_score))

confusion_matrix = metrics.confusion_matrix(test_y, predict_y)
print ("Confusion Matrix ", confusion_matrix)

print ("          Predicted")
print ("         |  0  |  1  |")
print ("         |-----|-----|")
print ("       0 | %3d | %3d |" % (confusion_matrix[0, 0],
                                   confusion_matrix[0, 1]))
print ("Actual   |-----|-----|")
print ("       1 | %3d | %3d |" % (confusion_matrix[1, 0],
                                   confusion_matrix[1, 1]))
print ("         |-----|-----|")
```

```
Model Score 0.84

Confusion Matrix  [[101   9]
 [ 20  49]]
          Predicted
         |  0  |  1  |
         |-----|-----|
       0 | 101 |   9 |
Actual   |-----|-----|
       1 |  20 |  49 |
         |-----|-----|
```

In [43]:

```
# 模型详细评估
from sklearn.metrics import classification_report
print(classification_report(test_y,
                            predict_y,
                            target_names=['Not Survived', 'Survived']))
```

```
             precision    recall  f1-score   support

Not Survived       0.83      0.92      0.87       110
    Survived       0.84      0.71      0.77        69

 avg / total       0.84      0.84      0.83       179
```

In [ ]:

```

```

数学教学系列----ARMA模型

自回归滑动平均模型（ARMA 模型，Auto-Regressive and Moving Average Model）是研究时间序列的重要方法，由自回归模型（简称AR模型）与滑动平均模型（简称MA模型）为基础“混合”构成。

由于只有对平稳的时间序列才能建立ARMA模型，因此在建立模型之前，有必要对序列进行预处理，主要包括了平稳性检验和纯随机检验。

时间序列检验是平稳的，且是非白噪声序列，则可以建立模型。建立模型之前需要识别模型阶数即确定阶数。阶数确定要借助于时间序列的相关图，即序列的自相关函数和偏自相关函数，并根据他们之间的理论模式进行阶数最后的确定。

自相关系数	偏相关系数	模型定阶
拖尾	p阶截尾	AR(p)模型
q阶截尾	拖尾	MA(q)模型
拖尾	截尾	ARMA(p,q)模型

然后进行参数估计，检验是否具有统计学特性。

假设检验，判断残差序列是否是白噪音序列。

最后，使用得到的模型进行预测。

<font size=3 face="黑体"> ARMA的基本思想是把AR模型和MA模型的想法结合在一个紧凑的形式中，使所用的参数个数保持很小。在收益率序列中，直接使用ARMA模型的机会很少。然而，ARMA模型的概念与波动率建模有密切的关系。事实上，广义自回归条件异方差（GARCH）模型就可以认为使对${\{a_{t}^2\}}$的ARMA模型，尽管是非标准的。
称一个时间序列r_t服从ARMA模型，则r_t满足

$$r_t - \phi_1 r_{t-1} = \phi_0 + a_t - \theta_1 a_{t-1}$$

其中$\{a_t\}$是白噪音序列。上式左边式模型的AR部分，右边是MA部分。常数项为ϕ_0。为使这样的一个模型有意义，要求$\phi_1 \neq \theta_1$。否则，在方程两端消去一个因子，方程所决定的过程就变成了一个白噪音序列。

1 ARMA(1,1)模型的性质¶

<font size=3 face="黑体"> ARM A(1,1)模型的性质是AR(1)模型的相应性质的推广，只是作一些小的修改来处理MA(1)部分的影响．首先讨论平稳性条件．在(2.25)式两端取期望，得到

$$E(r_t) - \phi_1 E(r_{t-1}) = \phi_0 + E(a_t) - \theta_1 E(a_{t-1})$$

因为对所有的i都有$E(a_i)=0$，所以只要序列是弱平稳的，则r_i的均值为

$$E(r_t) = \mu = \frac{\phi_0}{1-\phi_1}.$$

此结果和模型的结果完全一样.

为简单起见，假定 $\phi_0=0$.下面我们考虑r_t的自协方差函数．首先，在模型两端乘以a_t再取期望，我们有

$$E(r_t a_t) = E(a_t^2) - \theta_1 E(a_t a_{t-1}) = E(a_t^2) = \sigma_a^2,$$

把模型改写成

$$r_t = \phi_1 r_{t-1} + a_t - \phi_1 a_{t-1}.$$

在上式两端取方差，得到

$$Var(r_t) = \phi_1^2 Var(r_{t-1}) + \sigma_a^2 + \theta_1^2 \sigma_a^2 - 2\phi_1\theta_1 E(r_{t-1}a_{t-1})$$

这里用到$r_{t-1} $ 与a：不相关这一事实：我们得到

$$Var(r_t) - \phi_1^2 Var(r_{t-1}) = (1 - 2\phi_1\theta_1 + \theta_1^2))\sigma_a^2$$

从而，若序列r_i是弱平稳的，则$Var(r_t)=Var(r_{t1})$，且

$$Var(r_t) = \frac{(1 - 2\phi_1\theta_1 + \theta_1^2))\sigma_a^2}{1-\phi_1^2}$$

因为方差是正的，故要求 $\phi_1^2 < 1$（也即$|\phi_1|<1$).这又与AR(1)模型的平稳性条件一样了.

为了得到r_t的自协方差函数，我们假定$\phi_0=0$，并在上式两端乘以r_{t-1},得到

$$r_t R_{t-l} - \phi_1 r_{t-1} r_{t-l} = a_t r_{t-l} - \theta_1 a_{t-1} r_{t-l}.$$

对$l=1$,在上式两端取期望并利用$t-1$时的（2.26）式，我们有

$$\gamma_1 - \phi_1\gamma_0 = -\phi_1\sigma_a^2,$$

其中$\gamma_1=Cov(r_i,r_{t-l})$.这个结果不同于AR(1)情形，对AR(1)模型有$\gamma_1-\phi_1\gamma_0=0$.然而，对$l=2$，取期望后得到

$$\gamma_2 - \phi_1\gamma_1 = 0,$$

这与AR(1)情形一样．事实上，用相同的方法可得到

$$\eta - \phi_1\eta_{-1} = 0, l > 1.$$

对于ACF，上述结果表明：对平稳ARMA(1.1）模型，有

$$\rho_1 = \phi_1 - \frac{\theta_1\sigma_a^2}{\gamma_0},\ \rho_l = \phi_1\rho_{t-l}, l > 1$$

这样，ARMA(1, 1)模型的ACF 很像AR(1)模型的ACF，不同之处仅在于它的指数衰减是从间隔2开始的.因此，ARMA(1,1)模型的ACF不能在任意有限间隔后截尾.

现在来看偏自相关函数(PACF)．可以证明：ARMA(1,1)模型的PACF也不能在有限间隔后截尾．它与MA(1)模型的PACF表现很相似，只是指数衰减从间隔2开始，而不是从间隔1开始.

综上所述，ARMA(1,1)模型的平稳性条件与AR(1)模型的相同，ARMA(1,1)模型的ACF与AR(1)模型的ACF形式相似，只是这种形式从间隔2处开始.

2　一般的ARMA模型

<font size=3 face="黑体">　一般的$ARMA(p,q)$模型的形式为

$$r_t = \phi_0 + \sum_{i=1}^{p}\phi_i r_{t-i} + a_i - \sum_{i=1}^{q}\theta_i a_{t-i},$$

其中a_t是白噪声序列，p和q都是非负整数，AR和MA模型是ARMA(p, q)的特殊情形．利用向后推移算子，上述模型可写成

$$(1 - \phi_1 B - \cdots - \phi_p B^p)r_t = \phi_0 + (1 - \theta_1 B - \cdots - \theta_q B^q)a_t$$

模型的AR多项式是$1-\phi_1B-\cdots-\phi_pB^p$，MA多项式是$1-\theta_1B-\cdots-\theta_qB^q$. 我们要求AR多项式和MA多项式没有公因子,否则模型的阶(p,q)会降低.如 AR模型一样，AR多项式引进了ARMA模型的特征方程．如果特征方程所有根的绝对值都小于1,则该ARMA模型是弱平稳的．这时，模型的无条件均值为

$E(r_t)=\phi_0 / (1-\phi_1-\cdots-\phi_p).$

3　用ARMA模型进行预测

<font size=3 face="黑体">　和ACF一样，只要将MA部分对低步数预测的影响进行调整后，ARMA(p,q)的模型的预测就会与AR(p)模型的预测有相似的特征．设预测原点为h, F_h为在h 时刻所能得到的信息集合.r_{h+1}的向前一步预测为

$$\hat{r}_h(1) = E(r_{h+1}|F_h) = \phi_0 + \sum_{i=1}^{p}\phi_i r_{h+1-i} - \sum_{i=1}^{p}\theta_i a_{h+1-i},$$

相应的预测误差为$e_h(1)=r_{h+1}-\hat{r}_h(1)=a_{h+1}$.向前一步预测误差的方差为$Var[e_h(1)]=\sigma_a^2$.对向前$l$步预测，我们有

$$\hat{r}_h(l) = E(r_{h+1}|F_h) = \phi_0 + \sum_{i=1}^{p}\phi_i \hat{r}_h(l-i) - \sum_{i=1}^{p}\theta_i a_h(l-i),$$

其中，当$l-i\leq0$时，$\hat{r}_h(l-i)=r_{h+l-i}$;当$l-i>0$时，$a_h (l—i) = 0$；当$l-i\leq0$ 时$a_h(l-i)=a_{h+l-i}$.这样，ARMA模型的向前多步预测可以递推算得.相应的预测误差为

$$e_h(l) = r_{h+l} - \hat{r}_h(l)$$

4 ARMA模型的三种表示¶

本节将简单地讨论平稳ARMA(p,q)的模型的三种表示．这三种表示用于三种不同的目的．了解这三种表示会更好地理解ARMA模型‘第一种表示是（2.28)式，这个表示很紧凑并且在参数估计时很有用，另外，它也可用于递推计算r_t的向前多步预测，见2.6.4节的讨论．　对另外两种表示，我们用两个多项式比的级数展开式（长除法）．给定两个多项式$\phi(B)=1-\sum_{i=1}^{p}\phi_iB^i$和$\theta(B)=1-\sum^{q}_{i}\theta_iB^i$我们有

$$\frac{\theta(B)}{\phi(B)} = 1 + \psi_1 B + \psi_2 B^2 + \cdots \equiv \psi(B),$$

$$\frac{\theta(B)}{\phi(B)} = 1 - \pi_1 B - \pi_2 B^2 - \cdots \equiv \pi(B),$$

例如，若$\psi(B)=1-\phi_1B,\theta(B)=1-\theta_1B,$则

$$\psi(B) = \frac{1-\theta_1 B}{1-\phi_1 B} = 1 + (\phi_1 - \theta_1)B + \phi_1(\phi_1 - \theta_1)B^2 + \phi_1^2(\phi_1 - \theta_1)B^3 + \cdots,$$

$$\pi(B) = \frac{1-\phi_1 B}{1-\theta_1 B} = 1 - (\phi_1 - \theta_1)B - \theta_1(\phi_1 - \theta_1)B^2 - \theta_1^2(\phi_1 - \theta_1)B^2 - \cdots,$$

由定义知$\psi(B)\pi(B)=1$，利用$Bc=c$对任意常数c成立这个事实（因为常数是随时间不变的）,我们有

$$\frac{\phi_0}{\theta(1)} = \frac{\phi_0}{1-\theta_1-\cdots-\theta_q}, \frac{\phi_0}{\phi(1)} = \frac{\phi_0}{1-\phi_1-\cdots-\phi_q}.$$

AR表示¶

利用上式的结果，ARMA$(p，q)$模型可写成

$$r_t = \frac{\phi_0}{1-\theta_1-\cdots-\theta_q} + \pi_1 r_{t-1} + \pi_2 r_{t-2} + \pi_3 r_{t-3} + \cdots + a_t$$

这个表示给出了当前收益率r_i对过去收益率$r_{t-p},i>0$的依赖关系．系数$\{\pi_i\}$称

为ARMA模型的$\pi-权重$．为了说明延迟值r_{t-i}对r_i的贡献随i的增大而逐渐消

失,系数π_i应随i增大而趋于零，一个ARMA$(p，q)$模型如果具有这样的性质，则称它为$可逆的$，对纯AR模型,$\theta(B)=1$.故$\pi(B)=\phi(B)$这是一个有限阶的多项式．从而对$i>p$有$\pi_i=0$，模型是可逆的.对其他ARMA模型，可逆性的充分条件是：多项式$\theta(B)$的所有零点的模大于1.例如，对MA(1)模型$r_t=(1-\theta_1B)a_t$,一次多项式$1-\theta_1B$的零点是$B=1/\theta_1$.从而，如果$1/|\theta_1|>1$（也即$|\theta_1|<1$),则MA(1)是可逆的.

用AR表示，一个可逆的ARMA(p,q)序列r_t是当前的“抖动”a_t与序列过去值的加权平均的线性组合．对越来越远的过去值，权重呈指数衰减．

MA表示

同样，ARMA$(p，q)$模型也能写成

$$r_t = \mu + a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + \cdots = \mu + \psi(B) a_t$$

其中$\mu=E(r_t)=\phi_0/(1-\phi_1-\cdots-\phi_p)$.这个表示清楚地说明了过去的“扰动"$a_{t-i}(i>0)$对当前收益$r_t$的影响，系数$\{\psi_i\}$称为该ARMA模型的脉冲响应函数 （impulse response function） .对弱平稳序列，系数ψ_i随i的增加呈指数衰减.这一点是可以理解的，因为扰动a_{t-i}对收益率r_t的影响应该随时间而消失.这样，对平稳ARMA模型，扰动a_{t-i}不能对序列有永久的影响.如果$\psi_0\neq0$，这样的MA表示中有一个常数项，它就是r_t的均值（也即$\psi_0/(1-\psi_1-\cdots-\psi_p)$.MA表示在计算预测误差的方差时也是有用的、在预测原点h，我们有$a_h,a_{h-1},\cdots$从而，向前l步预测为

$$\hat{r}_h(l) = \mu + \psi_l a_h + \psi_{l+1} a_{h-1} + \cdots,$$

相应预测误差为

$$e_h(l) = a_{h+l} + \psi_1 a_{h+l-1} + \cdots + \psi_{l-1} a_{h+1},$$

因此，向前/步预测误差的方差为

$$Var[e_h(l)] = (1 + \psi_1^2 + \cdots + \psi_{l-1}^2)\sigma_a^2$$

正如所料，它是预测时间长度l的非减函数.

最后，由MA表示还提供了平稳序列均值回转的一个简单证明.平稳性意味着当$i\to\infty$时ψ_i趋于零.从而，由MA表示，我们有：当$i\to\infty$时，$\hat{r}_h(l)\to\mu$因为$\hat{r}_h(l)$是r_{h+l}在预测原点h的条件期望,上述结果表示，从长期来看，收益率序列预期会趋于它的均值，也就是说，序列是均值回转的.进一步地，由MA表示，我们有$Var(r_t)=(1+\sum_{i=1}^{\infty}\psi_i^2)\sigma_a^2.$从而，当$l\to\infty$时，$Var[e_h(l)]\to Var(r_t)$.$\hat{r}_h(l)$趋于$\mu$的速度决定了均值回转的速度.

In [ ]:

数学教学系列----拉格朗日对偶问题

在优化理论中，目标函数 f(x) 会有多种形式：如果目标函数和约束条件都为变量 xx 的线性函数, 称该问题为线性规划； 如果目标函数为二次函数, 约束条件为线性函数, 称该最优化问题为二次规划; 如果目标函数或者约束条件均为非线性函数, 称该最优化问题为非线性规划。每个线性规划问题都有一个与之对应的对偶问题，对偶问题有非常良好的性质，以下列举几个：

1. 对偶问题的对偶是原问题；
2. 无论原始问题是否是凸的，对偶问题都是凸优化问题；
3. 对偶问题可以给出原始问题一个下界；
4. 当满足一定条件时，原始问题与对偶问题的解是完全等价的；

求取有约束条件的优化问题时，拉格朗日乘子法（Lagrange Multiplier) 和KKT条件是非常重要的两个求取方法，对于等式约束的优化问题，可以应用拉格朗日乘子法去求取最优值；如果含有不等式约束，可以应用KKT条件去求取。当然，这两个方法求得的结果只是必要条件，只有当是凸函数的情况下，才能保证是充分必要条件。KKT条件是拉格朗日乘子法的泛化。下文，我们将详细说明拉格朗日对偶问题和原问题的联系。

拉格朗日对偶性¶

<font size=3 face="黑体">

$\ \ $在约束最优化问题中,常常利用拉格朗日对偶性（Lagrange duality）将原始问题转换为对偶问题,通过解对偶问题而得到原始问题的解.该方法应用在许多统计学习方法中,例如，最大熵模型与支持向量机.这里简要叙述拉格朗日对偶性的主要概念和结果.

1.原始问题¶

$\ \ $假设$f(x),c_i(x),h_j(x)$是定义在$R^n$上的连续可微函数.考虑约束最优化问题

$$\min_{\propto R^+} f(x) \tag{C.1}$$

$$s.t. \quad c_i(x) \leq 0, \quad i = 1, 2, \cdots, k \tag{C.2}$$

$$h_j(x) = 0, \quad j = 1, 2, \cdots, l \tag{C.3}$$

称此约束最优化问题为原始最优化问题或原始问题.

首先,引进广义拉格朗日函数(generalized Lagrange function)

$$L(x,\alpha,\beta)=f(x)+\sum_{i=1}^{k}\alpha_i c_i(x)+\sum_{j=1}^{l}\beta_j h_j(x) \tag{C.4}$$

这里,$x=(x^(1),x^(2),\cdots,x^(n))^T\in R^n,\alpha_i,\beta_j$是拉格朗日乘子.$a_i\geq0$考虑$x$的函数:

$$\theta_P(x)=\max_{\alpha,beta,\alpha_i\geq0}L(x,\alpha,\beta) \tag{C.5}$$

这里，下标P表示原始问题.

$\ \ $假设给定某个$x$.如果$x$违反原始问题的约束条件,即存在某个$i$使得$c_i(w)>0$,或者存在某个$j$使得$h_j(w)\neq0$,那么就有

$$\theta_P(x)=\max_{\alpha,\beta,\alpha_i\geq)}[f(x)+\sum_{j=1}^{t}\beta_j h_j(x)]=+\infty \tag{C.6}$$

因为若某个i使约束$c_i(x)>0$,则可令$\alpha_i\to +\infty$,若某个j使$h_j(x)\neq0$,可令使在β_j,使$\beta_jh_j(x)\to +\infty$,而将其余各α_i,β_j均取为0.

$\ \ $相反地,如果满足约束条件式(C.2)和式(C.3),则由式(C.5)和式(C.4)可知,$\theta_P(x)=f(x)$.因此,

$$\theta_P(x)=\begin{cases}f(x), & \text{x满足原始问题约束}\\ +\infty, & \text{其他}\end{cases} \tag{C.7}$$

所以如果考虑极小化问题

$$\min_x\theta_P(x)=\min_x\max_{\alpha,\beta,\alpha}L(x,\alpha,\beta) \tag{C.8}$$

它是与原始最优化问题$(C.1)\sim(C.3)$等价的，即它们有相同的解．问题$\min_{x}\max_{\alpha,\beta,\alpha}L(x,\alpha,\beta) $，称为广义拉格朗日函数的极小极大问题．这样一来，就把原始最优化问题表示为广义拉格朗日函数的极小极大问题．为了方便，定义原始问题的最优值

$$p^*=\min_x\theta_P(x) \tag{C.9}$$

称为原始问题的值．

2．对偶问题¶

$\ \ $定义

$$\theta_D(\alpha,\beta)=\min_x L(x,\alpha,\beta) \tag{C.10}$$

再考虑极大化$\theta_D(\alpha,\beta)=\min_{x}L(x,\alpha,\beta)$，即

$$\max_{\alpha,\beta,\alpha_i\geq0}\theta_D(\alpha,\beta)=\max_{\alpha,\beta,\alpha_i\geq0}\max_x L(x,\alpha,\beta) \tag{C.11}$$

问题$\max_{\alpha,\beta,\alpha_i\geq0}\max_{x}L(x,\alpha,\beta)$称为广义拉格朗日函数的极大极小问题．

$\ \ $可以将广义拉格朗日函数的极大极小问题表示为约束最优化问题：

$$\max_{\alpha,\beta}\theta_D(\alpha,\beta)=\max_{\alpha,\beta}\min_x L(x,\alpha,\beta) \tag{C.12}$$

$$s.t.\quad \alpha_i\geq0,\quad i=1,2,\cdots,k \tag{C.13}$$

称为原始问题的对偶问题．定义对偶问题的最优值

$$d^*=\max_{\alpha,\beta,\alpha_i\geq0}\theta_D(x,\alpha,\beta) \tag{C.14}$$

称为对偶问题的值.

3.原始问题和对偶问题的关系¶

下面讨论原始问题和对偶问题的关系．

定理C.若原始问题和对偶问题都有最优值，则

$$d^*=\max_{\alpha,\beta,\alpha_i\geq0}\min_x L(x,\alpha,\beta)\leq\min_x\max_{\alpha,\beta,\alpha_i\geq0}L(x,\alpha,\beta)=p^* \tag{C.15}$$

证明 $\ \ \ $ 由式（C.12）和式（C.5）．对任意的α,β,x，有

$$\theta_D(\alpha,\beta)=\min_x L(x,\alpha,\beta)\leq\max_{\alpha,\beta,\alpha_i\geq0}L(x,\alpha,\beta)=\theta_P(x) \tag{C.16}$$

即

$$\theta_D(\alpha,\beta)\leq\theta_P(x) \tag{C.17}$$

由于原始问题和对偶问题均有最优值，所以，

$$d^* = \max_{\alpha,\beta,\alpha_i \geq 0} \min_x L(x,\alpha,\beta) \leq \min_x \max_{\alpha,\beta,\alpha_i \geq 0} L(x,\alpha,\beta) = p^* \tag{C.19}$$

$\ \ \ $推论C.1$\ \ $设$x^*$和$\alpha^*,\beta^*$ 分别是原始问题(C.1)$\sim$(C.3)和对偶问题(C.12)$\sim$(C.13)的可行解，并且$d^*=p^*$,则x^*和α^*,β^*分别是原始问题和对偶问题的最优解.

$\ \ $在某些问题下去，原始问题和对偶问题的最优值相等，$d^*=p^*$．这时可以用解对偶问题替代解原始问题．下面以定理的形式叙述有关的重要结论而不予证明．
$\ \ $定理C.2$\ \ $考虑原始问题(C.1)$\sim$(C.3)和对偶问题(C.12)$\sim$(C.13)．假设函数$f(x)$和$c_i(x)$是凸函数，$h_j(x)$是仿射函数;并且假设不等式约束$c_i(x)$是严格可行的，即存在$x$，对所有$i$有$c_i(x)<0$，则存在$x^*$,$\alpha^*,\beta^*$,使$x^*$是对偶问题的解，并且

$$p^* = d^* = L(x^*,\alpha^*,\beta^*) \tag{C.20}$$

$\ \ $定理C.3，对原始问题(C.1)$\sim$(C.3)和对偶问题(C.12)$\sim$(C.13),，假设函数$f(x)$和$c_i(x)$是凸函数，$h_i(x)$是仿射函数 ，并且不等式约束$c_i(x)$是严格可行的，则$x^*$和$\alpha^*,\beta^*$分别是原始问题和对偶问题的解的充分必要条件是$x^*$,$\alpha^*,\beta^*$下面的Kanish-Kuhn-Tucker (KKT)条件：

$$\nabla_x L(x^*,\alpha^*,\beta^*) = 0 \tag{C.21}$$

$$\nabla_\alpha L(x^*,\alpha^*,\beta^*) = 0 \tag{C.22}$$

$$\nabla_\beta L(x^*,\alpha^*,\beta^*) = 0 \tag{C23}$$

$$a_i^* c_i(x^*) = 0, \quad i = 1,2,\cdots,k \tag{C.24}$$

$$c_i(x^*) \leq 0, \quad i = 1,2,\cdots,k \tag{C.25}$$

$$a_i^* \geq 0, \quad i = 1,2,\cdots,k \tag{C.26}$$

$$h_j(x^*) = 0, \quad j = 1,2,\cdots,l \tag{C.27}$$

$\ \ $特别指出，式(C.24)称为KKT的对偶互补条件，由此条件可知：若$a^*_i>0,$则$c_i(x^*)=0$.

In [ ]:

数学教学系列----梯度下降法

梯度下降法是一个一阶最优化算法，利用负梯度方向来决定每次迭代的新的搜索方向，使得每次迭代能使待优化的目标函数逐步减小。梯度下降法是2范数下的最速下降法。

向量微积分中,标量场的梯度是一个向量场。标量场中某一点上的梯度指向标量场增长最快的方向,梯度的长度是这个最大的变化率。

我们前一个系列介绍了机器学习的相关内容，梯度下降法相对牛顿法求取海瑟矩阵，耗费的内存较小，优化前期收敛速度较快得到了广泛的利用。

梯度下降法算法简介¶

　梯度下降法（gradient descent）或最速下降法（steepest decent）使求解无约束最优化的一种最常用的方法，有实现简单的优点。梯度下降法是迭代法，每一步需要求解目标函数的梯度向量。
　假设$f(x)$是R^n上具有一阶连续偏导数的函数。要求解的无约束最优化问题是

$$min f(x)$$

x^*表示目标函数$f(x)$的极小点。
　梯度下降法是一种迭代算法。选取适当的初值$x^{(0)}$，不断迭代，更新x的值，进行目标函数的极小化，直到收敛。由于负梯度方向是使函数值下降最快的方向，在迭代的每一步，以负梯度方向更新x的值，从而达到减少函数值的目的。
　由于$f(x)$具有一阶连续偏导数，若第k次迭代值为$x^{(k)}$,则可将$f(x)$在$x^{(k)}$附近进行一阶泰勒展开：

$$f(x) = f(x^{(k)}) + g_k^T (x - x^{(k)})$$

这里，$g_k=g(x^{(k)})=\triangledown{f(x^{(k)})}$为$f(x)$在$x^{(k)}$的梯度。
　求出第$k+1$次迭代值$x^{(k+1)}$：

$$x^{(k+1)} \leftarrow x^{(k)} + \lambda_k p_k$$

其中，p_k使搜索方向，取负梯度方向$p_k = -\triangledown{f(x^{(k)})}$,$\lambda_k$是步长，由一维搜索确定，即$\lambda_k$使得

$$f(x^{(k)} + \lambda_k p_k) = min(f(x^{(k)} + \lambda p_k), \lambda \succeq 0$$

梯度下降算法如下：
输入：目标函数$f(x)$，梯度函数$g(x) = \triangledown {f(x)}$,计算精度$\epsilon$；
输出：$f(x)$的极小值点x^*.

（1）取初值$x^{(0)}\in R^n$，值$k=0$
（2）计算$f(x^{(k)})$
（3）计算梯度$g_k = g(x^{(k)})$，当$||g_k|| < \epsilon$时，停止迭代，令$x^* = x^{(k)}$；否则，令$p_k = -g(x^{(k)})$，求λ_k,使

$$f(x^{(k)} + \lambda_k p_k) = min(f(x^{(k)} + \lambda p_k), \lambda \succeq 0$$

（4）置$x^{(k+1)}= x^{(k)}+\lambda_kp_k$，计算$f(x^{(k+1)})$,当$||f(x^{(k+1)})-f(x^{(k)})< \epsilon|| $时，停止迭代，令$x^* = x^{(k+1)}$。
（5）否则，置$k = k+1$，转（3）
当目标函数时凸函数，梯度下降法的解时全局最优解，一般情况下，其解不保证全局最优。梯度下降法的收敛速度也未必是很快的。

梯度下降方向为什么是下降最快的方向¶

　设$f(x,y)$为二元函数，$u = cos\theta i + sin \theta j$为一个单位向量，如果下列极限存在:
$\lim\limits_{t\to 0}\frac{f(x_0+tcos\theta,y_0+tsin\theta)-f(x_0,y_0)}{t}$次方向记为$D_uf$，则称这个极限值是$f$沿着$u$方向的方向导数，那么随着$\theta$ 的不同，我们可以求出任意方向的方向导数.这也表明了方向导数的用处，是为了给我们考虑函数对任意方向的变化率.
　在求方向导数的时候，除了用上面的定义法求之外，我们还可以用偏微分来简化我们的计算.表达式是:

$$D_uf(x,y) = f_x(x,y)cos\theta + f_y(x,y)sin\theta$$

　求解函数延那个方向变化率最大：$D_uf(x,y) = f_x(x,y)cos \theta + f_y(x,y)sin \theta$，设$A = (f_x(x,y),f_y(x,y))$，$I = (cos\theta,sin \theta)$，可以得到$D_uf(x,y) = A \centerdot I = |A|*|I|*con \alpha$,$\alpha$为向量$A$和向量$I$之间的夹角。
　那么此时如果$D_{u}f(x,y)$要取得最大值，也就是当$\alpha $为0度的时候，也就是向量$I$（这个方向是一直在变，在寻找一个函数变化最快的方向）与向量A（这个方向当点固定下来的时候，它就是固定的）平行的时候，方向导数最大.方向导数最大，也就是单位步长，函数值朝这个反向变化最快.

数学教学系列----B-S期权定价模型

　期权定价模型基于对冲证券组合的思想。投资者可建立期权与其标的股票的组合来保证报酬。在均衡时，此报酬得到无风险利率。

假设条件：

1、股价遵循几何布朗分布

2、股票交易连续进行

3、不存在交易费用及税收

4、允许卖空，且可利用所有卖空所得

5、再衍生品有效期间，股票不支付股利

6、在衍生品有效期间，无风险利率保持不变

7、所有无风险套利机会均被消除

影响欧式看涨期权价格的因素：

1、当股价越高，期权价格越高

2、到期执行价格越高，期权价格越低

3、距离到期日时间越长，期权价格越高

4、股价波动率越大，期权价格越高

5、无风险利率越高，期权价格越高

B-S定价公式¶

　Black和Scholes(1973)成功地得到了对欧式看涨期权与看跌期权价格的精确公式.下面我们利用风险中性定价来推导这些公式.B-S定价公式如下：$C = e^{-\delta t}S \times N(d_1) - e^{rt}K \times N(d_2)$

1 风险中性世界

　将漂移参数μ从B-S微分方程中去掉、在金融中这意味着此方程是与风险偏好独立的.换句话说,风险偏好不能影响这个方程的解.此性质的一个完美结果就是能够假设投资者是风险中性的.在一个风险中性世界里,我们有如下结论:

- 所有证券的期望收益率都是无风险利率r;
- 任何现金流的当前价值可以通过将它的期望价值以无风险利率折现得到.

2 公式

　在风险中性世界里,一个欧式看涨期权在到期日的期望价值为

$$E_*[max(P_T-K,0)],$$

其中E_*表示在无风险世界中的期望价值.看涨期权在t时刻的价格为

$$c_t = exp[-r(T-t)]E_*[max(P_T-K,0)]$$

然而,在风险中性世界里,我们有$\mu=r$,ln(P_T)是正态分布的:

$$In(P_T) \sim N[ln(P_t)+(r-\frac{\sigma^2}{2})(T-t),\sigma^2(T-t))].$$

令$g(P_T)$表示P_T的概率密度函数,则方程中看涨期权的价格为

$$c_t = exp[-r(T-t)]\int_K^\infty (P_T-K)g(P_T)dP_T.$$

通过积分的变量变换以及一些代数计算(附录A中给出详细推导),我们有

$$c_t = P_t\Phi(h_+)-Kexp[-r(T-t)\Phi(h_-)]$$

其中$\Phi(x)$是标准正态随机变量的积累分布函数在x点的值,

$$h_+ = \frac{ln(P_t/K)+(r+\sigma^2/2)(T-t)}{\sigma\sqrt{T-t}},$$
$$h_- = \frac{ln(P_t/K)+(r-\sigma^2/2)(T-t)}{\sigma\sqrt{T-t}} = h_+-\sigma\sqrt{T-t}.$$

实际中,$\Phi(x)$可以很容易地通过大多数统计软件包得到.另外一种可供选择的方法,可以运用附录B中给出的一个近似. 方程中的B-S看涨期权公式有一些好的解释.首先,如果在到期日执行期权,得到了股票,但我们必须要支付敲定价格.这个交换只有当期权是赚钱的(即$P_T>K$)时才会发生.当且仅当$P_T>K$时,第一项$P_t\Phi(h_+)$)是得到股票的当前价值;当且仅当$P_T>K$时,第二项$-Kexp[-r(T-t)\Phi(h_-)]$是支付执行价格的当前价值.第二个解释尤其有用.B-S微分方程的推导中显示的,$\Phi(h_+)=\frac{\partial G_t}{\partial P_t}$是资产组合中不涉及不确定性和维纳过程的股份的数量.这个量就是套期保值交易中众所周知的Δ.我们知道$c_t=P_t\Phi(h_+)+B_t$,其中B_t为投资于资产组合(或衍生资产空头头寸)中的无风险债券的美元总量.可见从对B-S公式的检查中可直接看出$B_t=-Kexp[-r(T-t)]\Phi(h_-)$.公式的第一项$P_t\Phi(h_+)$为投资在股票上的总量,而第二项$Kexp[-r(T-t)]\Phi(h_-)$是借入的总量. 类似地,我们得到一个欧式看跌期权的价格为

$$p_t = Kexp[-r(T-t)]\Phi(-h_-)-P_t\Phi(-h_+)$$

因为标准正态分布是关于它的均值0.0对称的,所以我们有:对任何$x,\Phi(x)=1-\Phi(-x)$.利用这个性质,我们有$\Phi(-h_i)=1-\Phi(h_i)$.这样,计算一个看跌期权价格需要的信息与计算看涨期权价格所需要的信息是相同的.另外一个方法,利用正态分布的对称性,很容易证明

$$p_t-c_t = Kexp[-r(T-t)]-P_t$$

称为涨一跌平价公式,而且可用来从c_t中得到p_t.涨一跌平价公式也可以通过考虑下面两个组合来得到:

(1)组合A:一个欧式看涨期权加$Kexp[-r(T-t)]$的现金.

(2)组合B:一个欧式看跌期权加一股标的股票.

到期权的到期日这两个组合的盈利为

$$max(P_T,K)$$

由于期权在到期日才能执行,组合必须与现价具有相等的价值.这意味着

$$c_t+Kexp[-r(T-t)] = p_t+P_t,$$

这正是前面所给出的涨一跌平价公式.

例 假设Intel股票的当前价格是每股80美元,年波动率为$\sigma=20\%$,进一步假设年无风险利率为8%,那么执行价格为90美元,而且在3个月内到期的Intel的一个欧式看涨期权的价格是多少?

由假设,我们有$P_t=80,K=90,T-t=0.25,\sigma=0.2,且r=0.08$. 因此

$$h_+ = \frac{ln(80/90)+(0.08+0.04/2)\times 0.25}{0.2\sqrt{0.25}} = -0.927\,8,$$
$$h_- = h_+-0.2\sqrt{0.25} = -1.027\,8.$$

利用python,我们有

$$\Phi(-0.927\,8) = 0.1767, \Phi(-1.027\,8) = 0.152\,0,$$

因此,一个欧式看涨期权的价格为

$$c_t = \$80\Phi(-0,927\,8)-\$90\Phi(-1.027\,8)exp(-0.02) = \$0.73,$$

对看涨期权的买者而言,只有股价升高10.73美元时,才能达到得失平衡.

在相同的假设下,一个欧式看跌期权的价格为

$$p_t = \$90exp(-0.08\times 0.25)\Phi(1.027\,8)-\$80\Phi(0,927\,8) = \$8.95.$$

这样,对于看跌期权的买者而言,股价可以升高额外的1.05美元而达到得失平衡.

例 前面例子中的敲定价格大大超出了当前股价.一个更现实的敲定价格是81美元.假设前面例子中其他的条件仍然成立,现在我们有$P_t=80,K=81,r=0.08,T-

t=0.25$,且$h_i$变为

$$h_+ = \frac{In(80/81) + (0.08+0.04/2) \times 0.25}{0.2\sqrt{0.25}} = 0.125\ 775,$$
$$h_- = h_+ - 0.2\sqrt{0.25} = 0.025\ 775.$$

利用附录B中的近似,我们有$\Phi(0.125\ 775)=0.550\ 0$和$\Phi(0.025\ 775)=0.510\ 3$,则一个欧式看涨期权的价格是

$$c_t = \$80\Phi(0.125\ 775) - \$81exp(-0.02)\Phi(0.025\ 775) = \$3.49.$$

对于看涨期权的买者而言,股价必须提高4.49美元,才得失相等.从另一方面来讲,在同样假定下的一个欧式看跌期权的价格为

$$p_t = \$81exp(-0.02)\Phi(-0.025\ 775) - \$80\Phi(-0.125\ 775)$$
$$= 81exp(-0.02) \times 0.489\ 72 - \$80 \times 0.449\ 96 = \$2.89.$$

对看跌期权的买者而言,股票价格必须降低1.89美元,才得失平衡.

3 欧式期权的下界

<font size=3 face="黑体"> 考虑没有支付分红的股票的看涨期权,可以证明欧式看涨期权的价格满足

$$c_t \geq P_t - Kexp[-r(T-t)];$$

也就是说欧式看涨期权的下界是$P_t-Kexp[-r(T-t)]$.考虑如下两个组合可以验证该结果:

(1)组合A:一个欧式看涨期权加$K\exp[-r(T-t)]$的现金.

(2)组合B:一股标的股票.

对于组合A,如果将现金以无风险利率进行投资,则在T时刻的现金数量为K.如果$P_T>K$,则T时刻执行期权,组合的价值为P_T.如果$P_T例 假定$P_t=\$30,K=\$28,$年利率为$r=6%$, $T-t=0.5.$在这种情形下,

$$P_t - K\exp[-r(T-t)] = \$[30 - 28\exp(-0.06 \times 0.5)] \approx \$2.83.$$

假定股票的欧式看涨价格为$\$2.50$,这比理论最小值$\$2.83$要小,套利者可以买该看涨期权并卖空股票,这样便产生了一个新的现金流\$(30-2.50)=\$27.50.如果以无风险利率投资6个月,则\$27.50变为$\$27.50\exp(0.06×0.5)=\$28.34$.在到期日,如果$P_T>\$28$,则套期保值者执行期权,并平了空头头寸,他获利$\$(28.34-28)=\$0.34$.另一方面,如果$P_T<\$28$,套利者可以从市场直接买股票平仓,他甚至可以获得更多的利益.作为说明,假定$P_T=\$27.00$,则获利将是$\$(28.34—27)=\$1.34$.

4 讨论

 由公式知,一个看涨或看跌期权的价格依赖于5个变量,即当前的股价P_t、敲定价格K、以年度量的到期日$T-t$、年波动率σ。以及年利率r.值得研究的是这5个变量对期权价格的影响.

1.边际效应.

首先考虑这5个变量对一个看涨期权价格c_t的边际效应.边际效应的意思是在固定其他变量的情况下改变其中一个变量.一个看涨期权的效应可以概括如下.

(1)当前价格P_T与$ln(P_t)$正相关,特别地,当$P_t\to0$时,$c_t\to0$;当$P_t\to\infty$时,$c_t\to\infty$.6-3a解释了$K=80$,年利率$r=6%$;,一老=0.25$T-t=0.25$年以及年波动率$\sigma=30\%$时的效应.

(2)执行价格K c_t与$ln(K)$负相关.具体来讲,当$K\to0$时,$c_t\to P_t$当$K\to\infty$时,$c_t\to0$.

图$6-3$当前股价对期权价格的边际效应,其中$K=80, T-t=0.25, \sigma=0.3,$
$r=0.06;$ (a)看涨期权; (b)看跌期权

(3)到期时间 c_t与$T-t$的相关性非常复杂,但是通过将h_+和h_-写成

$$h_+ = \frac{ln(P_t/K)}{\sigma\sqrt{T-t}} + \frac{(r+\sigma^2/2)\sqrt{T-t}}{\sigma},$$
$$h_- = \frac{ln(P_t/K)}{\sigma\sqrt{T-t}} + \frac{(r-\sigma^2/2)\sqrt{T-t}}{\sigma}.$$

可得到极限结果.若P_tK,则当$(T-t)\to0$时$c_t\to P_t-K$,且当$(T-t)\to\infty$时$c_t\to P_t$、图$6-4a$显示的是对三种不同的当前股价来说,$T-t$对c_t的边际效应.固定的变量是$K=80,r=0.06$和\sigma=0.3$.实线、点划线以及虚线分别对应于$P_t=70,80,90$.

$ (4)波动率σ通过将$h+*和h-$改写成

$$h_+ = \frac{ln(P_t/K) + r(T-t)}{\sigma\sqrt{T-t}} + \frac{\sigma}{2}\sqrt{T-t},$$
$$h_- = \frac{ln(P_t/K) + r(T-t)}{\sigma\sqrt{T-t}} - \frac{\sigma}{2}\sqrt{T-t}.$$

我们得到(a)如果$ln(P_t/K)+r(T-t)<0$,则当$\sigma\to0$时,$c_t\to0$;(b)如果$ln(P_t/K)+r(T—t)\geq0$,则当$\sigma\to0$时,$c_t\to P_t-Ke^{-r(T-t)}$;当$\sigma\to \infty$时,

$c_t\to P_t$.图6-5a表明$K=80,T-t=0.25,r=0.06.$以及P_t取3个不同值时σ对

c_t的效应.实线、点划线以及虚线分别对应于$P_t=70,80,90$.

(5)利率ct与r是正相关的,满足:当$r\to\infty$时,$c_t\to P$

这5个变量对一个看跌期权的边际效应可类似得到图6-3b、图6-4b和图6-5b对所选择的一些情况解释了其效应.

2.一些联合效应

波动率与敲定价格对一个看涨期权的联合效应,这里其他变量是固定的,$P_t=80,r=6\%,{T-t}=0.25$.正如所预料的,当波动率很高而且敲定价格正好低于当前股价时,看涨期权的价格更高,图6-7显示同样条件下,对一个看跌期权的效应.当波动率很高而且敲定价格正好高于当前的股价时,看跌期权的价格更高、而且,图形也说明了随着波动率的增加,敲定价格对看跌期权价格的效应将变得更加线性化.

In [ ]: